lightgbm,xgboost,gbdt 的区别与联系

按照这些方法出现的先后顺序叙述。

GBDT

梯度提升树实在提升树的基础上发展而来的一种使用范围更广的方法，当处理回归问题时，提升树可以看作是梯度提升树的特例(分类问题时是不是特例？)。 因为提升树在构建树每一步的过程中都是去拟合上一步获得模型在训练集上的残差。后面我们将会介绍，这个残存正好是损失函数的梯度，对应于 GBDT 每一步要拟合的对象。

主要思想

在目标函数所在的函数空间中做梯度下降，即把待求的函数模型当作参数，每一步要拟合目标函数关于上一步获得的模型的梯度，从而使得参数朝着最小化目标函数的方向更新。

一些特性

1. 每次迭代获得的决策树模型都要乘以一个缩减系数，从而降低每棵树的作用，提升可学习空间。
2. 每次迭代拟合的是一阶梯度。

XGBoost

XGBoost 是 GBDT 的一个变种，最大的区别是 xgboost 通过对目标函数做二阶泰勒展开，从而求出下一步要拟合的树的叶子节点权重（需要先确定树的结构），从而根据损失函数求出每一次分裂节点的损失减小的大小，从而根据分裂损失选择合适的属性进行分裂。

这个利用二阶展开的到的损失函数公式与分裂节点的过程是息息相关的。先遍历所有节点的所有属性进行分裂，假设选择了这个 a 属性的一个取值作为分裂节点，根据泰勒展开求得的公式可计算该树结构各个叶子节点的权重，从而计算损失减小的程度，从而综合各个属性选择使得损失减小最大的那个特征作为当前节点的分裂属性。依次类推，直到满足终止条件。

一些特性

- 1.除了类似于 GBDT 的缩减系数外，xgboost 对每棵树的叶子节点个数和权重都做了惩罚，避免过拟合
- 2.类似于随机森林，XGBoost 在构建树的过程中，对每棵树随机选择一些属性作为分裂属性。

- 3.分裂算法有两种，一种是精确的分裂，一种是近似分裂算法，精确分裂算法就是把每个属性的每个取值都当作一次阈值进行遍历，采用的决策树是 CART。近似分裂算法是对每个属性的所有取值进行分桶，按照各个桶之间的值作为划分阈值，xgboost 提出了一个特殊的分桶策略，一般的分桶策略是每个样本的权重都是相同 的，但是 xgboost 使每个样本的权重为损失函数在该样本点的二阶导(泰勒展开不应该是损失函数关于模型的展开吗？为什么会有在该样本点的二阶导这种说法？ 因为模型是对所有样本点都通用的，把该样本输入到二阶导公式中就可以得到了)。

- 4.xgboost 添加了对稀疏数据的支持，在计算分裂收益的时候只利用没有 missing 值的那些样本，但是在推理的时候，也就是在确定了树的结构，需要将样本映射到叶子节点的时候，需要对含有缺失值的样本进行划分，xgboost 分别假设该样本属于左子树和右子树，比较两者分裂增益，选择增益较大的那一边作为该样本的分裂方向。

- 5.xgboost 在实现上支持并行化，这里的并行化并不是类似于 rf 那样树与树之间的并行化，xgboost 同 boosting 方法一样，在树的粒度上是串行的，但是在构建树的过程中，也就是在分裂节点的时候支持并行化，比如同时计算多个属性的多个取值作为分裂特征及其值，然后选择收益最大的特征及其取值对节点分裂。

- 6.xgboost 在实现时，需要将所有数据导入内存，做一次 pre-sort（exact algorithm），这样在选择分裂节点时比较迅速。

缺点

- level-wise 建树方式对当前层的所有叶子节点一视同仁，有些叶子节点分裂收益非常小，对结果没影响，但还是要分裂，加重了计算代价。
- 预排序方法空间消耗比较大，不仅要保存特征值，也要保存特征的排序索引，同时时间消耗也大，在遍历每个分裂点时都要计算分裂增益(不过这个缺点可以被近似算法所克服)

lightGBM

- https://github.com/Microsoft/LightGBM/wiki/Features
  关于 lightGBM 的论文目前并没有放出来，只是从网上一些信息得出以下的一些与 xgboost 不同的地方：

- xgboost 采用的是 level-wise 的分裂策略，而 lightGBM 采用了 leaf-wise 的策略，区别是 xgboost 对每一层所有节点做无差别分裂，可能有些节点的增益非常小，对结果影响不大，但是 xgboost 也进行了分裂，带来了务必要的开销。 leaft-wise 的做法是在当前所有叶子节点中选择分裂收益最大的节点进行分裂，如此递归进行，很明显 leaf-wise 这种做法容易过拟合，因为容易陷入比较高的深度中，因此需要对最大深度做限制，从而避免过拟合。

- lightgbm 使用了基于 histogram 的决策树算法，这一点不同与 xgboost 中的 exact 算法，histogram 算法在内存和计算代价上都有不小优势。
  -. 内存上优势：很明显，直方图算法的内存消耗为(#data* #features * 1Bytes)(因为对特征分桶后只需保存特征离散化之后的值)，而 xgboost 的 exact 算法内存消耗为：(2 * #data * #features* 4Bytes)，因为 xgboost 既要保存原始 feature 的值，也要保存这个值的顺序索引，这些值需要 32 位的浮点数来保存。
  -. 计算上的优势，预排序算法在选择好分裂特征计算分裂收益时需要遍历所有样本的特征值，时间为(#data),而直方图算法只需要遍历桶就行了，时间为(#bin)
- 直方图做差加速
  -. 一个子节点的直方图可以通过父节点的直方图减去兄弟节点的直方图得到，从而加速计算。

- lightgbm 支持直接输入 categorical 的 feature
  -. 在对离散特征分裂时，每个取值都当作一个桶，分裂时的增益算的是”是否属于某个 category“的 gain。类似于 one-hot 编码。

- 但实际上 xgboost 的近似直方图算法也类似于 lightgbm 这里的直方图算法，为什么 xgboost 的近似算法比 lightgbm 还是慢很多呢？
  -. xgboost 在每一层都动态构建直方图， 因为 xgboost 的直方图算法不是针对某个特定的 feature，而是所有 feature 共享一个直方图(每个样本的权重是二阶导),所以每一层都要重新构建直方图，而 lightgbm 中对每个特征都有一个直方图，所以构建一次直方图就够了。
  -. lightgbm 做了 cache 优化？

- .lightgbm 哪些方面做了并行？
  -. feature parallel
  一般的 feature parallel 就是对数据做垂直分割（partiion data vertically，就是对属性分割），然后将分割后的数据分散到各个 workder 上，各个 workers 计算其拥有的数据的 best splits point, 之后再汇总得到全局最优分割点。但是 lightgbm 说这种方法通讯开销比较大，lightgbm 的做法是每个 worker 都拥有所有数据，再分割？（没懂，既然每个 worker 都有所有数据了，再汇总有什么意义？这个并行体现在哪里？？）
  -. data parallel
  传统的 data parallel 是将对数据集进行划分，也叫 平行分割(partion data horizontally)，分散到各个 workers 上之后，workers 对得到的数据做直方图，汇总各个 workers 的直方图得到全局的直方图。 lightgbm 也 claim 这个操作的通讯开销较大，lightgbm 的做法是使用”Reduce Scatter“机制，不汇总所有直方图，只汇总不同 worker 的不同 feature 的直方图(原理？)，在这个汇总的直方图上做 split，最后同步。