

# 280412xgboost

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）。—可以通过booster [default=gbtree]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。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。—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，叶子节点输出L2平滑是新增的。

4.shrinkage and column subsampling —还是为了**防止过拟合**，论文2.3节有介绍，这里答主已概括的非常到位（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 $\eta$ （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)：—理解的还不够透彻，需要进一步学习（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，以下引用的还是wepon大神的总结：

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

我觉得关于xgboost并行化设计仅仅从论文PPT博客上学习是远远不够的,有时间还要从代码层面去学习分布式 xgboost的设计理念。

先附上北大weapon 大神的一些理解:

回复 @肖岩在评论里的问题,因为有些公式放正文比较好。评论里讨论的问题的大意是“xgboost代价函数里加入正则项,是否优于cart的剪枝”。其实陈天奇大神的slides里面也是有提到的,我当一下搬运工。决策树的学习过程就是为了找出最优的决策树,然而从函数空间里所有的决策树中找出最优的决策树是NP-C问题,所以常采用启发式(Heuristic)的方法,如CART里面的优化GINI指数、剪枝、控制树的深度。这些启发式方法的背后往往隐含了一个目标函数,这也是大部分人经常忽视掉的。xgboost的目标函数如下:

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

Training lossComplexity of the Trees

其中正则项控制着模型的复杂度,包括了叶子节点数目T和leaf score的L2模的平方:

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

Number of leavesL2 norm of leaf scores

那这个跟剪枝有什么关系呢??? 跳过一系列推导,我们直接来看xgboost中树节点分裂时所采用的公式:

- For each leaf node of the tree, try to add a split. The change of objective after adding the split is

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

the score of left childthe score of right childthe score of if we do not splitThe complexity cost by introducing additional leaf

这个公式形式上跟ID3算法（采用entropy计算增益）、CART算法（采用gini指数计算增益）是一致的，都是用分裂后的某种值 减去 分裂前的某种值，从而得到增益。为了限制树的生长，我们可以加入阈值，当增益大于阈值时才让节点分裂，上式中的gamma即阈值，它是正则项里叶子节点数T的系数，所以xgboost在优化目标函数的同时**相当于做了预剪枝**。另外，上式中还有一个系数lambda，是正则项里leaf score的L2模平方的系数，对leaf score做了平滑，也起到了防止过拟合的作用，这个是传统GBDT里不具备的特性。

## xgboost使用经验总结

- 多类别分类时，类别需要从0开始编码
- Watchlist不会影响模型训练。
- 类别特征必须编码，因为xgboost把特征默认都当成数值型的
- 调参： [Notes on Parameter Tuning](#) 以及 [Complete Guide to Parameter Tuning in XGBoost \(with codes in Python\)](#)
- 训练的时候，为了结果可复现，记得**设置随机数种子**。
- XGBoost的特征重要性是如何得到的？某个**特征的重要性（feature score）**，等于它被选中为树节点分裂特征的次数的和，比如特征A在第一次迭代中（即第一棵树）被选中了1次去分裂树节点，在第二次迭代被选中2次.....那么最终特征A的feature score就是 1+2+....

原文链接： <http://wepon.me/>

---

原文： <https://blog.csdn.net/sb19931201/article/details/52557382>

## 1- 前言

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可移植性，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。

花了几天时间粗略地看完了xgboost原论文和作者的slide讲解，仅仅是入门入门入门笔记。给我的感觉就是xgboost算法比较复杂，针对传统GBDT算法做了很多细节改进，包括损失函数、正则化、切分点查找算法优化、稀疏感知算法、并行化算法设计等等。本文主要介绍xgboost基本原理以及与传统gbdt算法对比总结，后续会基于python版本做了一些实战调参试验。想详细学习xgboost算法原理建议通读作者原始论文与slide讲解。

**相关文献资料：** [Xgboost Slides](#) [XGBoost中文版原理介绍](#) [原始论文XGBoost: A Scalable Tree Boosting System](#) [XGBoost Parameters \(official guide\)](#)

