RF GBDT XGBoost都属于集成学习，集成学习的目的是通过结合多个基学习器的预测结果来改善单个学习器的泛化能力和鲁棒性。

根据个体学习器的生成方式，目前的集成学习方法大致分为两大类：即个体学习器之间存在强依赖关系、必须串行生成的序列化方法，以及个体学习器间不存在强以来关系、可同时生成的并行化方法；前者的代表是boosting，后者的代表是bagging和随机森林。

1、RF

1.1 原理

提到随机森林，就不得不提bagging，bagging可以理解为：放回抽样，多数表决（分类）或简单平均（回归），同时bagging的基学习器之间属于并列生成，不存在强依赖关系。

随机森林是bagging的扩展变体，它在以决策树为基学习器构建bagging集成的基础上，进一步在决策树的训练过程中引入了随机特征选择，因此可以概括RF包括四个部分：1）随机选择样本（放回抽样）2）随机选择特征 3）构建决策树 4）随机森林投票

随机选择样本和bagging相同，随机选择特征是指在树的构建中，会从样本集的特征集合中随机选择部分特征，然后再从这个子集中选择最优的属性用于划分，这种随机性导致随机森林的偏差会有稍微的增加（相比于单棵不随机树），但是由于随机森林的“平均”特性，会使得它的方差减小，而且方差的减小补偿了偏差的增大，因此总体而言是更好的模型。

在构建决策树的时候，RF每棵决策树都最大可能的进行生长而不进行剪枝；在对预测输出进行结合时，RF通常对分类问题使用简单投票法，回归任务使用简单平均法。

RF的重要特性时不用对其进行交叉验证或者使用一个独立的测试集获得无偏估计，它可以在内部进行评估，也就是说在生成的过程中可以

对误差进行无偏估计，由于每个基学习器只使用了训练集中约63.2%的样本，剩下约36.8%的样本可以用做验证集来对其泛化性能进行“包外估计”。

RF和bagging对比：RF的起始性能比较差，特别当只有一个基学习器时，随着学习器数目增多，随机森林通常会收敛到更低的误差。随机森林的训练效率也会高于bagging，因为在单个决策树的构建中，bagging使用的是“确定性”决策树，在选择特征划分节点时，要对所有的特征进行考虑，而随机森林使用的时“随机性”特征树，只需考虑特征的子集。

1.2 优缺点

优点：

1. 在数据集上表现良好，相比于其他算法有较大的优势（训练速度、预测准确度）
2. 能够处理很高维的数据，并且不用特征选择，而且在训练后，给出特征的重要性
3. 容易做成并行化方法

缺点：

在噪声较大的分类或者回归问题上会过拟合

2.GBDT

提GBDT之前，谈一下Boosting，Boosting是一种与Bagging很类似的技术。不论是Boosting还是Bagging，所使用的多个分类器类型都是一致的。但是在前者当中，不同的分类器是通过串行训练而获得的，每个新分类器都根据已训练的分类器的性能来进行训练。Boosting是通过关注被已有分类器错分的那些数据来获得新的分类器。

由于Boosting分类的结果是基于所有分类器的加权求和结果的，因此Boosting与Bagging不太一样，Bagging中的分类器权值是一样的，而

Boosting中的分类器权重并不相等，每个权重代表对应的分类器在上一轮迭代中的成功度。

2.1 原理

GBDT与传统的boosting区别较大，他的每一次计算都是为了减少上一次的残差，而为了消除残差，我们可以在残差减小的梯度方向上建立模型，所以说，在gradient boosting中，每个新模型的建立是为了使得之前的模型的残差往梯度方向下降的方法，与传统的boosting中关注错误分类的样本加权有很大区别。

GBDT中，关键就是利用损失函数的负梯度方向在当前模型的值作为残差的近似值，进而拟合一颗CART回归树。

GBDT会累加所有树的结果，而这种累加是无法通过分类完成的，因此GBDT的树都是CART回归树，而不是分类树（尽管GBDT调整后也可以用于分类但不代表GBDT的树为分类树）。

