# 原理

**XGBoost中文文档：**<https://xgboost.apachecn.org/#/>

**XGBoost完整参数：**<https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst>

**XGBoost原理讲解：**<https://blog.csdn.net/anshuai_aw1/article/details/82970489#_604>

**（推荐，详尽且有实例展示建树过程，包括gi hi计算，增益计算，分裂点选择）**

---> 原理及推导可参考 https://www.cnblogs.com/kuangsyx/p/9043168.html， 描述的还算详细

---> 数学模型: y\_pred\_i = ∑(1,K)f\_k(x\_i), k-->[1,K]

---> 损失函数: loss\_function = (y^i−yi)^2 (可自定义其他loss func， 前提是lossfunc 一阶和二阶可导)

---> 加法训练过程：y\_pred{t} = y\_pred{t-1} + f\_t(x\_i) f\_t(x\_i) 为每次的新增函数

---> 模型正则项： Omiga(f) = γ\*T + 1/2\*λ\*sum(w\_j^2) (j-->[1,T])

---> 优化目标： Obj\_t = loss\_function + Omiga(f)

****1. 简单介绍一下 xgboost(或 xgboost的算法思想)****

基于集成思想，由很多CART回归数集成，通过若干弱分类器的组合成一个强分类器，使模型具有更强的泛化能力。

具体来说就是不断地通过特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数。当训练完k棵树后，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中样本会被到对应的一个叶子节点，每个叶子节点刚好对应一个预测值，将所有这些叶子节点对应的值加起来就是该样本的预测值。

主要是对GBDT进行了一系列的优化，比如损失函数进行了二阶泰勒展开、目标函数加入正则项、支持并行和默认缺失值处理等，在可扩展性和训练速度上有了巨大的提升，但其核心思想没有大的变化。

**--> xgboost建树过程：**

**关键 --- 分裂特征和该特征下最优切分点的选择**

***（每个特征有一个最大Gain对应的特征值，比较所有特征的的最大Gain,选择最大Gain的特征进行分裂，分裂点即为该特征下最大Gain对应的特征值value，feature<value为左子节点，feature>=value为右子节点）***

1. **计算**损失函数在当前模型下每个样本的一阶导gi和二阶导hi（第一棵树的gi和hi需要利用base\_score参数计算，范围 [0,1], 默认0.5，base\_score相当于预测值y\_i\_pred, 残差=y\_i\_pred - y\_i）
2. 分裂特征选择：

遍历特征，以当前特征的每一个属性值作为划分节点，根据gi和hi计算该划分方案下的增益Gain，则每一个特征对应一个最大Gain；则这些最大Gain中的最大值对应的特征就是创建当前树的分裂特征；

1. 分裂特征中的分裂点：

分裂特征的最大Gain对应的特征值feature\_split即为分裂点，小于该值的样本划分到左子节点，其他的划分到右子节点（需要综合考虑**gamma**/min\_split\_loss和**min\_child\_weight**阈值下的预剪枝情况）；

1. 各节点中，重复上述步骤，直到分裂结束，即当前树创建完成；
2. 当前树建成后，更新该棵树对每个样本的预测值y\_i\_pred（相当于第一棵树时初始化的base\_score），重复（1）--（4），完成下一棵树的创建。
3. 以此类推，完成n个Tree的创建，即构造出由n个弱分类器组合成的强分类器。

**NOTES**: 上述步骤中，最耗时的分裂特征选择是可以通过并行实现的，加速效果非常明显。

**--> GBDT:**

它是一种基于boosting增强策略的加法模型，训练的时候采用前向分布算法进行贪婪的学习，每次迭代都学习一棵CART树来拟合之前 t-1 棵树的预测结果与训练样本真实值的残差。（每加入一棵树期望整体表达效果更好）

**--> xgboost与randomForest的区别：**

randomForest的各个决策树是独立的、每个决策树在样本堆里随机选一批样本，随机选一批特征进行独立训练; 而xgboost的决策树依赖于前面决策树的训练和预测结果，即**下一棵决策树输入样本会与前面决策树的训练和预测相关**。