**精彩博文：** [XGBoost深入浅出——wepon](#) [xgboost: 速度快效果好的boosting模型](#) [Complete Guide to Parameter Tuning in XGBoost \(with codes in Python\)](#)

[XGBoost Plotting API](#)以及[GBDT组合特征实践](#)

## 2- 补充！LightGBM！

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6，没有权威。看了GitHub上的介绍以及知乎上的一些回答，大致理解了性能提升的原因。主要是两个：

①histogram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。

细节大家直接看作者的解释以及GitHub上的介绍吧,还是挺好理解的~ 链接：

<https://www.zhihu.com/question/51644470/answer/130946285>

<https://github.com/Microsoft/LightGBM/wiki/Features>

## 3- xgboost基本原理介绍

### 3.1 简介

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。

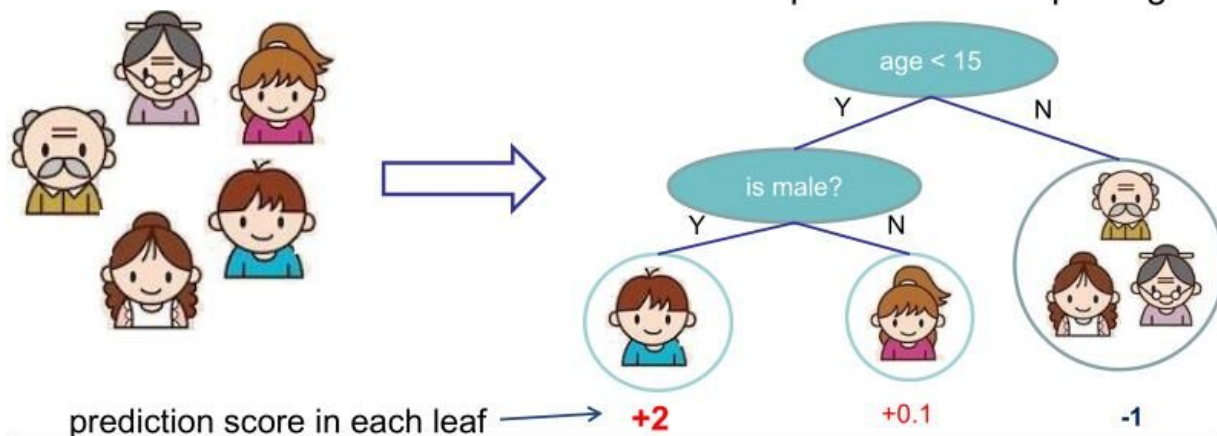
### 3.2 Regression Tree and Ensemble (What are we Learning , 得到学习目标)

