**LightGBM算法梳理**

1. LightGBM
2. LightGBM的起源
3. Histogram VS pre-sorted
4. leaf-wise VS level-wise
5. 特征并行和数据并行
6. 顺序访问梯度
7. 支持类别特征
8. 应用场景
9. sklearn参数
10. CatBoost(了解)

LightGBM是一个梯度Boosting框架，使用基于决策树的学习算法。它可以说是分布式的，高效的，有以下优势：

1）更快的训练效率  2）低内存使用  3）更高的准确率 4）支持并行化学习 5）可以处理大规模数据

mini-batch:

在梯度下降中需要对所有数据计算一遍再决定下一步，当数据集样本规模特别大时，这样的一次迭代会非常耗时，这时的梯度下降叫full-batch。为提高效率，我们将样本总数为NN NN的数据集分割为TT TT个含mm mm个样本的子数据集，循环遍历这TT TT个子集，对每个子集做一次梯度下降，到下个子集时继续进行梯度下降，这就是mini-batch。

而GBDT在每一次迭代的时候，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。尤其面对工业级海量的数据，普通的GBDT算法是不能满足其需求的。

LightGBM提出的主要原因就是为了解决GBDT在海量数据遇到的问题，让GBDT可以更好更快地用于工业实践。

Histogram VS pre-sorted

xgboost是对特征pre-sorted的方法:在训练之前，预先对数据进行排序，然后保存block结构，后面的迭代中重复的使用这个结构。计算过程中是按照value的排序，逐个数据样本来计算划分收益，这样的算法能够精确的找到最佳划分值，但是代价比较大同时也没有较好的推广性。

LightGBM是histogram算法：将这些精确的连续的特征值分成许多小筒，以每一个桶为bin做直方图，进而在筒上搜索分裂点，减少了计算代价和存储代。而且直方图带来了一定的正则化的效果，能够使我们做出来的模型避免过拟合且具有更好的推广性。

leaf-wise VS level-wise

xgboost用的是level-wise, LightGBM 用得是leaf-wise。leaf-wise是每次从当前所有叶子中，找到分裂增益最大的一个叶子，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。但是它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限制，在保证高效率的同时防止过拟合。

特征并行和数据并行

LightGBM支持并行学习，目前支持特征并行(Featrue Parallelization)和数据并行(Data Parallelization)两种，还有一种是基于投票的数据并行(Voting Parallelization)

特征并行的主要思想是在不同机器、在不同的特征集合上分别寻找最优的分割点，然后在机器间同步最优的分割点。通过在本地保存全部数据避免对数据切分结果的通信。

数据并行则是让不同的机器先在本地构造直方图，然后进行全局的合并，最后在合并的直方图上面寻找最优分割点。使用分散规约 (Reduce scatter) 把直方图合并的任务分摊到不同的机器，降低通信和计算，并利用直方图做差，进一步减少了一半的通信量。

基于投票的数据并行(Voting Parallelization)则进一步优化数据并行中的通信代价，使通信代价变成常数级别。在数据量很大的时候，使用投票并行可以得到非常好的加速效果。

顺序访问梯度

pre-sort算法中有两个频繁的操作会带来系统性能下降，导致cache-miss，也就是缓存消失，即梯度随机访问和索引表随机访问。

对梯度的访问：在计算增益的时候需要利用梯度，对于不同的特征，访问梯度的顺序是不一样的，并且是随机的

对于索引表的访问：预排序算法使用了行号和叶子节点号的索引表，防止数据切分的时候对所有的特征进行切分。同访问梯度一样，所有的特征都要通过访问这个索引表来索引。

LightGBM使用的直方图算法能很好的解决这类问题。首先。对梯度的访问，因为不用对特征进行排序，同时，所有的特征都用同样的方式来访问，所以只需要对梯度访问的顺序进行重新排序，所有的特征都能连续的访问梯度。并且直方图算法不需要把数据id到叶子节点号上，不需要访问索引表。

支持类别特征

传统的机器学习一般不能支持直接输入类别特征，需要先转化成多维的0-1特征，这样无论在空间上还是时间上效率都不高。LightGBM通过更改决策树算法的决策规则，直接原生支持类别特征，不需要转化，提高了速度。