****2. 分裂算法 -- xgboost的节点是如何分裂的？（或xgboost是怎么寻找最佳分裂点？）节点的预测值怎么确定？****

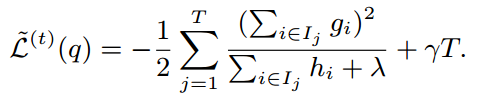
**分裂算法**：基于加权分位数的近似分割算法（贪心算法）

**节点值预测**：二次函数最优化

1）**分裂依据**：

Xgboost的节点分裂是依据样本群基于某个特征值分裂后的损失函数的减小值L\_split​来确定的​。

由xgboost优化推导过程可知 目标函数（带正则的损失函数）如下：



其中，Ij​表示被分到编号为j的个叶子节点的样本。

样本群Ij按某个特征值分类后的损失函数减小量（增益Gain）用*Gain*​表示，Xgboost选择***Gain*=*ObjC*−*ObjL*−*ObjR*** ​这个准则，计算增益*Gain*：



其中，γ可以控制树的复杂度，只有大于*γ*时候才选择分裂。这个其实起到预剪枝的作用。

由上式可知，分裂后的损失函数减小值Gain可简化为如下值：



当前分裂点的选择只考虑能使得**当前**损失函数减少量score最大的点。

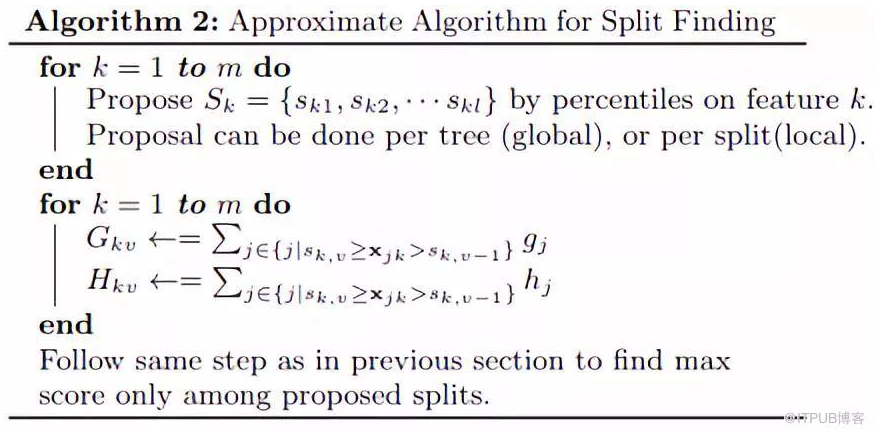
**2）基于加权分位数的近似分割算法（本质：贪心算法）**

该算法首先是根据训练数据中某个特征K的取值的分布情况，通过**加权分位数算法来分桶**选取l个分位数Sk={sk1,sk2,…skl}，将位于相邻分位数点之间的样本分在一个桶中。在遍历特征寻找最优分裂点时就**只搜索这l个分位数点**，并**从这l个分位数点中找到最好的分裂点**作为该特征上最优分裂点的近似。

**（注意：**

**（1）引入的分割不一定会使得情况变好，因为在引入分割的同时也引入新叶子的惩罚项。所以通常需要设定一个阈值，如果引入的分割带来的增益小于一个阀值**γ**的时候，我们可以剪掉这个分割（预剪枝）。**

**（2）在XGBoost的具体实践中，通常会设置树的深度(max\_depth)来控制树的复杂度，避免单个树过于复杂带来的过拟合问题。）**

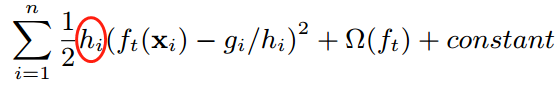


**--->** **特征k的分位数选取：加权分位数算法**

特征k的分位数时并没有按照均匀分位的方式将样本等分到各个区间，而是以**二阶导hi作为权重**进行加权分位数。

**--->** **为什么要采用加权分位数？**

因为我们要均分的是loss，而不是样本的数量，而每个样本对loss的贡献可能是不一样的，按样本均分会导致loss分布不均匀，取到的分位点会有偏差。所以要在每个样本前面加个权重。在xgboost中，样本权重是loss function的二阶偏导 hi（因为目标函数可以简化为带权重的形式，如下，其中权重就是hi）：



