



山东大学
SHANDONG UNIVERSITY

XGBoost 算法阅读报告 (2018 届)

学院：控制科学与工程学院

专业：自动化

姓名：范钧宇

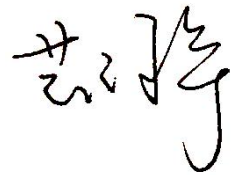
学号：201800171021

完成时间：2021 年 5 月 7 日

原创性声明

兹提交的实验报告，是本人在指导老师指导下独立完成的成果。本人在设计中参考的其他个人或集体的成果，均在设计作品文字说明中以明确方式标明。本人依法享有和承担由此设计作品而产生的权利和责任。

声明人（签名）：

A handwritten signature in black ink, appearing to be '赵玲' (Zhao Ling), written in a cursive style.

2021 年 5 月 7 日

目录

一、xgboost 是什么.....	1
二、基础知识, GBDT.....	1
三、xgboost 算法原理知识.....	2
1. 定义树的复杂度.....	2
2. xgboost 中的 boosting tree 模型.....	3
3. 对目标函数的改写.....	3
四、xgboost 的改进点总结.....	15
五、参数.....	18
六、补充.....	21

一、xgboost 是什么

全称：eXtreme Gradient Boosting

作者：陈天奇(华盛顿大学博士)

基础：GBDT

所属：boosting 迭代型、树类算法。

适用范围：分类、回归

优点：速度快、效果好、能处理大规模数据、支持多种语言、支持自定义损失函数等等。

缺点：发布时间短（2014），工业领域应用较少，待检验

xgboost 是大规模并行 boosted tree 的工具，它是目前最快最好的开源 boosted tree 工具包，比常见的工具包快 10 倍以上。在数据科学方面，有大量 kaggle 选手选用它进行数据挖掘比赛，其中包括两个以上 kaggle 比赛的夺冠方案。在工业界规模方面，xgboost 的分布式版本有广泛的移植性，支持在 YARN, MPI, Sungrid Engine 等各个平台上面运行，并且保留了单机并行版本的各种优化，使得它可以很好地解决于工业界规模的问题。

二、基础知识，GBDT

xgboost 是在 GBDT 的基础上对 boosting 算法进行的改进，内部决策树使用的是回归树，简单回顾 GBDT 如下：



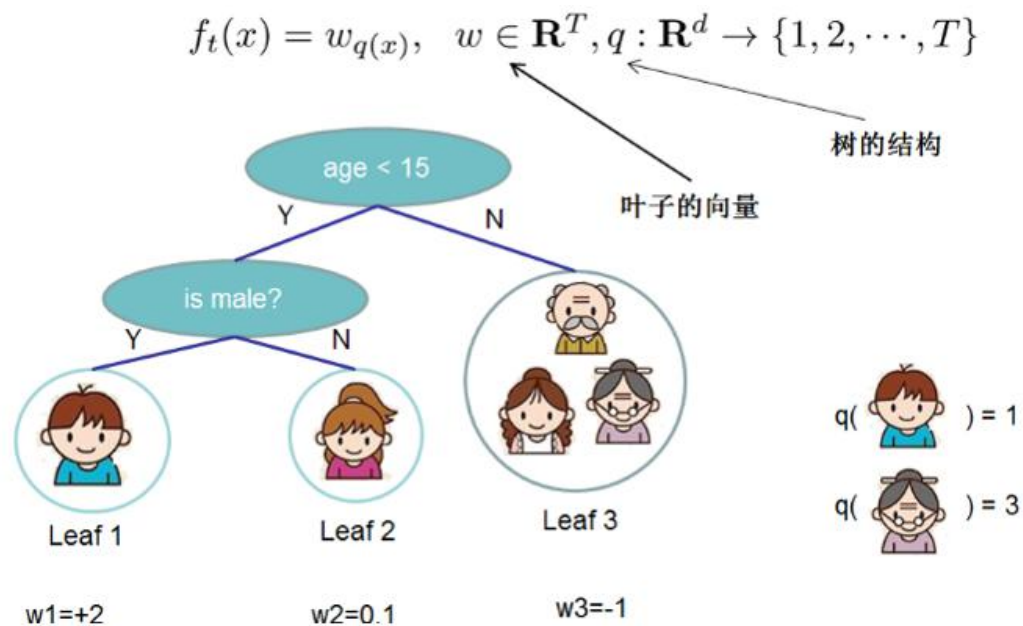
回归树的分裂结点对于平方损失函数，拟合的就是残差；对于一般损失函数（梯度下降），拟合的就是残差的近似值，分裂结点划分时枚举所有特征的值，选取划分点。

