**LightGBM**

1. **LightGBM简介**

GBDT (Gradient Boosting Decision Tree)主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。

GBDT应用广泛，通常被用于多分类、点击率预测、搜索排序等任务，也常在各种数据挖掘竞赛中被使用。

LightGBM（Light Gradient Boosting Machine）是一个实现GBDT算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点。

## 1.1 LightGBM提出的动机

常用的机器学习算法，例如神经网络等算法，都可以以mini-batch的方式训练，训练数据的大小不会受到内存限制。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。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。

## 1.2 XGBoost的缺点

XGBoost是基于预排序方法的决策树算法。基本思想是：

1、对所有特征都按照特征的数值进行预排序。

2、在遍历分割点的时候用的增益重要性或分裂重要性等找到一个特征上的最好分割点。

3、在找到一个特征的最好分割点后，将数据分裂成左右子节点。

这样的预排序算法的优点是能精确地找到分割点。但是缺点也很明显：

1、空间消耗大。这样的算法需要保存数据的特征值，还保存了特征排序的结果（例如，为了后续快速的计算分割点，保存了排序后的索引），这就需要消耗训练数据两倍的内存。

2、时间上有较大的开销，在遍历每一个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。

3、对cache优化不友好。在预排序后，特征对梯度的访问是一种随机访问，并且不同的特征访问的顺序不一样，无法对cache进行优化。

## 1.3 LightGBM的优化

LightGBM相比于XGBoost上主要有4方面的优化。

1、Histogram算法：直方图加速算法。

2、GOSS算法：基于梯度的单边采样算法。

3、EFB算法：互斥特征捆绑算法。

4、带深度限制的Leaf-wise的叶子生长策略

LightGBM = XGBoost + Histogram + GOSS + EFB。

**1.3.1 Histogram直方图算法**

直方图算法是替代XGBoost的预排序(pre-sorted)算法的。

预排序算法首先将样本按照特征取值排序，然后从全部特征取值中找到最优的分裂点位，该算法的候选分裂点数量与样本数量成正比。

而直方图算法通过将连续特征值离散化到固定数量(如255个)的bins上，使得候选分为点位为常数个(num\_bins-1)。

此外，直方图算法还能够作直方图差加速。当节点分裂成两个时，右边叶子节点的直方图等于其父节点的直方图减去左边叶子节点的直方图。

从而大大减少构建直方图的计算量。

**1.3.2 GOSS算法**

GOSS算法全称为Gradient-based One-Side Sampling，即基于梯度的单边采样算法。

其主要思想是通过对样本采样的方法来减少计算目标函数增益时候的复杂度。

但如果对全部样本进行随机采样，势必会对目标函数增益的计算精度造成较大的影响。

GOSS算法的创新之处在于它只对梯度绝对值较小的样本按照一定比例进行采样，而保留了梯度绝对值较大的样本。

这就是所谓的单边采样。由于目标函数增益主要来自于梯度绝对值较大的样本，因此这种方法在计算性能和计算精度之间取得了很好的平衡。

**1.3.3 EFB算法**

EFB算法全称是Exclusive Feature Bundling，即互斥特征绑定算法。

EFB算法可以有效减少用于构建直方图的特征数量，从而降低计算复杂度，尤其是特征中包含大量稀疏特征的时候。

在许多应用场景下，数据集中会有大量的稀疏特征，这些稀疏特征大部分样本都取值为0，只有少数样本取值非0。

通常可以认为这些稀疏特征是互斥的，即它们几乎不会同时取非零值。

利用这种特性，可以通过对某些特征的取值重新编码，将多个这样互斥的特征捆绑成为一个新的特征。

有趣的是，对于类别特征，如果转换成onehot编码，则这些onehot编码后的多个特征相互之间是互斥的，

从而可以被捆绑成为一个特征。因此，对于指定为类别特征的特征，LightGBM可以直接将每个类别取值和一个bin关联，从而自动地处理它们。

而无需预处理成onehot编码多此一举。

**1.3.4 Leaf-wise**

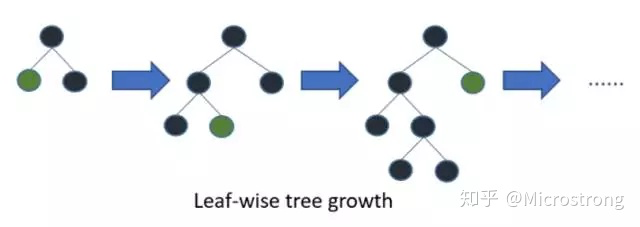
大多数GBDT工具使用低效的按层生长 (level-wise) 的决策树生长策略，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销。实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。LightGBM使用了带有深度限制的按叶子生长 (leaf-wise) 算法。

