关于参数使用的一个参考

下方是未整理过的版本，emmm，比讲的要混乱一些

**Advantages:**

**1.Regularization（正则化）:**

Xgboost就是一个以”正则化提升“技术闻名的工具，很明显，这可以减少过拟合。

**2.Parallel Processing（并行处理）:**

如果大家看过我前面分享的一篇集成学习的文章： 集成学习之bagging、boosting及AdaBoost的实现不免心生疑问，那篇文章中明确指出，boosting算法是串行算法，每个学习器的生成都是依赖于前面一个学习器的生成的，那么Xgboost又是如何实现并行的呢，详情请戳：Parallel Gradient Boosting Decision Trees。

**3.High Flexibility（高度灵活）：**

Xgboost可以让使用者自定义优化目标与评估标准。

**4.Handling Missing Values（处理缺失值）：**

Xgboost通过一个内置的程序来处理缺失值，但是需要用户提供一个与其他观察值不同的缺失值，并作为参数传递。

**5.Tree Pruning（树剪枝）:**

Xgboost中有个参数max\_depth，因此Xgboost会持续分裂直到达到max\_depth，然后回溯剪枝

**6.Built-in Cross-Validation（内置的交叉验证）：**

Xgboost允许用户在每次boosting迭代的过程中应用交叉验证

**8.Continue on Existing Model（继续现有模型）：**

用户可以从上一次运行的最后一次迭代中开始训练XGBoost模型。这在某些特定的应用程序中具有很大的优势。

**Parameters:**

1. **Classification**

General Parameters（通用参数）：设置整体功能

Booster Parameters（提升参数）：选择你每一步的booster(树or回归）

Learning Task Parameters（学习任务参数）：指导优化任务的执行

1. **General Parameters**

下面这些参数定义了Xgboost的总体功能：

1.booster [default=gbtree]

选择每次迭代过程中需要运行的模型，一共有两种选择：

gbtree：基于树的模型

gbliner：线性模型

gbtree: tree-based models

gb\_trees(General Balanced Trees)

DESCRIPTION:

An efficient implementation of Prof. Arne Andersson's General Balanced Trees. These have no storage overhead compared to unbalanced binary trees, and their performance is in general better than AVL trees.

This module considers two keys as different if and only if they do not compare equal (==).

通用二叉树,通常被用作有序字典.与普通未平衡二叉树相比没有额外的储存开销,这里所说的额外的存储开销是指是否使用额外的metadata记录节点相关的信息,dict和array的实现就使用了这样的描述信息,换句话说gb\_trees是自描述的.性能优于AVL trees.和proplists,orddict相比它能够支持更大的数据量.

平衡二叉树(AVL树) ,左右子树的深度(一棵树中最大的结点度数)只差绝对值不超过1.这个插值称为平衡因子(balance factor),查找,插入和删除在平均和最坏情况下都是O(logn).节点的插入和删除平衡树失衡会触发重新平衡.重新平衡通过四种旋转实现:LL LR RR RL.gb\_trees由于节点删除操作并不会增加树的高度,所以节点删除之后并没有进行再平衡.

gb\_treess数据项比较使用的是相等==操作符.

2.silent [default=0]

设置模型是否有logo打印：

0：有打印

1：无打印

3.nthread [default to maximum number of threads available if not set]

这个主要用于并行处理的，如果不指定值，工具会自动检测

剩余两个参数是Xgboost自动指定的，无需设置

2、silent[默认0]

当这个参数值为1时，静默模式开启，不会输出任何信息。 一般这个参数就保持默认的0，因为这样能帮我们更好地理解模型。

3、nthread[默认值为最大可能的线程数]