最后预测的结果是每棵树的预测结果相加。

三、xgboost 算法原理知识

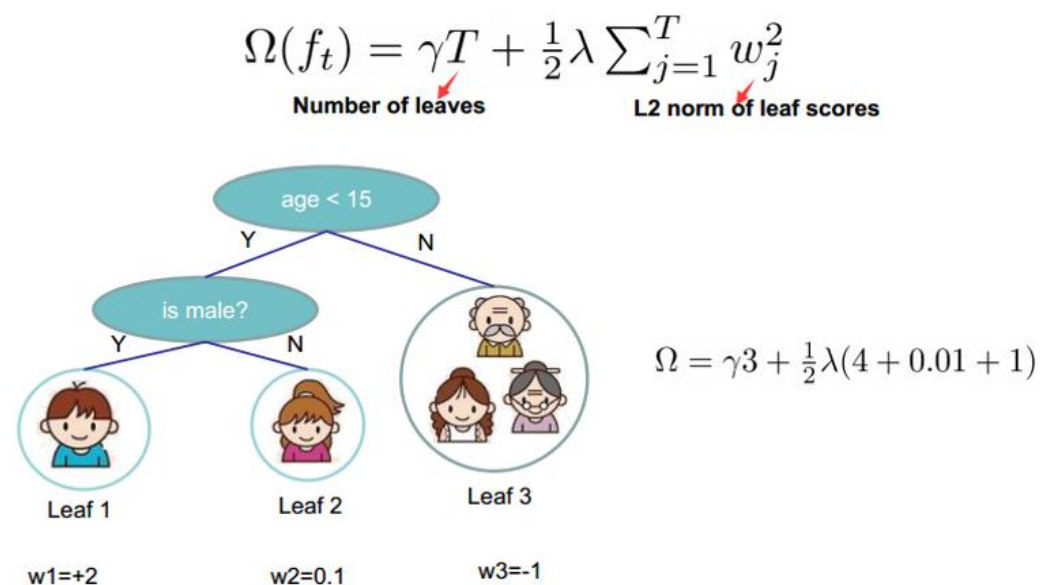
提升方法是一种非常有效的机器学习方法，在前几篇笔记中介绍了提升树与 GBDT 基本原理，xgboost (eXtreme Gradient Boosting) 可以说是提升方法的完全加强版本。xgboost 算法在各大比赛中展现了强大的威力。

1. 定义树的复杂度



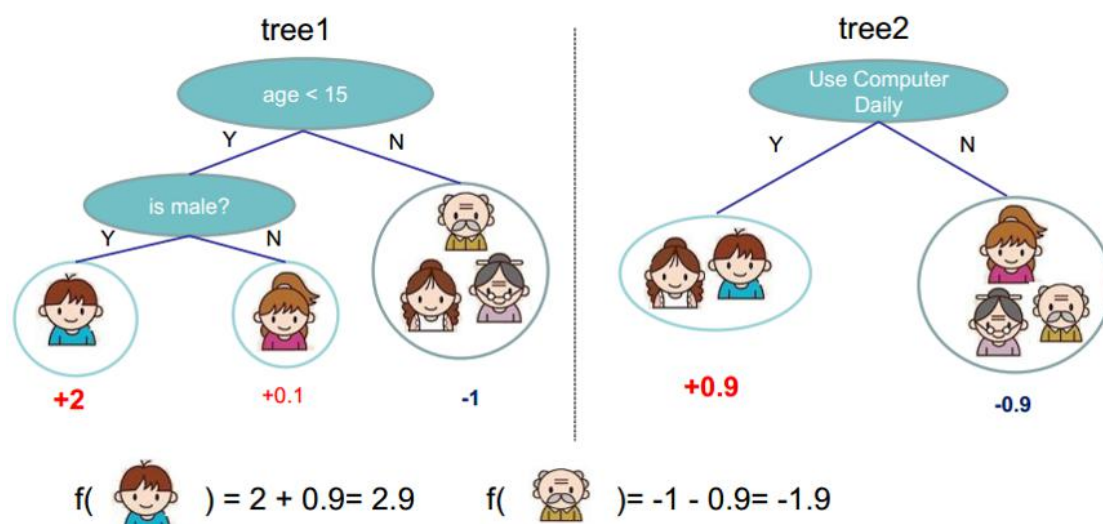
把树拆分成结构部分 q 和叶子权重部分 w 。

树的复杂度函数和样例：



定义树的结构和复杂度的原因很简单，这样就可以衡量模型的复杂度了啊，从而可以有效控制过拟合。

2.xgboost 中的 boosting tree 模型



Prediction of is sum of scores predicted by each of the tree

和传统的 boosting tree 模型一样，xgboost 的提升模型也是采用的残差（或梯度负方向），不同的是分裂结点选取的时候不一定是最小平方损失。

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$$

3.对目标函数的改写

- Goal $Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$
 - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
 - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
 - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

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

最终的目标函数只依赖于每个数据点的在误差函数上的一阶导数和二阶导数。这么写的原因很明显，由于之前的目标函数求最优解的过程中只对平方损失函数时候方便求，对于其他的损失函数变得很复杂，通过二阶泰勒展开式的变换，这样求解其他损失函数变得可行了。

当定义了分裂候选集合的时候， $I_j = \{i | q(x_i) = j\}$ 可以进一步改目标函数。分裂结点的候选响集是很关键的一步，这是 xgboost 速度快的保证，怎么选出来这个集合，后面会介绍。