**leaf-wise：**

每次选择分裂后收益最大的叶节点。

优点是拟合更为精确；

缺点是可能生成较深的树造成过拟合，要加上深度的限制；分裂过程是顺序的。

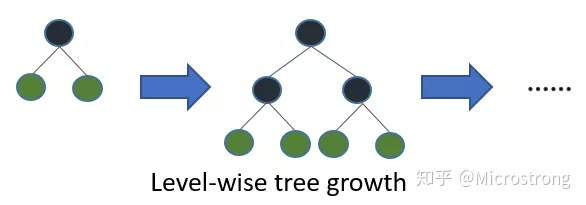


**level-wise：**

不需要挑选结点，每次分裂同一层所有结点

优点是同一层结点可以并行计算

缺点是有的结点可能本身不需要分裂，增加了内存消耗



1. **LightGBM优缺点**

## 2.1 LightGBM的主要优点

* 简单易用。提供了主流的Python\C++\R语言接口，用户可以轻松使用LightGBM建模并获得相当不错的效果。
* 高效可扩展。在处理大规模数据集时高效迅速、高准确度，对内存等硬件资源要求不高。
* 鲁棒性强。相较于深度学习模型不需要精细调参便能取得近似的效果。
* LightGBM直接支持缺失值与类别特征，无需对数据额外进行特殊处理。
* 速度更快
* LightGBM采用了直方图算法将遍历样本转变为遍历直方图，极大的降低了时间复杂度；
* LightGBM在训练过程中采用单边梯度算法过滤掉梯度小的样本，减少了大量的计算；
* LightGBM采用了基于Leaf-wise算法的增长策略构建树，减少了很多不必要的计算量；
* LightGBM采用优化后的特征并行、数据并行方法加速计算，当数据量非常大的时候还可以采用投票并行的策略；
* LightGBM对缓存也进行了优化，增加了缓存命中率；
* 内存更小
* LightGBM使用了直方图算法将特征值转变为bin值，且不需要记录特征到样本的索引，将空间复杂度从O(2\*#data)降低为O(#bin)，极大的减少了内存消耗；
* LightGBM在训练过程中采用互斥特征捆绑算法减少了特征数量，降低了内存消耗。

## 2.2 LightGBM的主要缺点

* 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；
* Boosting族是迭代算法，每一次迭代都根据上一次迭代的预测结果对样本进行权重调整，所以随着迭代不断进行，误差会越来越小，模型的偏差（bias）会不断降低。由于LightGBM是基于偏差的算法，所以会对噪点较为敏感；
* 在寻找最优解时，依据的是最优切分变量，没有将最优解是全部特征的综合这一理念考虑进去；
* 相对于深度学习模型无法对时空位置建模，不能很好地捕获图像、语音、文本等高维数据。在拥有海量训练数据，并能找到合适的深度学习模型时，深度学习的精度可以遥遥领先LightGBM。

1. **实例**

## 3.1参数

## 3.2调参方法

对于基于决策树的模型，调参的方法都是大同小异。一般都需要如下步骤：

1、首先选择较高的学习率，大概0.1附近，为了加快收敛的速度；

2、对决策树基本参数调参；

3、正则化参数调参；

4、最后降低学习率，这里是为了最后提高准确率。

**Step1：学习率learning\_rate和估计器boosting/boot/booting\_type及其数目**

先把学习率先定一个较高的值，如0.1，其次确定估计器boosting/boost/boosting\_type的类型，不过默认都会选gbdt。

为了确定估计器的数目（boosting迭代的次数），也可以说是残差树的数目，参数名为n\_estimators/num\_iterations/num\_round/num\_boost\_round。可以先将该参数设成一个较大的数，然后在cv结果中查看最优的迭代次数。

**Step2：max\_depth 和 num\_leaves —> 提高精确度的最重要的参数**

max\_depth：设置树深度，深度越大可能过拟合

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)，但是它的值的设置应该小于2^(max\_depth)，否则可能会导致过拟合。

（可以用sklearn里的GridSearchCV()函数进行搜索。）

**Step3：min\_data\_in\_leaf 和 min\_sum\_hessian\_in\_leaf🡪降低过拟合**

min\_data\_in\_leaf也叫min\_child\_samples，它的值取决于训练数据的样本个数和num\_leaves，将其设置的较大可以避免生成一个过深的树，但有可能导致欠拟合。

**Step4：feature\_fraction 和 bagging\_fraction🡪降低过拟合**

feature\_fraction进行特征的子抽样，用来防止过拟合及提高训练速度。

bagging\_fraction+bagging\_freq参数必须同时设置，bagging\_fraction相当于subsample样本采样，可以使bagging更快的运行，同时也可以降拟合。bagging\_freq默认0，表示bagging的频率，0意味着没有使用bagging，k意味着每k轮迭代进行一次bagging。

**Step5：正则化参数**

正则化参数lambda\_l1(reg\_alpha), lambda\_l2(reg\_lambda)，毫无疑问，是降低过拟合的，两者分别对应l1正则化和l2正则化。

**Step6：降低learning\_rate**

之前使用较高的学习速率是为了更快收敛，最后使用较低的学习速率，以及使用更多的决策树n\_estimators来训练数据，代入之前优化好的参数。



1. **一些思考**

LightGBM与XGBoost的联系和区别有哪些？有哪些改进？