参考: <https://blog.csdn.net/weixin_39807102/article/details/81912566>

<https://blog.csdn.net/qq_32577043/article/details/86215754>

<https://cloud.tencent.com/developer/news/375910>

​GBDT (Gradient Boosting Decision Tree) 是机器学习中一个长盛不衰的模型，其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 在工业界应用广泛，通常被用于点击率预测，搜索排序等任务。GBDT也是各种数据挖掘竞赛的致命武器，据统计 Kaggle 上的比赛有一半以上的冠军方案都是基于GBDT。

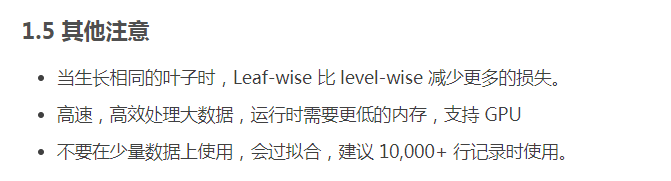
​LightGBM （Light Gradient Boosting Machine）是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有以下优点：

更快的训练速度

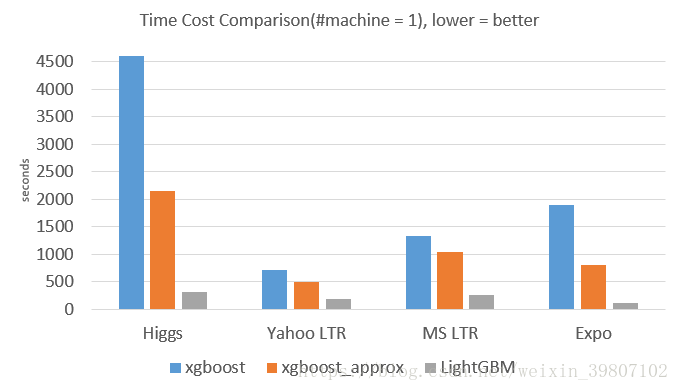
更低的内存消耗

更好的准确率

分布式支持，可以快速处理海量数据



​如下图，在 Higgs 数据集上 LightGBM 比 XGBoost 快将近 10 倍，内存占用率大约为 XGBoost 的1/6，并且准确率也有提升。



​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。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

Pre-sorted 方法：前已有的 GBDT 工具基本都是基于**预排序**的方法（pre-sorted）的决策树算法(如 xgboost)

1. 对所有的特征按照特征的数值进行预排序
2. 其次，在遍历分割点的时候用O(#data)代价找到一个特征上的最好分割点
3. 最后，找到一个特征的分割点都，将数据集分裂成左右子节点
4. 这样的排序算法的有点：能够找到分割点

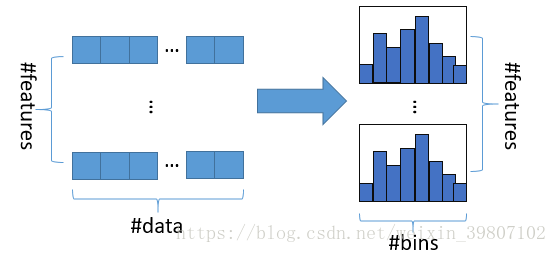
缺点：

空间消耗大，不仅需要保存数据的特征值，还保存了排序特征的结果（排序后的索引，为了后续快速的计算分割点），这里需要消耗训练数据两倍的内存。

时间上的消耗大，在遍历每一个分割点时，都要进行分割增益的计算，消耗的代价大

最后，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顺序是不一样的，无法对cache进行优化，同时，在每一层长树的时候，需要随机访问一个行索引到叶子索引的数组，并且不同的特征访问的顺序也不一样，也会造成较大的cacha miss

Histogram算法：直方图的基本思想是先把连续的浮点特征值离散化成K个整数，同时构造一个宽度为K的直方图，在遍历数据的时候，根据离散化的值作为索引在直方图中累计统计量，当遍历一次数据后，直方图累积了需要的统计量，根据直方图的离散值，遍历寻找最优的分割点。