#### ( 1 ) Regression Tree (CART)回归树

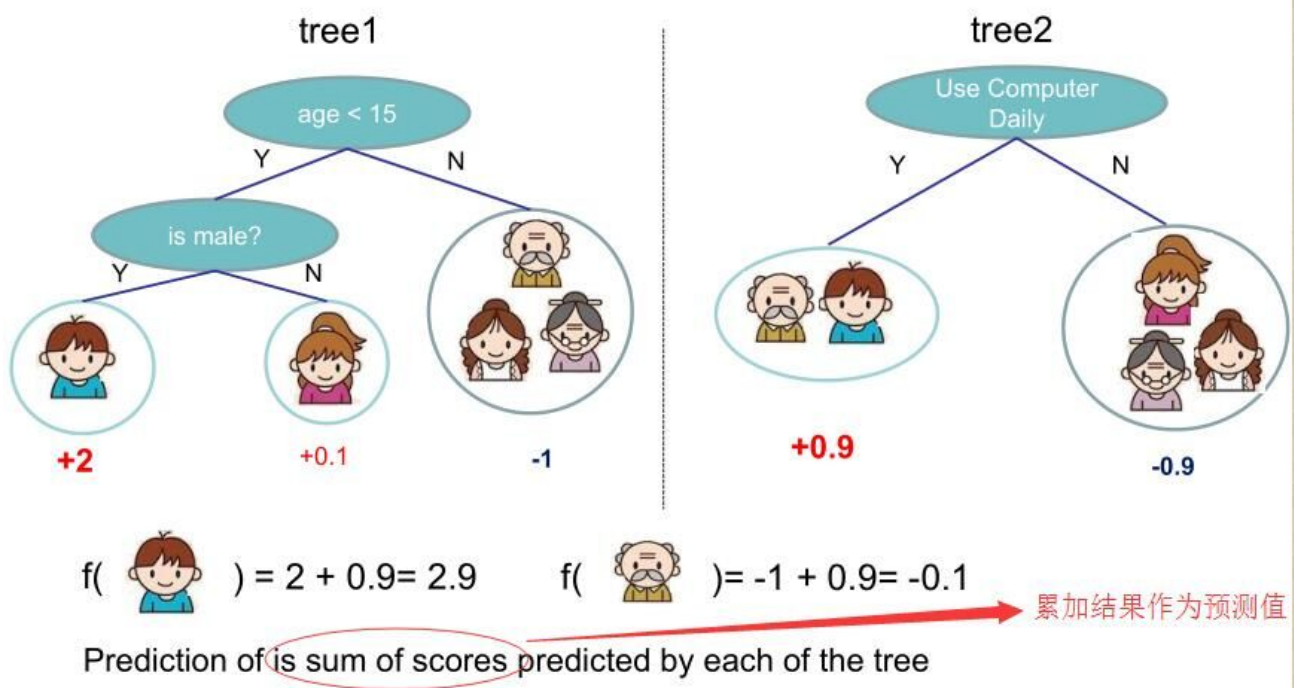
- regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value

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）。

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

So How do we Learn?

目标函数：

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



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。

## 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  $t$

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

Goal: find  $f_t$  to minimize this

- Consider square loss

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

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 $f_t$ 。

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

保留二次项

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

- 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。

### 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。

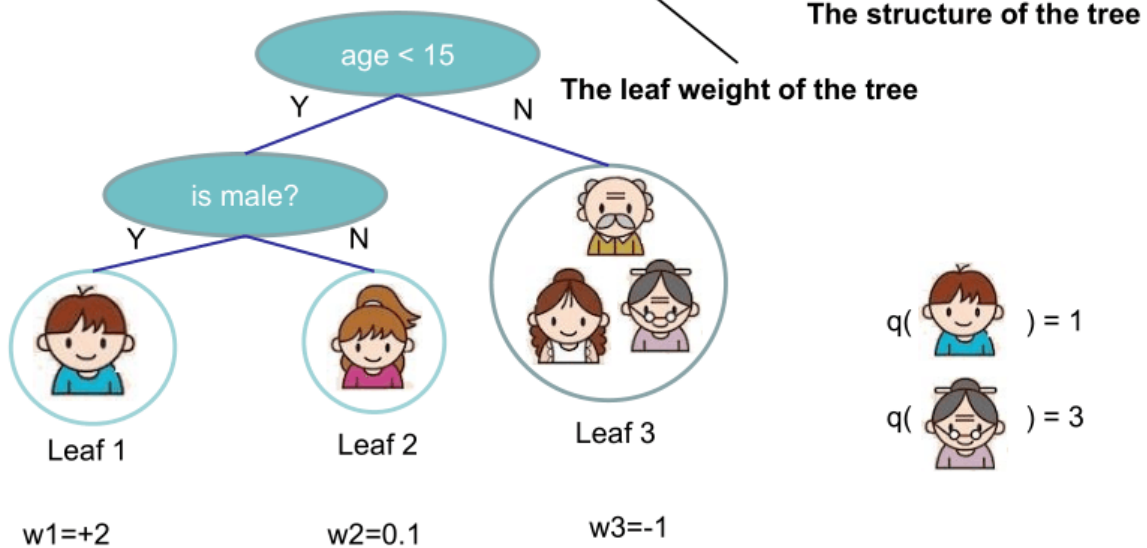
- Why spending so much efforts to derive the objective, why not just grow trees ...