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


求解：

$$\text{定义 } G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$$

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

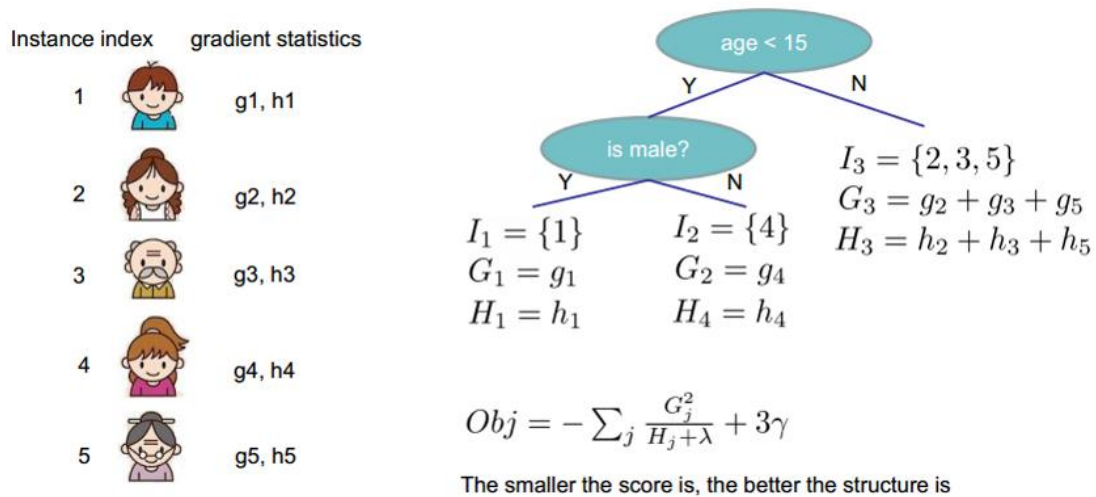
对 w_j 求导等于0得：

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

 This measures how good a tree structure is!

4. 树结构的打分函数

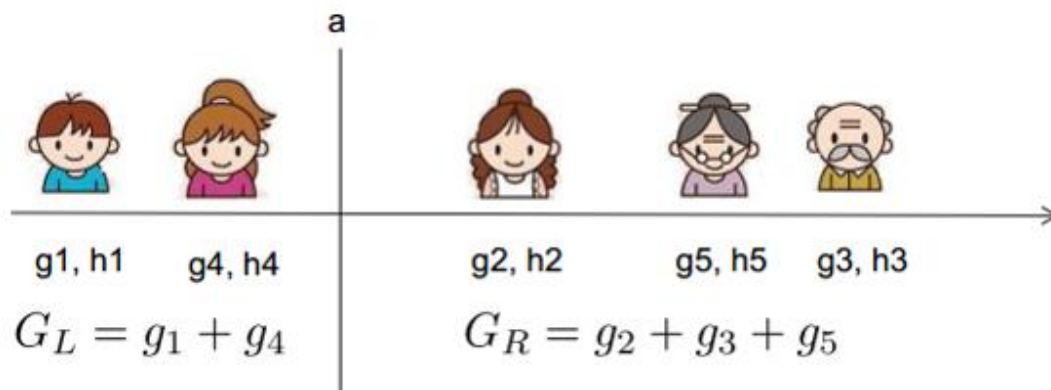
Obj 代表了当指定一个树的结构的时候，在目标上面最多减少多少。(structure score)



对于每一次尝试去对已有的叶子加入一个分割

$$Gain = \underbrace{\frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} \right]}_{\text{左子树分数}} + \underbrace{\frac{1}{2} \left[\frac{G_R^2}{H_R + \lambda} \right]}_{\text{右子树分数}} - \underbrace{\frac{1}{2} \left[\frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right]}_{\text{不分割我们可以拿到的分数}} - \underbrace{\gamma}_{\text{加入新叶子节点引入的复杂度代价}}$$

这样就可以在建树的过程中动态的选择是否要添加一个结点。



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 左边和右边的导数和。对于所有的 a ，我们只要做一遍从左到右的扫描就可以枚举出所有分割的梯度和 G_L 、 G_R 。然后用上面的公式计算每个分割方案的分数就可以了。

5.寻找分裂结点的候选集

(1) 暴力枚举

(2) 近似方法

近似方法通过特征的分布，按照百分比确定一组候选分裂点，通过遍历所有的候选分裂点来找到最佳分裂点。

两种策略：全局策略和局部策略。在全局策略中，对每一个特征确定一个全局的候选分裂点集合，就不再改变；而在局部策略中，每一次分裂都要重选一次分裂点。前者需要较大的分裂集合，后者可以小一点。对比补充候选集策略与分裂点数目对模型的影响。全局策略需要更细的分裂点才能和局部策略差不多

(3) Weighted Quantile Sketch

- 对第k个特征，构造数据集 $D_k = (x_{1k}, h_1), (x_{2k}, h_2), \dots, (x_{nk}, h_n)$
- h_i 是该数据点对应的损失函数的二阶梯度
- 定义序函数为带权的序函数

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

- 上式代表第k个特征小于z的样本比例
- 候选集的目标要使得相邻的两个候选分裂结点相差不超过阈值 ϵ

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \quad s_{k1} = \min_i x_{ik}, \quad s_{kl} = \max_i x_{ik}.$$