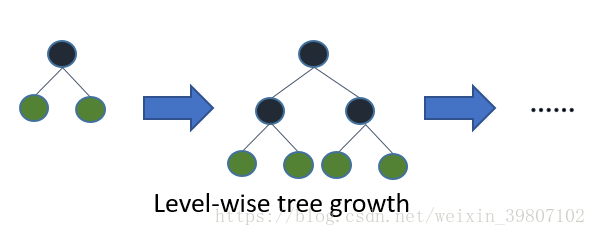
**优点：**降低内存消耗，直方图不仅不需要额外存储排序的结果，而且只可以保存特征离散化后的值，而这个值一般用8位整型存储就够了，内存消耗可以降低为原来的1/8

计算的代价上，预排序算法每遍历一个特征值就需要计算一次分裂的增益，而直方图只需要计算k次，时间复杂度从O(#data\*#features)优化到O(#k\*#features)

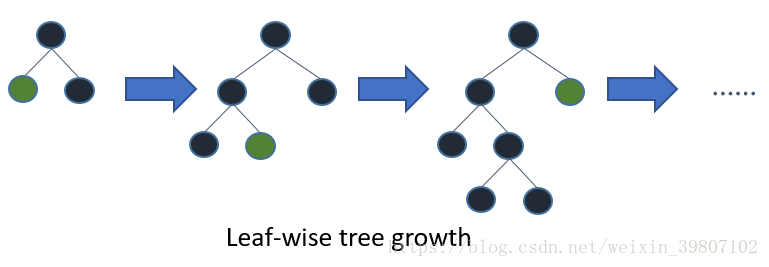
**缺点中包含的优点：**Histogram 算法并不是完美的。由于特征被离散化后，找到的并不是很精确的分割点，所以会对结果产生影响。但在不同的数据集上的结果表明，离散化的分割点对最终的精度影响并不是很大，甚至有时候会更好一点。原因是决策树本来就是弱模型，分割点是不是精确并不是太重要；较粗的分割点也有正则化的效果，可以有效地防止过拟合；即使单棵树的训练误差比精确分割的算法稍大，但在梯度提升（Gradient Boosting）的框架下没有太大的影响。

leaf-wise VS level-wise

在 Histogram 算法之上，LightGBM 进行进一步的优化。首先它抛弃了大多数 GBDT 工具使用的按层生长 (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，也好控制模型复杂度，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。