**--->** **加权分位数计算过程**

（1）首先将训练集样本根据特征k的值从小到大的顺序进行排列；

（2）选定一个合适的权重累积阈值ϵ，将总的权重分成1/ε等份（等分数即为区间数+2）；

（3）对于i=0,1,2,…,[1/ε]循环执行如下操作：

设定hiv=0，从第一个样本开始遍历，累加各样本权重hiv+=hi,当hiv>ε时，停止累加，并将已经遍历过样本归入第i个区间，得到第i+2个分位数。

对于第（3）步求分位数中，第1个和最后一个分位数是例外，它们默认为特征k的最小值和最大值。

**3）稀疏特征处理 --- 缺失值不敏感**

Xgboost 在处理带缺失值的特征时，先对非缺失的样本进行排序，对该特征缺失的样本先不处理，然后在遍历每个分裂点时，将这些缺失样本分别划入左子树和右子树来计算增益，**哪个增益大就划分到哪边**。所以这个含有缺失值样本的分裂方向是根据训练数据自动学习到的。

如果**训练样本中没有缺失值，而预测过程中出现了缺失值**，那么样本会被默认分到**右子树**。

Refer1: <https://blog.csdn.net/dpengwang/article/details/87910480>

Refer2: <https://blog.csdn.net/w5688414/article/details/78027545>

Refer3: http://blog.itpub.net/31542119/viewspace-2199549/

3. ****XGBoost树的分裂在什么情况下停止？****

主要三种情况会停止建树：

1. **分裂增益（即损失函数减少量）未增加**：当引入的分裂带来的增益小于一个阀值（**gamma** or min\_split\_loss）的时候，可以直接剪掉这个分裂（相当于预剪枝）；
2. **叶子节点数量太少**：样本权重和hi小于设定阈值（超参数：最小叶子节点样本权重和**min\_child\_weight**）时则停止建树；
3. 达到**树的最大深度**时停止建树：超参数**max\_depth**，树过深容易学习局部样本，易过拟合。