陈天奇提出并从概率角度证明了一种带权重的分布式的 Quantile Sketch。

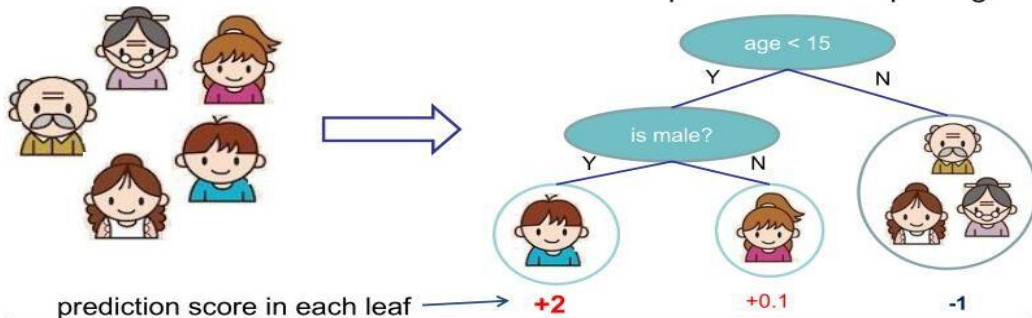
6. Regression Tree and Ensemble (What are we Learning, 得到学习目标)

(1) Regression Tree (CART) 回归树

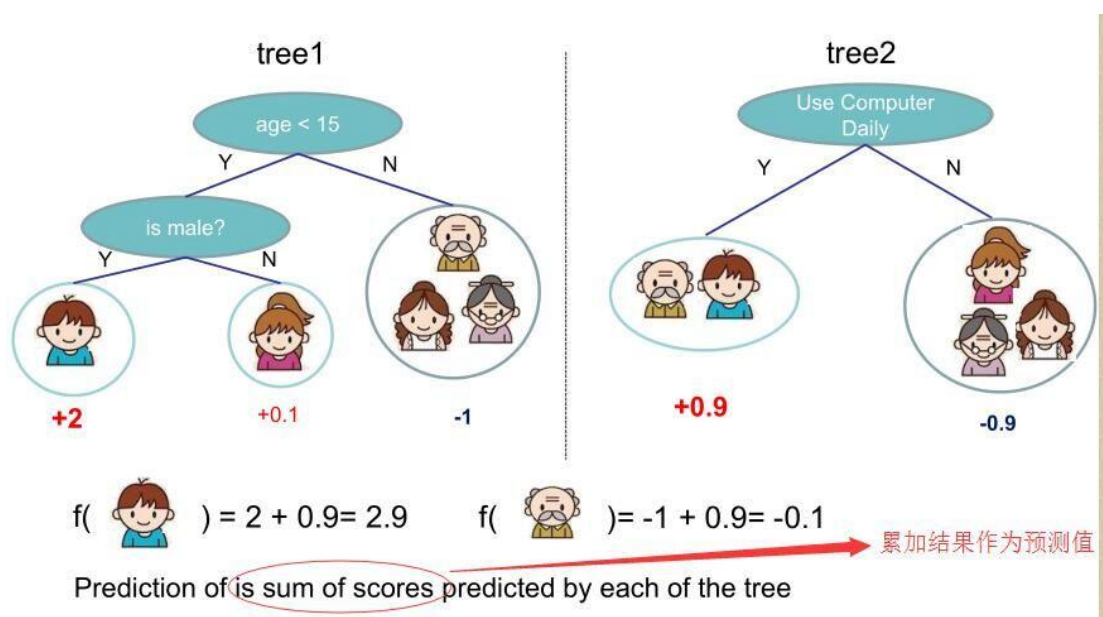
- regression tree (also known as classification and regression tree):
 - Decision rules same as in decision tree
 - Contains one score in each leaf value

Input: age, gender, occupation, ...

Does the person like computer games



(2) Regression Tree Ensemble 回归树集成



在上面的例子中，我们用两棵树来进行预测。我们对于每个样本的预测结果就是每棵树预测分数的和。

(3) Objective for Tree Ensemble 得到学习目标函数

- Model: assuming we have K trees

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

- Objective

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

Training loss

Complexity of the Trees

这里是构造一个目标函数，然后我们要做的就是去尝试优化这个目标函数。读到这里，感觉与 gbdt 好像没有什么区别，确实如此，不过在后面就能看到他们的不同了（构造（学习）模型参数）。

7.Gradient Boosting (How do we Learn, 如何学习)

(1) So How do we Learn?

$$\sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$$

We can not use methods such as SGD, to find f (since they are trees, instead of just numerical vectors)

Solution: Additive Training (Boosting)

Start from constant prediction, add a new function each time

- Start from constant prediction, add a new function each time

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

Model at training round t

Keep functions added in previous round

这里理解很关键，这里目标函数优化的是整体的模型， y_i' 是整个累加模型的输出，正则化项是所有树的复杂度之和，这个复杂度组成后面（6）会讲。这种包含树的模型不适合直接用 SGD 等优化算法直接对整体模型进行优化，因而采用加法学习方法，boosting 的学习策略是每次学习当前的树，找到当前最佳的树模型加入到整体模型中，因此关键在于学习第 t 棵树。

(2) Additive Training : 定义目标函数，优化，寻找最佳的 f_t 。

How do we decide which f to add?