这个参数用来进行多线程控制，应当输入系统的核数。 如果你希望使用CPU全部的核，那就不要输入这个参数，算法会自动检测它。

还有两个参数，XGBoost会自动设置，目前你不用管它。接下来咱们一起看booster参数。

1. Booster Parameters

提升参数

虽然有两种类型的booster，但是我们这里只介绍tree。因为tree的性能比线性回归好得多，因此我们很少用线性回归。

**1.eta [default=0.3, alias: learning\_rate]**

学习率，可以缩减每一步的权重值，使得模型更加健壮：

典型值一般设置为：0.01-0.2

为了防止过拟合，更新过程中用到的收缩步长。在每次提升计算之后，算法会直接获得新特征的权重。 eta通过缩减特征的权重使提升计算过程更加保守。缺省值为0.3

取值范围为：[0,1]

通常最后设置eta为0.01~0.2

**2.min\_child\_weight [default=1]**

定义了一个子集的所有观察值的最小权重和。

这个可以用来减少过拟合，但是过高的值也会导致欠拟合，因此可以通过CV来调整min\_child\_weight。

孩子节点中最小的样本权重和。**如果一个叶子节点的样本权重和小于min\_child\_weight则拆分过程结束。**在现行回归模型中，这个参数是指建立每个模型所需要的最小样本数。该成熟越大算法越conservative。即调大这个参数能够控制过拟合。

**3.max\_depth [default=6]**

树的最大深度，值越大，树越复杂。

这个可以用来控制过拟合，典型值是3-10。

树的深度越大，则对数据的拟合程度越高（过拟合程度也越高）。即该参数也是控制过拟合

建议通过交叉验证（xgb.cv ) 进行调参

通常取值：3-10

**4.gamma [default=0, alias: min\_split\_loss]**

这个指定了一个结点被分割时，所需要的最小损失函数减小的大小。

这个值一般来说需要根据损失函数来调整。

minimum loss reduction required to make a further partition on a leaf node of the tree. the larger, the more conservative the algorithm will be.

range: [0,∞]

在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。Gamma指定了节点分裂所需的最小损失函数下降值。

模型在默认情况下，对于一个节点的划分只有在其loss function 得到结果大于0的情况下才进行，而gamma 给定了所需的最低loss function的值

gamma值使得算法更conservation，且其值依赖于loss function ，在模型中应该进行调参。

**5.max\_delta\_step(默认= 0)**

这个参数通常并不需要。

这参数限制每棵树权重改变的最大步长。如果这个参数的值为0，那就意味着没有约束。如果它被赋予了某个正值，那么它会让这个算法更加保守。 通常，这个参数不需要设置。但是当各类别的样本十分不平衡时，它对逻辑回归是很有帮助的。 这个参数一般用不到，但是你可以挖掘出来它更多的用处。

Maximum delta step we allow each tree’s weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to value of 1-10 might help control the update

取值范围为：[0,∞]

如果取值为0，那么意味着无限制。如果取为正数，则其使得xgboost更新过程更加保守。

通常不需要设置这个值，但在使用logistics 回归时，若类别极度

**6.subsample [default=1]**

样本的采样率，如果设置成0.5，那么Xgboost会随机选择一般的样本作为训练集。

用于训练模型的子样本占整个样本集合的比例。如果设置为0.5则意味着XGBoost将随机的从整个样本集合中抽取出50%的子样本建立树模型，这能够防止过拟合。

取值范围为：(0,1]

这个参数控制对于每棵树，随机采样的比例。 减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合。 典型值：0.5-1

**7.colsample\_bytree [default=1]**

构造每棵树时，列采样率（一般是特征采样率）。

在建立树时对特征随机采样的比例。缺省值为1

用来控制每棵随机采样的列数的占比(每一列是一个特征)。 典型值：0.5-1

取值范围：(0,1]

**8.colsample\_bylevel [default=1]**

每执行一次分裂，列采样率。这个一般很少用，6和7参数调节就足够了。

决定每次节点划分时子样例的比例

通常不使用，因为subsample和colsample\_bytree已经可以起到相同的作用了

**9.lambda [default=1, alias: reg\_lambda]**

L2正则化（与岭回归中的正则化类似：传送门）这个其实用的很少。

L2 正则的惩罚系数

用于处理XGBoost的正则化部分。通常不使用，但可以用来降低过拟合

alpha [default=0]