4. ****XGBoost为什么不容易过拟合（可以设置哪些参数防止过拟合）？****

通常有两种方式可以控制模型中的过拟合：

第一种：直接控制模型的**复杂度**

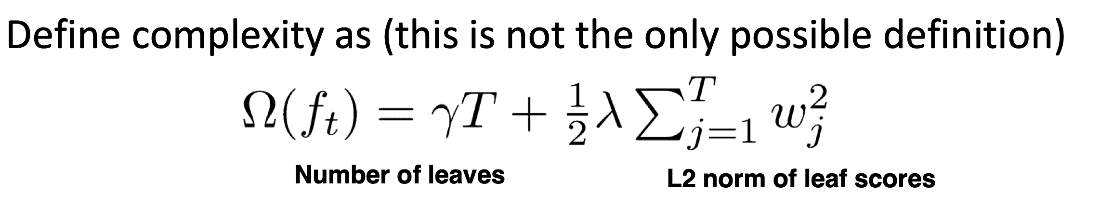
第二种：增加**随机性**，使训练对噪声有更强的鲁棒性

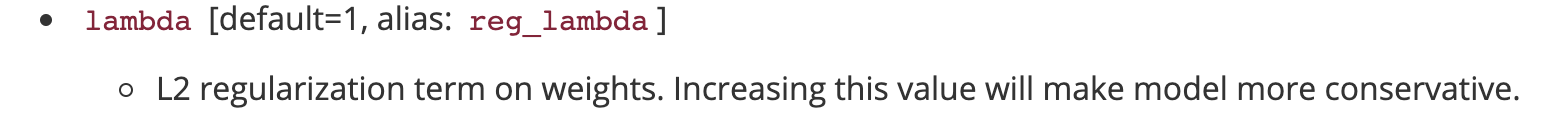
**1）正则项**

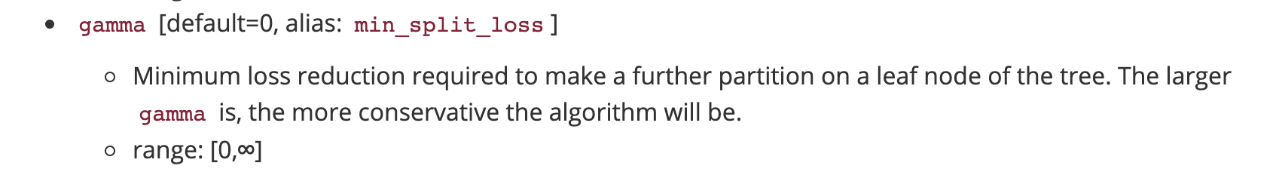
L2正则（参数**λ**）和针对叶子节点数量的惩罚—相当于预剪枝（参数：**γ**）