Optimize the objective!! 目标优化

- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
 This is what we need to decide in round t

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

Goal: find f_t to minimize this

- Consider square loss

$$\begin{aligned}Obj^{(t)} &= \sum_{i=1}^n \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right)^2 + \Omega(f_t) + \text{const} \\ &= \sum_{i=1}^n \left[2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2 \right] + \Omega(f_t) + \text{const}\end{aligned}$$

残差

This is usually called residual from previous round

如图所示，第 t 轮的模型预测等于前 $t-1$ 轮的模型预测 $y^{(t-1)}$ 加上 f_t ，因此误差函数项记为 $l(y_i, y^{(t-1)} + f_t)$ ，后面一项为正则化项。

在当前步, y_i 以及 $y^{(t-1)}$ 都是已知值, 模型学习的是 f_t 。

(3) Taylor Expansion Approximation of Loss 对误差函数进行二阶泰勒近似展开

- Goal $Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$
 - Seems still complicated except for the case of square loss

- Take Taylor expansion of the objective

- Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$ 保留二次项
- Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

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

- If you are not comfortable with this, think of square loss

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 (\hat{y}^{(t-1)} - y_i)^2 = 2$$

- Compare what we get to previous slide

把平方损失函数的一二次项带入原目标函数, 你会发现与之前那张 ppt 的损失函数是一致的。

至于为什么要这样展开呢, 这里就是 xgboost 的特点了, 通过这种近似, 你可以自定义一些损失函数(只要保证二阶可导), 树分裂的打分函数是基于 g_i, h_i (G_j, H_j) 计算的。

(4) Our New Goal 得到了新的目标函数

- Objective, with constants removed

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

- where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

从这里就可以看出 xgboost 的不同了, 目标函数保留了泰勒展开的二次项。

• Why spending s much efforts to derive the objective, why not just grow trees ...

- Theoretical benefit: know what we are learning, convergence
- **Engineering** benefit, recall the elements of supervised learning
 - g_i and h_i comes from definition of loss function
 - The learning of function only depend on the objective via g_i and h_i
 - Think of how you can separate modules of your code when you are asked to implement boosted tree for both square loss and logistic loss

<http://blog.csdn.net/sb19931201>

(5) Refine the definition of tree 重新定义每棵树

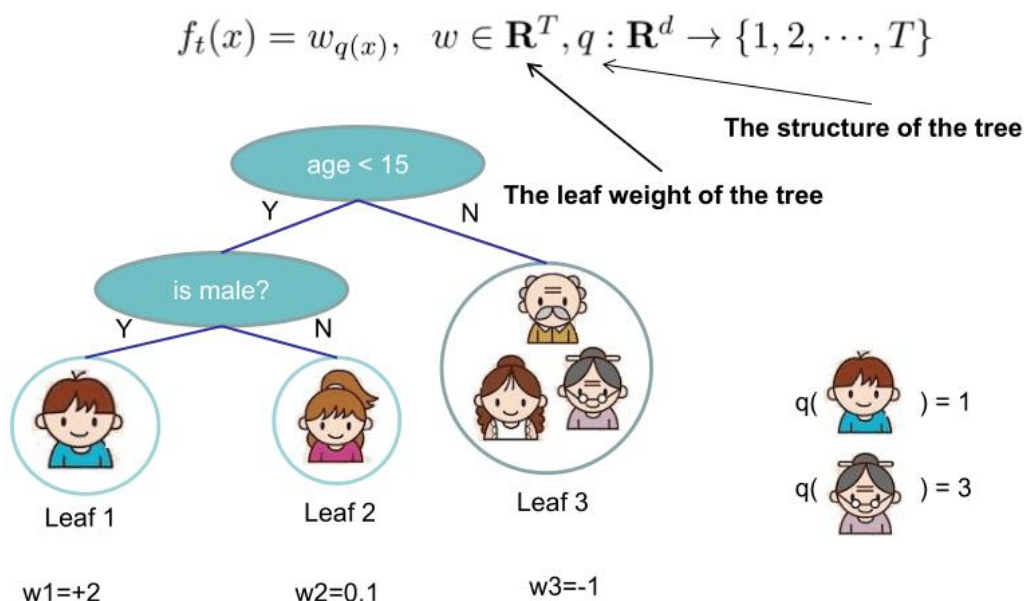
We define tree by a vector of scores in leafs, and a leaf index