**10.alpha [default=0, alias: reg\_alpha]**

L1正则化（与lasso回归中的正则化类似：传送门）这个主要是用在数据维度很高的情况下，可以提高运行速度。

**11.scale\_pos\_weight, [default=1]**

在类别高度不平衡的情况下，将参数设置大于0，可以加快收敛。

**12.alpha [default=0]**

L1 正则的惩罚系数

当数据维度极高时可以使用，使得算法运行更快。

**13.lambda\_bias**

在偏置上的L2正则。缺省值为0（在L1上没有偏置项）

学习任务参数

这类参数主要用来明确学习任务和相应的学习目标的

**1.objective [default=reg:linear]**

这个主要是指定学习目标的：而分类，还是多分类or回归

“reg:linear” –linear regression：回归

“binary:logistic”：二分类

“multi:softmax” ：多分类，这个需要指定类别个数

这个参数定义需要被最小化的损失函数。最常用的值有：

binary:logistic 二分类的逻辑回归，返回预测的概率(不是类别)。 multi:softmax 使用softmax的多分类器，返回预测的类别(不是概率)。

在这种情况下，你还需要多设一个参数：num\_class(类别数目)。 multi:softprob 和multi:sof

**2.eval\_metric [default according to objective]**

\*评估方法，主要用来验证数据，根据一个学习目标会默认分配一个评估指标

“rmse”:均方根误差（回归任务）

“error”:分类任务

“map”:Mean average precision（平均准确率，排名任务）

等等

对于有效数据的度量方法。 对于回归问题，默认值是rmse，对于分类问题，默认值是error。 典型值有：

rmse 均方根误差(∑Ni=1?2N??????√) mae 平均绝对误差(∑Ni=1|?|N) logloss 负对数似然函数值 error 二分类错误率(阈值为0.5) merror 多分类错误率 mloglo

**3.seed [default=0]**

随机数种子，可以用来生成可复制性的结果，也可用来调参

'learning\_rate': 0.01,

'boosting\_type': 'gbdt', # 训练方式

'objective': 'regression', #训练目标 回归

'metric': 'mse', # 损失函数 均方误差

'sub\_feature': 0.7, # feature sub-sample, will random select 80% feature to train on each iteration

# alias: sub\_feature

'num\_leaves': 60, #叶的数量

'colsample\_bytree': 0.7, #colsample\_bytree 每棵随机采样的列数的占比

'feature\_fraction': 0.7,

'min\_data': 100, # minimal number data for one leaf, use this to deal with over-fit

# alias : min\_data\_per\_leaf, min\_data

'min\_hessian': 1, # minial sum hessians for one leaf, use this to deal with over-fit

'verbose': -1, #运行时不显示详细信息

整max\_depth 和min\_child\_weight

我们调整这两个参数是因为这两个参数对输出结果的影响很大。我们首先将这两个参数设置为较大的数，然后通过迭代的方式不断修正，缩小范围。

（接下来的网格搜索，会消耗很多时间）

Xgboots LightGbm

booster(default=gbtree) boosting(default=gbdt)

eta(default=0.3) learning\_rate(default=0.1)

max\_depth(default=6) num\_leaves(default=31)

min\_child\_weight(default=1) min\_sum\_hessian\_in\_leaf(1e-3)

gamma(default=0) min\_gain\_to\_split(default=20)

subsample(default=1) bagging\_fraction(default=1.0)

colsample\_bytree(default=1) feature\_fraction( default=1.0)

alpha(default=0) lambda\_l1(default=0)

lambda(default=1) lambda\_l2(default=0)

objective( default=reg:linear) application(default=regression)

eval\_metric(default according to objective) metric

nthread num\_threads

1. 使用num\_leaves

因为LightGBM使用的是leaf-wise的算法，因此在调节树的复杂程度时，使用的是num\_leaves而不是max\_depth。

大致换算关系：num\_leaves = 2^(max\_depth)。它的值的设置应该小于2^(max\_depth)，否则可能会导致过拟合。

这是控制树模型复杂性的重要参数。理论上，我们可以通过设定num\_leaves = 2^(max\_depth) 去转变成为depth-wise tree。但这样容易过拟合，因为当这两个参数相等时, leaf-wise tree的深度要远超depth-wise tree。因此在调参时，往往会把 num\_leaves的值设置得小于2^(max\_depth)。