λ越大也是越希望获得结构简单的树；

γ越大，对较多叶子节点的树的惩罚越大，表示越希望获得结构简单的树

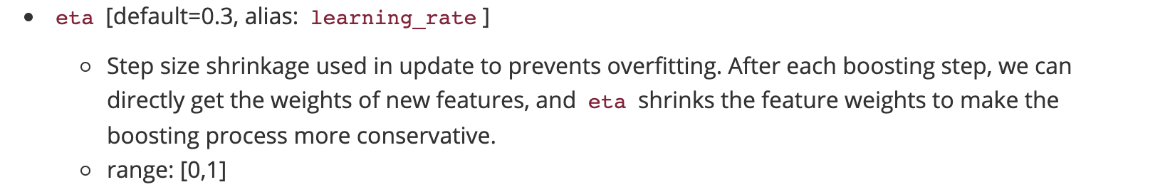






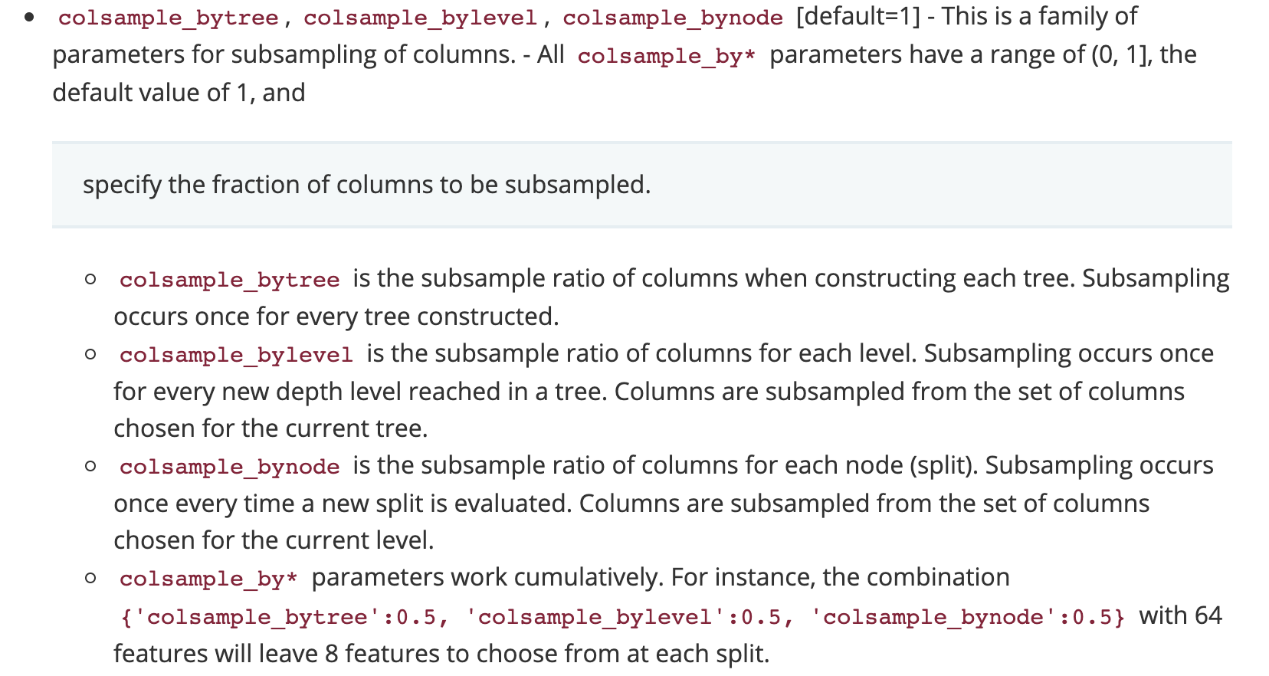
**2）Shrinkage 权重缩减**

相当于学习速率（xgboost中的**eta**）。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。



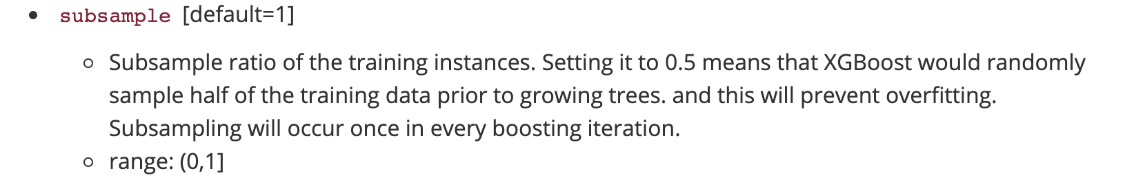
**3）Column Subsampling 列采样**

借用了RandomForest中的列采样思想，只考虑部分特征。不仅支持每颗树（**colsample\_bytree**）/每层（**colsample\_bylevel**）分裂时的列采样，还支持在划分节点（**colsample\_bynode**）时采样。



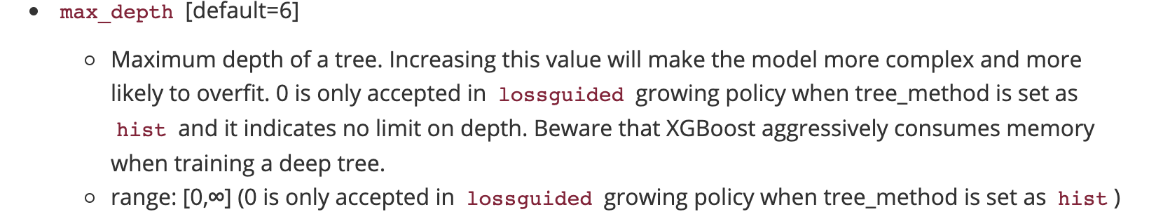
**4）训练数据采样（行采样）**

训练时，对训练数据进行采样（**subsample**），来自随机森林的做法。



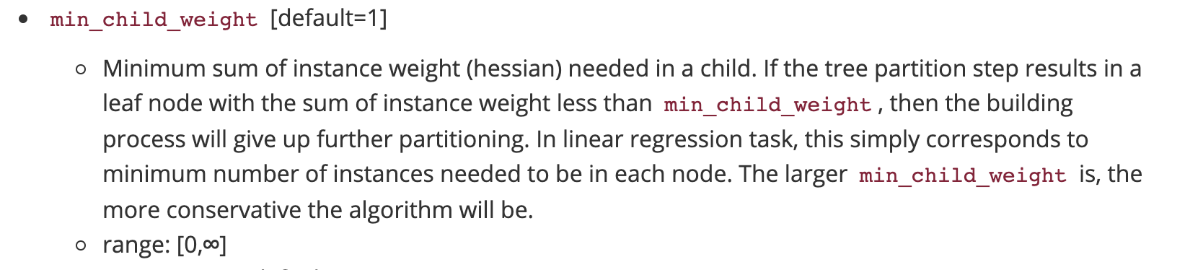
**5）限制树的深度**

树的深度（**max\_depth**），越深越容易过拟合



**6）最小叶子节点样本权重和阈值**

当选择特征X1的最大Gain对应的特征值为分裂点时，如果叶子节点上min(HL​,HR​) 小于**min\_child\_weight**，则放弃该最大Gain的特征值分裂点，考虑次最大Gain的特征值，直至Gain大于最小叶子节点样本权重和(**min\_child\_weight**)时才分裂子树，相当于预剪枝作用。如果没有找到一个满足的，则不进行分裂。



**7）Early-stop（model.****fit中的参数）**

如果在fit方法中设置**early\_stopping\_rounds**，表示从某一轮开始连续**early\_stopping\_rounds**这么多轮迭代中模型的效果没有提升，可以提前终止训练