- 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

## 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,  $wq(x)$ 是该节点的打分,即该样本的模型预测值。

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

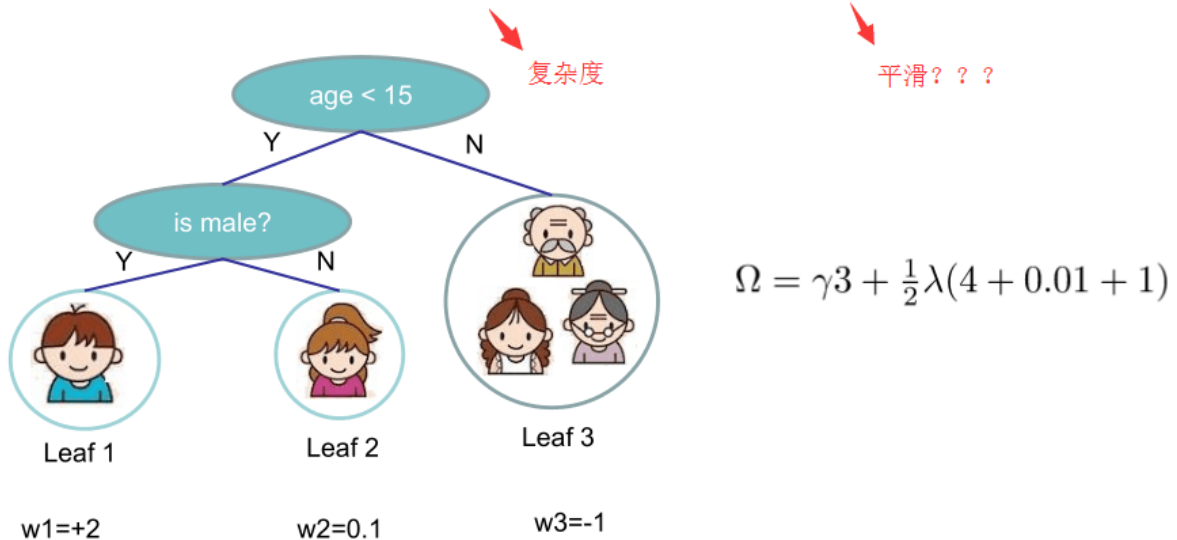


## Define the Complexity of Tree 树的复杂度项

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

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

Number of leaves      L2 norm of leaf scores



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。

## 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。

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  $\Omega$  as follows

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

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

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

and calculate the corresponding optimal objective function value by

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

- Two facts about single variable quadratic function

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

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

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






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

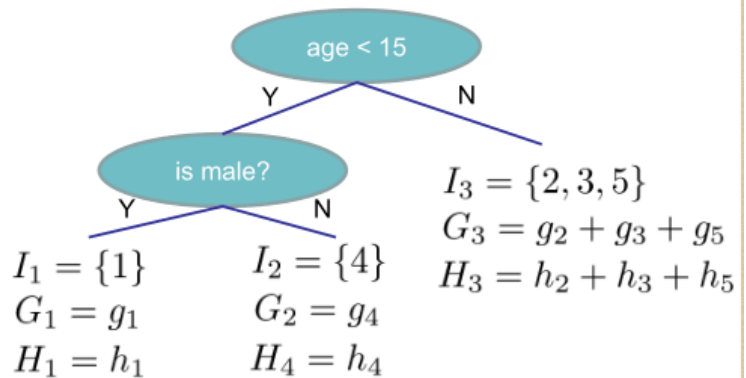
$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad \text{Obj} = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