2.对于非平衡数据集：可以param['is\_unbalance']='true’

3. Bagging参数：bagging\_fraction+bagging\_freq（必须同时设置）、feature\_fraction。bagging\_fraction可以使bagging的更快的运行出结果，feature\_fraction设置在每次迭代中使用特征的比例。

4. min\_data\_in\_leaf：这也是一个比较重要的参数，调大它的值可以防止过拟合，它的值通常设置的比较大。

5.max\_bin:调小max\_bin的值可以提高模型训练速度，调大它的值和调大num\_leaves起到的效果类似。

2. min\_data\_in\_leaf. 这是另一个避免leaf-wise tree算法过拟合的重要参数。该值受到训练集数量和num\_leaves这两个值的影响。把该参数设的更大能够避免生长出过深的树，但也要避免欠拟合。在分析大型数据集时，该值区间在数百到数千之间较为合适。

3. max\_depth. 也可以通过设定 max\_depth 的值来限制树算法生长过深。

**提高速度的参数**

· 通过设定bagging\_fraction和bagging\_freq来使用 bagging算法

· 通过设定 feature\_fraction来对特征采样

· 设定更小的max\_bin值

· 使用save\_binary 以方便往后加载数据的速度

· 设定平行计算参数

**提高精度的参数**

· 设定更大的max\_bin值(但会拖慢速度)

· 设定较小的learning\_rate值，较大的num\_iterations值

· 设定更大的num\_leaves值(但容易导致过拟合)

· 加大训练集数量（更多样本，更多特征）

· 试试boosting= dart

**避免过拟合的参数**

· 设定较小的max\_bin

· 设定更小的num\_leaves

· 设定min\_data\_in\_leaf和min\_sum\_hessian\_in\_leaf

· 通过设定bagging\_fraction和bagging\_freq来使用 bagging算法

· 通过设定feature\_fraction来对特征采样。

· 加大训练集数量（更多样本，更多特征）

· 通过设定lambda\_l1, lambda\_l2以及min\_gain\_to\_split来采取正则化措施

· 通过设定max\_depth以避免过拟合

(level-wise) 的决策树生长策略，而使用了带有深度限制的按叶子生长 (leaf-wise) 算法。 level-wise 过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，不容易过拟合。但实际上level-wise是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销。因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

leaf-wise则是一种更为高效的策略，每次从当前所有叶子中，找到分裂增益最大(一般也是数据量最大)的一个叶子，然后分裂，如此循环。因此同 level-wise 相比，在分裂次数相同的情况下，leaf-wise 可以降低更多的误差，得到更好的精度。leaf-wise 的缺点是可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在leaf-wise 之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。

我们会使用和GBM中相似的方法。需要进行如下步骤：

1. 选择较高的学习速率(learning rate)。一般情况下，学习速率的值为0.1。但是，对于不同的问题，理想的学习速率有时候会在0.05到0.3之间波动。选择对应于此学习速率的理想决策树数量。XGBoost有一个很有用的函数“cv”，这个函数可以在每一次迭代中使用交叉验证，并返回理想的决策树数量。

2. 对于给定的学习速率和决策树数量，进行决策树特定参数调优(max\_depth, min\_child\_weight, gamma, subsample, colsample\_bytree)。在确定一棵树的过程中，我们可以选择不同的参数，待会儿我会举例说明。

3. xgboost的正则化参数的调优。(lambda, alpha)。这些参数可以降低模型的复杂度，从而提高模型的表现。

4. 降低学习速率，确定理想参数。