**---> L1正则化 和 L2正则化**

**L1正则化**：

L1正则化将系数w的l1范数作为惩罚项加到损失函数上，由于正则项非零，这就迫使那些弱的特征所对应的系数变成0。因此L1正则化往往会使学到的模型很稀疏（系数w经常为0），这个特性使得L1正则化成为一种很好的特征选择方法。

L1正则化像非正则化线性模型一样也是不稳定的，如果特征集合中具有相**关联的特征**，当数据发生细微变化时也有**可能导致很大的模型差异**。

**L2正则化**：

L2正则化将系数向量的L2范数添加到了损失函数中。由于L2惩罚项中系数是二次方的，这使得L2和L1有着诸多差异，最明显的一点就是，L2正则化会让系数的取值变得平均。

对于关联特征，这意味着他们能够获得更相近的对应系数。还是以Y=X1+X2为例，假设X1和X2具有很强的关联，如果用L1正则化，不论学到的模型是Y=X1+X2还是Y=2X1，惩罚都是一样的，都是2 alpha。

但是对于L2来说，第一个模型的惩罚项是2 alpha，但第二个模型的是4\*alpha。可以看出，**系数之和为常数时，各系数相等时惩罚是最小的，所以才有了L2会让各个系数趋于相同的特点**。

L2正则化对于特征选择来说一种稳定的模型，L1正则化系数会因为细微的数据变化而波动。所以L2正则化和L1正则化提供的价值是不同的，L2正则化对于特征理解来说更加有用：表示能力强的特征对应的系数是非零。

5. ****XGBoost的并行化主要体现在哪些方面？****

XGBOOST实现的并行化不是在Tree粒度上，这个并行化是**特征粒度上的并行化。即用多线程，尝试把各个特征都作为分裂的特征，找到各个特征的最优分割点，计算根据它们分裂后产生的增益，选择增益最大的那个特征作为分裂的特征。**

**NOTES:**

Boosting算法在Tree粒度上是串行的，都是一次迭代完才能进行下一次迭代（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 构建树的**分裂策略是 level-wise**（另外一个经典的 gbdt 实现 LightGBM 采用的是 leaf-**wise**），在**每一层的分裂过程中是并行执行**的。

****6. XGBoost与GBDT有什么不同和联系？****

**不同：**

(1) **基分类器**：XGBoost的基分类器不仅支持CART决策树，还支持线性分类器，此时XGBoost相当于带L1和L2正则化项的Logistic回归（分类问题）或者线性回归（回归问题）。

(2) **导数信息**：XGBoost对损失函数做了二阶泰勒展开，GBDT只用了一阶导数信息，并且XGBoost还支持自定义损失函数，只要损失函数一阶（**梯度方向**）、二阶可导（**样本权重**）。

(3) **正则项**：XGBoost的目标函数加了正则项， 相当于预剪枝，使得学习出来的模型更加**不容易过拟合**。从Bias-variance tradeoff角度来讲，正则项降低了模型的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

(4) **列抽样**：XGBoost支持列采样，与随机森林类似，用于**防止过拟合，还能降低计算**。

(5) **缺失值处理**：对树中的每个非叶子结点，XGBoost可以自动学习出它的默认分裂方向。如果某个样本该特征值缺失，会将其划入默认分支。

(6) **并行化**：注意不是tree维度的并行，而是特征维度的并行。XGBoost预先将每个特征按特征值排好序，存储为块结构，分裂结点时可以采用多线程并行查找每个特征的最佳分割点，极大提升训练速度。

(7) **可并行的近似直方图算法**（用于**高效地生成候选的分割点**），大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。

**联系：**

**--> XGBoost为什么使用泰勒二阶展开?**

(1)可以更为精准的逼近真实的损失函数;

(2)可扩展性:损失函数支持自定义，只需要新的损失函数二阶可导

**--> Xgboost为什么比GBDT快？**

当数据集大的时候使用近似算法  
Block与并行  
CPU cache 命中优化  
Block压缩、Block拆分等

****7. 损失函数****

L(ϕ)=∑i L(y^i−yi)+ ∑k Ω(fk)