参数

针对 leaf-wise 树的参数优化：

num\_leaves：控制了叶节点的数目。它是控制树模型复杂度的主要参数。

如果是level-wise，则该参数为2^{depth}，其中depth为树的深度。但是当叶子数量相同时，leaf-wise的树要远远深过level-wise树，非常容易导致过拟合。因此应该让num\_leaves小于2^{depth}。在leaf-wise树中，并不存在depth的概念。因为不存在一个从leaves到depth的合理映射。

min\_data\_in\_leaf：每个叶节点的最少样本数量。它是处理leaf-wise树的过拟合的重要参数。将它设为较大的值，可以避免生成一个过深的树。但是也可能导致欠拟合。

max\_depth： 控制了树的最大深度。该参数可以显式的限制树的深度。

针对更快的训练速度：

通过设置 bagging\_fraction 和 bagging\_freq 参数来使用 bagging 方法

通过设置 feature\_fraction 参数来使用特征的子抽样

使用较小的 max\_bin

使用 save\_binary 在未来的学习过程对数据加载进行加速

获取更好的准确率：

使用较大的 max\_bin （学习速度可能变慢）

使用较小的 learning\_rate 和较大的 num\_iterations

使用较大的 num\_leaves （可能导致过拟合）

使用更大的训练数据

尝试 dart

缓解过拟合：

使用较小的 max\_bin

使用较小的 num\_leaves

使用 min\_data\_in\_leaf 和 min\_sum\_hessian\_in\_leaf

通过设置 bagging\_fraction 和 bagging\_freq 来使用 bagging

通过设置 feature\_fraction 来使用特征子抽样

使用更大的训练数据

使用 lambda\_l1, lambda\_l2 和 min\_gain\_to\_split 来使用正则

尝试 max\_depth 来避免生成过深的树

核心参数：

boosting 或者’boost’ 或者 ‘boosting\_type’： 一个字符串，给出了基学习器模型算法。可以为：

‘gbdt’： 表示传统的梯度提升决策树。默认值为’gbdt’

‘rf’： 表示随机森林。

‘dart’： 表示带dropout 的gbdt

goss：表示Gradient-based One-Side Sampling 的gbdt

data或者train或者train\_data：一个字符串，给出了训练数据所在的文件的文件名。默认为空字符串。lightgbm将使用它来训练模型。

valid或者test或者valid\_data或者test\_data：一个字符串，表示验证集所在的文件的文件名。默认为空字符串。lightgbm将输出该数据集的度量。如果有多个验证集，则用逗号分隔。

num\_iterations或者num\_iteration或者num\_tree或者num\_trees或者num\_round或者num\_rounds或者num\_boost\_round 一个整数，给出了boosting的迭代次数。默认为 100。

learning\_rate或者shrinkage\_rate： 个浮点数，给出了学习率。默认为1。在dart 中，它还会影响dropped trees 的归一化权重。

num\_leaves或者num\_leaf：一个整数，给出了一棵树上的叶子数。默认为 31

tree\_learner或者tree：一个字符串，给出了tree learner，主要用于并行学习。 默认为’serial’。

num\_threads 或者num\_thread 或者nthread：一个整数

学习控制参数：

max\_depth： 一个整数，限制了树模型的最大深度，默认值为-1。如果小于0，则表示没有限制。

min\_data\_in\_leaf 或者 min\_data\_per\_leaf 或者 min\_data或者min\_child\_samples： 一个整数，表示一个叶子节点上包含的最少样本数量。默认值为 20

min\_sum\_hessian\_in\_leaf 或者 min\_sum\_hessian\_per\_leaf或者 min\_sum\_hessian 或者 min\_hessian或者min\_child\_weight： 一个浮点数，表示一个叶子节点上的最小hessian 之和。（也就是叶节点样本权重之和的最小值） 默认为1e-3 。

feature\_fraction或者sub\_feature或者colsample\_bytree：一个浮点数，取值范围为[0.0,1.0]， 默认值为0。如果小于1.0，则lightgbm 会在每次迭代中随机选择部分特征。如0.8 表示：在每棵树训练之前选择80% 的特征来训练。

feature\_fraction\_seed： 一个整数，表示feature\_fraction 的随机数种子，默认为2。

bagging\_fraction 或者sub\_row 或者 subsample：一个浮点数，取值范围为[0.0,1.0]， 默认值为0。如果小于1.0，则lightgbm 会在每次迭代中随机选择部分样本来训练（非重复采样）。如0.8 表示：在每棵树训练之前选择80% 的样本（非重复采样）来训练。

bagging\_freq 或者subsample\_freq：一个整数，表示每bagging\_freq 次执行bagging。如果该参数为0，表示禁用bagging。

bagging\_seed 或者 bagging\_fraction\_seed：一个整数，表示bagging 的随机数种子，默认为 3 。

early\_stopping\_round 或者 early\_stopping\_rounds或者early\_stopping：一个整数，默认为0。如果一个验证集的度量在early\_stopping\_round 循环中没有提升，则停止训练。如果为0则表示不开启早停。

lambda\_l1 或者reg\_alpha： 一个浮点数，表示L1正则化系数。默认为0

lambda\_l2 或者reg\_lambda： 一个浮点数，表示L2正则化系数。默认为0

min\_split\_gain 或者min\_gain\_to\_split： 一个浮点数，表示执行切分的最小增益，默认为0

drop\_rate： 一个浮点数，取值范围为[0.0,1.0]，表示dropout 的比例，默认为1。 该参数仅在dart 中使用