用叶子节点集合以及叶子节点得分表示

mapping function that maps an instance to a leaf

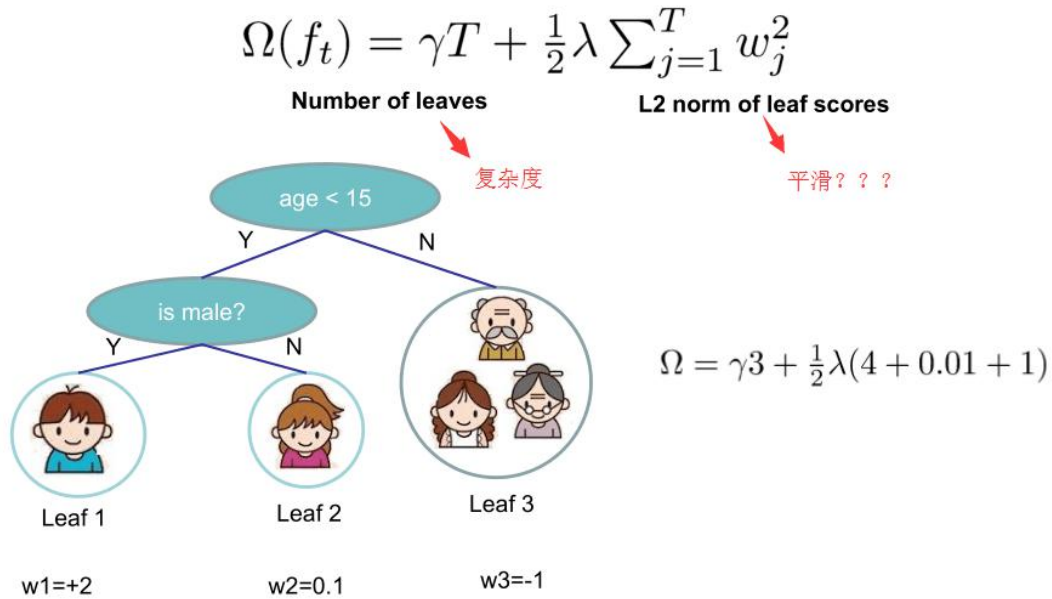
每个样本都落在一个叶子节点上

$q(x)$ 表示样本 x 在某个叶子节点上, $w_q(x)$ 是该节点的打分, 即该样本的模型预测值。



(6) Define the Complexity of Tree 树的复杂度项

Define complexity as (this is not the only possible definition)



从图中可以看出，xgboost 算法中对树的复杂度项包含了两个部分，一个是叶子节点总数，一个是叶子节点得分 L2 正则化项，针对每个叶结点的得分增加 L2 平滑，目的也是为了避免过拟合。

(7) Revisit the Objectives 更新

- Define the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$
- Regroup the objective by each leaf 根据叶结点重新组合目标函数

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

- This is sum of T independent quadratic functions

独立的二次函数的和

注意，这里优化目标的 $n \rightarrow T$, T 是叶子数。

论文中定义了：Define $I_j = \{i | q(x_i) = j\}$ as the instance set of leaf j . 这一步是由于 xgb 加了两个正则项，一个是叶子节点个数 (T)，一个是叶节点分数 (w)。原文中的等式 4，加了正则项的目标函数里就出现了两种累加，一种是 $i \rightarrow n$ (样本数)，一种是 $j \rightarrow T$ (叶子节点数)。这步转换是为了统一目标函数中的累加项， I_j 是每个叶节点 j 上的样本集合。

(8) The Structure Score 这个 score 是用来评价树结构的。根据目标函数得到（见论文公式(4)、(5)、(6)），用于切分点查找算法。

Define $I_j = \{i | q(\mathbf{x}_i) = j\}$ as the instance set of leaf j . We can rewrite Eq (3) by expanding the regularization term Ω as follows

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^n [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T\end{aligned}\quad (4)$$

For a fixed structure $q(\mathbf{x})$, we can compute the optimal weight w_j^* of leaf j by

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \quad (5)$$

and calculate the corresponding optimal objective function value by

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^T \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \quad (6)$$

- Two facts about single variable quadratic function

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

- Let us define $G_j = \sum_{i \in I_j} g_i$ $H_j = \sum_{i \in I_j} h_i$

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

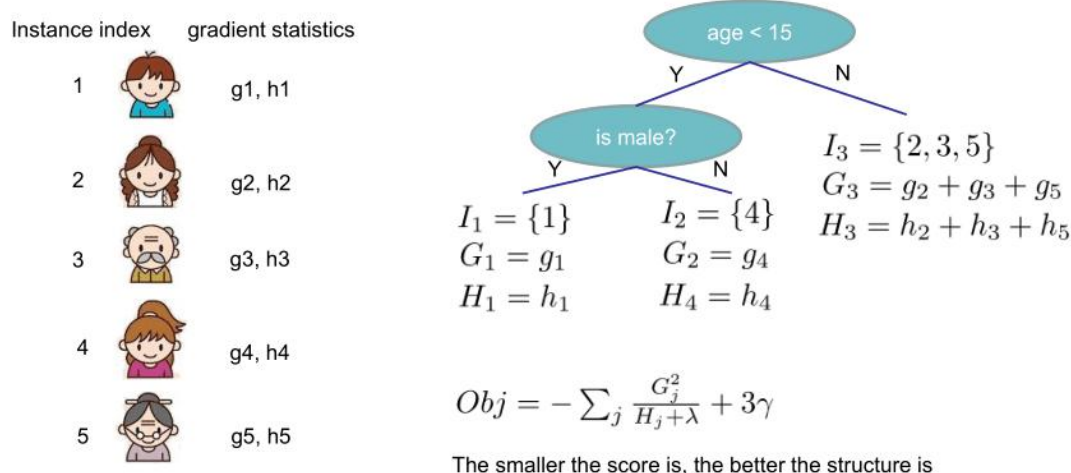
- Assume the structure of tree ($q(\mathbf{x})$) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

This measures how good a tree structure is!