其中：L(y^i−yi): 样本 xi 的训练误差

Ω(fk)表示第k颗树的正则项。

8. xgboost的下一颗树的输入是什么？(或者xgboost拟合的是什么？)

（我的理解是输入y和y\_pred就可以了，可以根据它们算出一阶导数gi和二阶导数hi。Xgboost最重要的是构造损失函数并让他降到最低。）xgboost不刻意拟合任何数值，它在第t步只是寻找一种能使当前损失最小的树。因此它不像adaboost（拟合带权值样本集）和**gbdt（拟合负梯度）**一样以拟合为核心，而是**以使损失函数最低为核心**。它的方法就是通过分裂节点，使得新树的gain大于原来树的gain，从而降低损失函数，而不是数据拟合。“输入”对xgboost来说只是一种形式。

---> BoostingTree / GBDT / XGBoost 拟合描述

(refer: <https://blog.csdn.net/qq_41987033/article/details/81570604>)

**BoostingTree：拟合残差（loss func为mse时）**

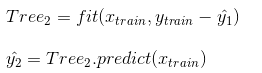
第一棵树的拟合目标是:



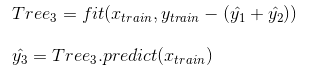
预测输出：



第二棵树的拟合目标是：



第三棵树的拟合目标是：



最终预测值是：



**梯度提升树（GDBT）：拟合loss function的负梯度在当前模型的值**

第一棵树的拟合目标是：



预测输出是：



此时需要计算第一棵树的损失函数，即L_{1}=L(y, \hat{y_{1}})，损失函数是关于\hat{y}_{1}的函 数，计算该损失函数对\hat{y}_{1}的导数的相反数（负梯度）在Tree1上的值。

假设某一损失函数L=(y-\hat{y})^2，某一样本输入x_{i}的标签y_{i}=2，此时样本损 失函数为L_{i}=(2-\hat{y}_{i})^2，其负梯度为-2(\hat{y}_{i}-2)，它就是下一轮样本x_{i}的输入。

综上，第二棵树的拟合目标是：



预测输出是：



第三棵树的拟合目标是：



预测输出是：



最终预测值是：



**Xgboost：拟合不是核心，关键目的在于构造出损失函数，并让它取到最小值**

在XGBOOST中，我们可以选择拟合预测值y\_pred，不过关键点在于构造出损失函数并让它最小化，计算出gi和hi，”输入”只是一种形式。

9. xgboost的优点？

10. 对于不平衡数据要如何处理？

11. Xgboost如何用于特征选择？

https://blog.csdn.net/anshuai\_aw1/article/details/82970489#\_604

12. 决策树、RF、GBDT、xgboost如何处理缺失值？是否缺失值敏感？

<https://blog.csdn.net/qq_19446965/article/details/81637199>

https://www.zhihu.com/question/58230411

13. 看下Xgboost的工程优化 （待验证）

Refer: <https://blog.csdn.net/anshuai_aw1/article/details/85093106>

这部分因为没有实战经验，都是论文、博客解读来的，所以也不十分确定，供参考。

* Column Block for Parallel Learning

总的来说：按列切开，升序存放；

方便并行，同时解决一次性样本读入炸内存的情况

由于将数据按列存储，可以同时访问所有列，那么可以对所有属性同时执行split finding算法，从而并行化split finding（切分点寻找）-特征间并行

可以用多个block(Multiple blocks)分别存储不同的样本集，多个block可以并行计算-特征内并行

• Blocks for Out-of-core Computation

数据大时分成多个block存在磁盘上，在计算过程中，用另外的线程读取数据，但是由于磁盘IO速度太慢，通常更不上计算的速度，

将block按列压缩，对于行索引，只保存第一个索引值，然后只保存该数据与第一个索引值之差(offset)，一共用16个bits来保存 offset，因此，一个block一般有2\*\*16个样本。

14. bagging和boosting以及对应的常用模型

<https://www.cnblogs.com/aixiao07/p/11375168.html>

15. GBDT

<https://blog.csdn.net/anshuai_aw1/article/details/82888222>

16. LightGBM

https://blog.csdn.net/anshuai\_aw1/article/details/83659932