**This measures how good a tree structure is!** 越大越好，总体损失越小

The Structure Score Calculation :

Instance index      gradient statistics

1		$g_1, h_1$
2		$g_2, h_2$
3		$g_3, h_3$
4		$g_4, h_4$
5		$g_5, h_5$



$$\text{Obj} = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

## 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

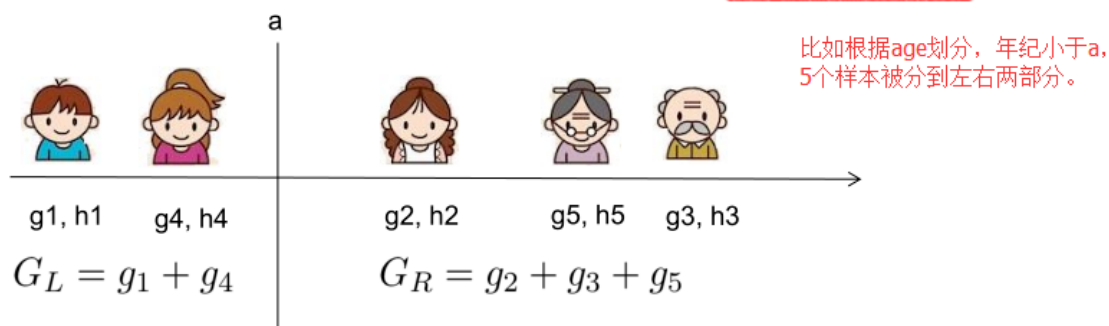
$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

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

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

上图中G都是各自区域内的gi总和，根据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**  $sorted(I, \text{by } \mathbf{x}_{jk})$  **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.

---

## 10 小结

小结一下：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  $f_t(x)$

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T \quad \text{贪心算法寻找切分点，生产新一轮新的树}$$

- 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
  - $\epsilon$  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

## 二、xgboost特点（与gbdt对比）

说明一下：这部分内容参考了知乎上的一个问答—[机器学习算法中GBDT和XGBOOST的区别有哪些？](#)，答主是 [wepon大神](#)，根据他的总结我自己做了一理解和补充。

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）。—可以通过booster [default=gbtrees]设置参数:gbtree: tree-based models/gblinea: 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。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。—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，论文2.3节有介绍，这里答主已概括的非常到位

（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)：**理解的还不够透彻，需要进一步学习**（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，以下引用的还是wepon大神的总结：

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。**一稀疏感知算法，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，会大大减慢划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

**我觉得关于xgboost并行化设计仅仅从论文PPT博客上学习是远远不够的，有时间还要从代码层面去学习分布式xgboost的设计理念。**

## 三、xgboost参数详解

官方参数介绍看这里：[Parameters \(official guide\)](#)

**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

**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。

**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

## 四、实战

官方样例: [XGBoost Python API Reference \(official guide\)](#) [XGBoost Demo Codes \(xgboost GitHub repository\)](#)

**xgboost参数设置代码示例:**

```
# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, random_state=1729)
print(X_train.shape, X_test.shape)

#模型参数设置
xlf = xgb.XGBRegressor(max_depth=10,
                        learning_rate=0.1,
                        n_estimators=10,
                        silent=True,
                        objective='reg:linear',
                        nthread=-1,
```

```
gamma=0,  
min_child_weight=1,  
max_delta_step=0,  
subsample=0.85,  
colsample_bytree=0.7,  
colsample_bylevel=1,  
reg_alpha=0,  
reg_lambda=1,  
scale_pos_weight=1,  
seed=1440,  
missing=None)
```

```
xlf.fit(X_train, y_train, eval_metric='rmse', verbose = True, eval_set = [(X_test,  
y_test)],early_stopping_rounds=100)
```

```
# 计算 auc 分数、预测
```

```
preds = xlf.predict(X_test)
```