越大越好，总体损失越小

The Structure Score Calculation:



(8) 切分点查找算法

- In practice, we grow the tree greedily
 - Start from tree with depth 0
 - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

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

Annotations:

- $\frac{G_L^2}{H_L + \lambda}$: the score of left child
- $\frac{G_R^2}{H_R + \lambda}$: the score of right child
- $\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}$: the score of if we do not split
- γ : The complexity cost by introducing additional leaf

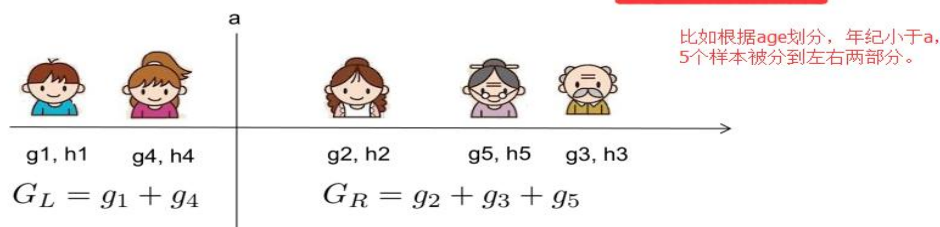
未切分前该父结点的分数值

切分后左右子树的得分之和

- Remaining question: how do we find the best split?

如上图可见 Gain 还加了一个叶子节点复杂度项，有点类似 CART 的剪枝。

- What is the gain of a split rule $x_j < a$? Say x_j is age



- All we need is sum of g and h in each side, and calculate

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

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

上图中 G 都是各自区域内的 g_i 总和，根据 $\text{Gain}(\max)$ 选择最优分割点。此外，作者针对算法设计对特征进行了排序，分位点划分等，有兴趣的可以阅读原始论文，这里不做详解。

算法步骤：

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

根据特征划分有无数可能的树结构，因此采用近似算法（特征分位点，候选分割点）

Algorithm 2: Approximate Algorithm for Split Finding

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

 Propose $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ 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.

(9) 小结 Boosted Tree Algorithm

- Add a new tree in each iteration

- Beginning of each iteration, calculate

$$\underbrace{g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})}_{\text{一阶}}, \quad \underbrace{h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})}_{\text{二阶}}$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T \quad \text{贪心算法寻找切分点，生产每一轮新的树}$$

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

- Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$ 称之为：缩减因子 同样为了避免过拟合
- ϵ is called step-size or shrinkage, usually set around 0.1
- This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

四、xgboost 的改进点总结

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 L1 和 L2 正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。一可以通过 booster [default=gbtrees] 设置参数:gbtree: tree-based models/gblinear: linear models

2.传统 GBDT 在优化时只用到一阶导数信息，xgboost 则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。

xgboost 工具支持自定义代价函数，只要函数可一阶和二阶求导。一对损失函数做了改进（泰勒展开，一阶信息 g 和二阶信息 h, 上一章节有做介绍）

3.xgboost 在代价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 的 L2 模的平方和。从 Bias-variance tradeoff 角度来讲，正则项降低了模型 variance，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

—正则化包括了两个部分，都是为了防止过拟合，剪枝是都有的，叶子结点输出 L2 平滑是新增的。

4.shrinkage and column subsampling —还是为了防止过拟合

(1) **shrinkage** 缩减类似于学习速率，在每一步 **tree boosting** 之后增加了一个参数 **n**（权重），通过这种方式来减小每棵树的影响力，给后面的树提供空间去优化模型。

(2) **column subsampling** 列(特征)抽样，说是从随机森林那边学习来的，防止过拟合的效果比传统的行抽样还好（行抽样功能也有），并且有利于后面提到的并行化处理算法。

5.split finding algorithms(划分点查找算法):

(1) **exact greedy algorithm**—贪心算法获取最优切分点

(2) **approximate algorithm**— 近似算法，提出了候选分割点概念，先通过直方图算法获得候选分割点的分布情况，然后根据候选分割点将连续的特征信息映射到不同的 **buckets** 中，并统计汇总信息。详细见论文 3.3 节

(3) **Weighted Quantile Sketch**—分布式加权直方图算法，论文 3.4 节

这里的算法（2）、（3）是为了解决数据无法一次载入内存或者在分布式情况下算法（1）效率低的问题

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。

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。——稀疏感知算法，论文 3.4 节，Algorithm 3: Sparsity-aware Split Finding

7.Built-in Cross-Validation（内置交叉验证）

XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

This is unlike GBM where we have to run a grid-search and only a limited values can be tested.

8.continue on Existing Model（接着已有模型学习）

User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications.

GBM implementation of sklearn also has this feature so they are even on this point.

9.High Flexibility（高灵活性）

XGBoost allow users to define custom optimization objectives and evaluation criteria.

This adds a whole new dimension to the model and there is no limit to what we can do.