2.2 优缺点

GBDT的性能在RF的基础上又有一步提升，因此其优点也很明显。

1. 能灵活的处理各种类型的数据
2. 在相对较少的调参时间下，预测的准确度较高

当然由于它是boosting，因此基学习器之间存在串行关系，难以并行训练数据

3.XGBoost

3.1 原理

XGBoost的性能在GBDT上又有一步提升，而其性能也能通过各种比赛管窥一二。坊间对XGBoost最大的认知在于**其能够自动地运用CPU的多线程进行并行计算**，同时在算法精度上也进行了精度的提高。

由于GBDT在合理的参数设置下，往往要生成一定数量的树才能达到令人满意的准确率，在数据集较复杂时，模型可能需要几千次迭代运算。但是XGBoost利用并行的CPU更好的解决了这个问题。

其实XGBoost和GBDT的差别也较大，这一点也同样体现在其性能表现上，详见XGBoost与GBDT的区别。

3.2 GBDT和XGBoost区别

1. 传统的GBDT以CART树作为基学习器，XGBoost还支持线性分类器，这个时候XGBoost相当于L1和L2正则化的逻辑斯蒂回归（分类）或者线性回归（回归）
2. 传统的GBDT在优化的时候只用到一阶导数信息，XGBoost则对代价函数进行了二阶泰勒展开，得到一阶和二阶导数
3. XGboost在代价函数中加入了正则项，用于控制模型的复杂度。从权衡方差偏差来看，它降低了模型的方差，使学习出来的模型更加简单，防止过拟合，这也是XGBoost由于GBDT的一个特性
4. shrinkage(缩减），相当于学习速率（xgboost中的eta)。xgboost在进行完一次迭代时，会将叶子节点的权值乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间
5. 列抽样。xgboost借鉴了随机森林的做法，支持列抽样，不仅防止过拟合，还能减少计算
6. 对缺失值的处理。对于特征值有缺失的样本，xgboost可以自动学习出它的分裂方向
7. XGBoost工具支持并行。Boosting不是一种串行的结构吗?怎么并行 的？**注意XGBoost的并行不是tree粒度的并行**，XGBoost也是一次迭代完才能进行下一次迭代的（第t次迭代的代价函数里包含了前面t-1次迭代的预测值）。**XGBoost的并行是在特征粒度上的。**我们知道，决策树的学习最耗时的一个步骤就是**对特征的值进行排序（**因为要确定最佳分割点），XGBoost在训练之前，**预先对数据进行了排序，然后保存为block结构，后面的迭代 中重复地使用这个结构，大大减小计算量。这个block结构也使得并**

行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

3.3 xgboost缺点

1. 以level-wise建树方式对当前层的所有叶子节点一视同仁，有些叶子节点分列收益非常小，对结果没影响，但还是要分裂，加重了计算代价

2. 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，在遍历每个分裂节点时都要计算分类增益（不过这个缺点可以被近似算法所克服）

4.Lightgbm

4.1 与xgboost对比

1. xgboost采用的是level-wise的分类策略，而lightgbm是leaf-wise策略，区别是xgboost对每一层所有节点做无差别分类，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，带来了不必要的开销。leaf-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，因此容易陷入比较高的深度中，因此需要对最大深度做限制，避免过拟合。

2. lightgbm采用了基于直方图的决策树算法，这一点不同于Xgboost的exact算法，直方图在内存和计算代价上都有不少优势

1）内存上：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。

（2）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)

3、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。

4、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。

5、多线程优化

1.相同点

都是由多棵树组成，最终的结果都是由多棵树一起决定

2.不同点

1. 组成随机森林的树可以是分类树也可以是回归树，而GBDT只由回归树组成
2. 组成RT的树可以并行生成，而GBDT是串行生成
3. 随机森林的结果是多数表决决定的，GBDT是多棵树累加之和
4. 随机森林对异常值不敏感，而GBDT对异常值比较敏感
5. 随机森林是减少模型的方差，而GBDT是减少模型的偏差
6. 随机森林不需要进行特征归一化，GBDT需要