数据并行和特征并行

LightGBM 还具有支持高效并行的优点。LightGBM 原生支持并行学习，目前支持特征并行和数据并行的两种。

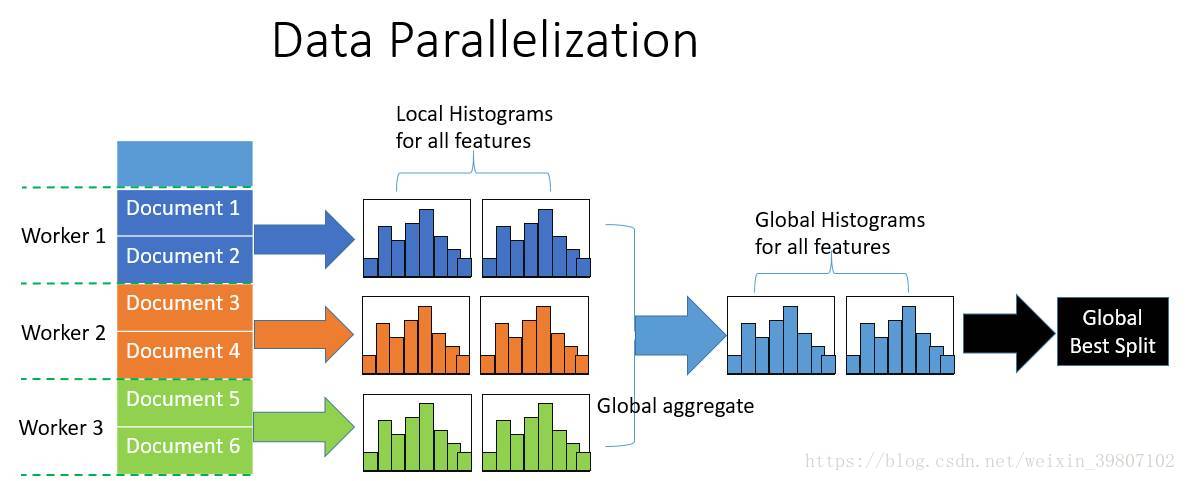
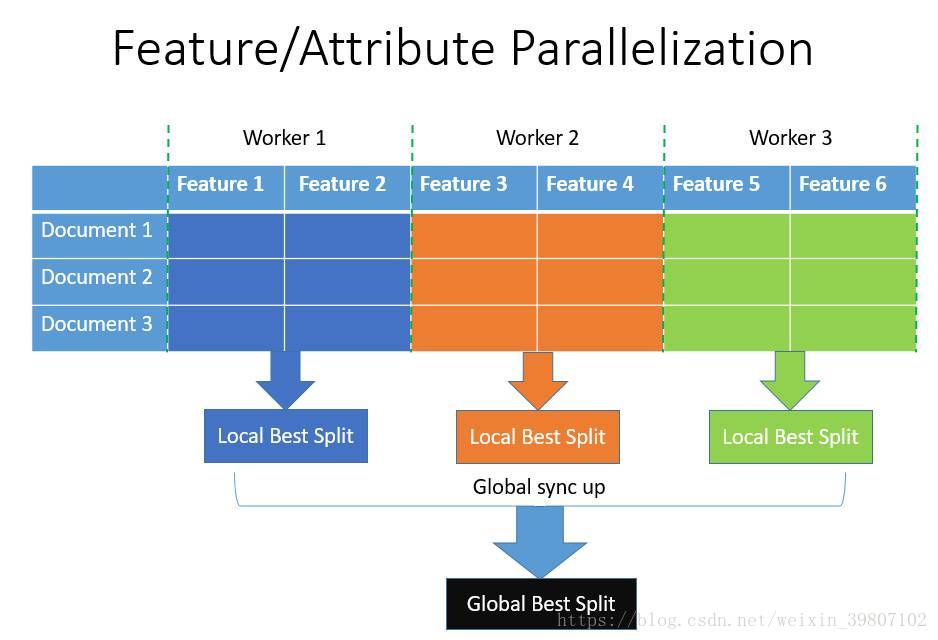
特征并行的主要思想是在不同机器在不同的特征集合上分别寻找最优的分割点，然后在机器间同步最优的分割点。

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。

​ LightGBM 针对这两种并行方法都做了优化：

在特征并行算法中，通过在本地保存全部数据避免对数据切分结果的通信；

在数据并行中使用分散规约 (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。



支持类别特征：

实际上大多数机器学习工具都无法直接支持类别特征，一般需要把类别特征，转化到多维的0/1 特征，降低了空间和时间的效率。而类别特征的使用是在实践中很常用的。基于这个考虑，LightGBM 优化了对类别特征的支持，可以直接输入类别特征，不需要额外的0/1 展开。并在决策树算法上增加了类别特征的决策规则。在 Expo 数据集上的实验，相比0/1 展开的方法，训练速度可以加速 8 倍，并且精度一致。据我们所知，LightGBM 是第一个直接支持类别特征的 GBDT 工具。

​ LightGBM 的单机版本还有很多其他细节上的优化，比如 cache 访问优化，多线程优化，稀疏特征优化等等。优化汇总如下：



应用场景：

回归，预测

Sklearn参数：https://blog.csdn.net/weixin\_39807102/article/details/81912566