10.并行化处理 ——系统设计模块,块结构设计等

xgboost 工具支持并行。boosting 不是一种串行的结构吗?怎么并行的? 注意 **xgboost** 的并行不是 tree 粒度的并行, **xgboost** 也是一次迭代完才能进行下一次迭代的（第 t 次迭代的代价函数里包含了前面 $t-1$ 次迭代的预测值）。**xgboost** 的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特

征的值进行排序（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。

此外 xgboost 还设计了高速缓存压缩感知算法，这是系统设计模块的效率提升。当梯度统计不适合于处理器高速缓存和高速缓存丢失时，会大大减慢切分点查找算法的速度。

（1）针对 **exact greedy algorithm** 采用缓存感知预取算法

（2）针对 **approximate algorithms** 选择合适的块大小

五、参数

1.General Parameters（常规参数）

（1）**booster [default=gbtree]:**

选择基分类器，gbtree: tree-based models/gblinear: linear models

（2）**silent [default=0]:**

设置成 1 则没有运行信息输出，最好是设置为 0.

（3）**nthread**

[default to maximum number of threads available if not set]: 线程数

2.Booster Parameters（模型参数）

（1）**eta [default=0.3]**

shrinkage 参数，用于更新叶子节点权重时，乘以该系数，避免步长过大。参数值越大，越可能无法收敛。把学习率 eta 设置的小一些，小学习率可以使得后面的学习更加仔细。

（2）**min_child_weight [default=1]:**

这个参数默认是 1，是每个叶子里面 h 的和至少是多少，对正负样本不均衡时

的 0-1 分类而言, 假设 h 在 0.01 附近, `min_child_weight` 为 1 意味着叶子节点中最少需要包含 100 个样本。这个参数非常影响结果, 控制叶子节点中二阶导的和的最小值, 该参数值越小, 越容易 overfitting。

(3) `max_depth [default=6]:`

每颗树的最大深度, 树高越深, 越容易过拟合。

(4) `max_leaf_nodes:`

最大叶结点数, 与 `max_depth` 作用有点重合。

(5) `gamma [default=0]:`

后剪枝时, 用于控制是否后剪枝的参数。

(6) `max_delta_step [default=0]:`

这个参数在更新步骤中起作用, 如果取 0 表示没有约束, 如果取正值则使得更新步骤更加保守。可以防止做太大的更新步子, 使更新更加平缓。

(7) `subsample [default=1]:`

样本随机采样, 较低的值使得算法更加保守, 防止过拟合, 但是太小的值也会造成欠拟合。

(8) `colsample_bytree [default=1]:`

列采样, 对每棵树的生成用的特征进行列采样. 一般设置为: 0.5-1

(9) `lambda [default=1]:`

控制模型复杂度的权重值的 L2 正则化项参数, 参数越大, 模型越不容易过拟合。

(10) `alpha [default=0]:`

控制模型复杂程度的权重值的 L1 正则项参数, 参数值越大, 模型越不容易过拟合。

(11) `scale_pos_weight [default=1]:`

如果取值大于 0 的话, 在类别样本不平衡的情况下有助于快速收敛。

3.Learning Task Parameters (学习任务参数)

(1) `objective [default=reg:linear]:`

定义最小化损失函数类型，常用参数：

`binary:logistic` - logistic regression for binary classification, returns predicted probability (not class)

`multi:softmax` - multiclass classification using the softmax objective, returns predicted class (not probabilities)

you also need to set an additional `num_class` (number of classes) parameter defining the number of unique classes

`multi:softprob` - same as softmax, but returns predicted probability of each data point belonging to each class.

(2) eval_metric [default according to objective]:

The metric to be used for validation data.

The default values are rmse for regression and error for classification.

Typical values are:

`rmse` - root mean square error

`mae` - mean absolute error

`logloss` - negative log-likelihood

`error` - Binary classification error rate (0.5 threshold)

`merror` - Multiclass classification error rate

`mlogloss` - Multiclass logloss

`auc`: Area under the curve

(3) seed [default=0]:

The random number seed. 随机种子，用于产生可复现的结果

Can be used for generating reproducible results and also for parameter tuning.

注意: python sklearn style 参数名会有所变化

`eta` -> `learning_rate`

`lambda` -> `reg_lambda`

`alpha` -> `reg_alpha`

六、补充

微软出了个 LightGBM,号称性能更强劲,速度更快。简单实践了一波,发现收敛速度要快一些,不过调参还不好,没有权威。

主要是两个:

① histogram 算法替换了传统的 Pre-Sorted,某种意义上是牺牲了精度(但是作者声明实验发现精度影响不大)换取速度,直方图作差构建叶子直方图挺有创造力的。(xgboost 的分布式实现也是基于直方图的,利于并行)

② 带有深度限制的按叶子生长 (leaf-wise) 算法代替了传统的(level-wise) 决策树生长策略,提升精度,同时避免过拟合危险。

附录

1.xgboost 入门与实战(原理篇):[https://blog.csdn.net/sb19931201/article/details/](https://blog.csdn.net/sb19931201/article/details/52557382)

52557382

2.终于有人说清楚了--XGBoost 算法:[https://www.cnblogs.com/mantch/p/1116422](https://www.cnblogs.com/mantch/p/11164221.html)

1.html