**随机森林、GBDT、XGBOOST高频面试题**

RF、GBDT和XGBoost都属于集成学习（Ensemble Learning），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依赖关系、可同时生成的并行化方法；前者的代表就是Boosting，后者的代表是Bagging和“随机森林”（Random Forest）。

一，随机森林

随机森林是一个用随机方式建立的，包含多个决策树的集成分类器。其输出的类别由各个树投票而定（如果是回归树则取平均）。随机森林的生成过程如下：

1. 从原始样本中采取有放回抽样的方法选取n个样本（随机选择样本）；
2. 对n个样本选取a个特征中的随机k个，用建立决策树的方法获得最佳分割点（随机选择特征）；
3. 重复m次，获得m个决策树（构建决策树）；
4. 对输入样例进行预测时，每个子树都产生一个结果，采用多数投票机制输出（随机森林投票-分类问题使用简单投票法，回归任务使用简单平均法）。

随机森林的随机性主要体现在两个方面：

1. 数据集的随机选取：从原始的数据集中采取有放回的抽样（bagging），构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。
2. 待选特征的随机选取：与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。

以上两个随机性能够使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

随机森林的优点：

1. 实现简单，训练速度快，可以并行实现，因为训练时树与树之间是相互独立的；
2. 相比单一决策树，能学习到特征之间的相互影响，且不容易过拟合；
3. 能处理高维数据（即特征很多），并且不用做特征选择，因为特征子集是随机选取的；
4. 对于不平衡的数据集，可以平衡误差；
5. 相比SVM，对特征缺失不敏感，因为待选特征也是随机选取；
6. 训练完成后可以给出哪些特征比较重要。

随机森林的缺点：

1. 在噪声过大的分类和回归问题还是容易过拟合；
2. 相比于单一决策树，它的随机性让我们难以对模型进行解释。

二，GBDT （Gradient Boost Decision Tree 梯度提升决策树）

GBDT是以决策树为基学习器的迭代算法，注意GBDT里的决策树都是回归树而不是分类树。Boost是”提升”的意思，一般Boosting算法都是一个迭代的过程，每一次新的训练都是为了改进上一次的结果。   
GBDT的核心就在于：每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量。比如A的真实年龄是18岁，但第一棵树的预测年龄是12岁，差了6岁，即残差为6岁。那么在第二棵树里我们把A的年龄设为6岁去学习，如果第二棵树真的能把A分到6岁的叶子节点，那累加两棵树的结论就是A的真实年龄；如果第二棵树的结论是5岁，则A仍然存在1岁的残差，第三棵树里A的年龄就变成1岁，继续学习。   
GBDT优点是适用面广，离散或连续的数据都可以处理，几乎可用于所有回归问题（线性/非线性），亦可用于二分类问题（设定阈值，大于阈值为正例，反之为负例）。缺点是由于弱分类器的串行依赖，导致难以并行训练数据。

三、XGBOOST

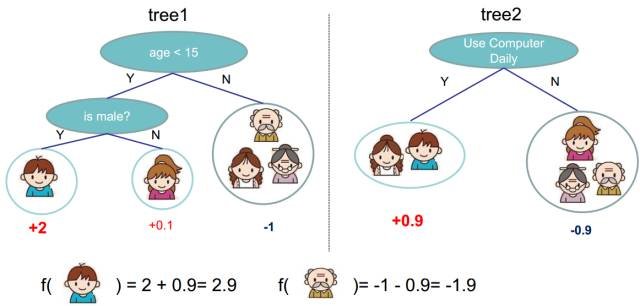
1. GBDT算法原理

XGBoost实现的是一种通用的Tree Boosting算法，此算法的一个代表为梯度提升决策树（Gradient Boosting Decision Tree, GBDT），又名MART(Multiple Additive Regression Tree)。

GBDT的原理是，首先使用训练集和样本真值（即标准答案）训练一棵树，然后使用这棵树预测训练集，得到每个样本的预测值，由于预测值与真值存在偏差，所以二者相减可以得到“残差”。接下来训练第二棵树，此时不再使用真值，而是使用残差作为标准答案。两棵树训练完成后，可以再次得到每个样本的残差，然后进一步训练第三棵树，以此类推。树的总棵数可以人为指定，也可以监控某些指标（例如验证集上的误差）来停止训练。

在预测新样本时，每棵树都会有一个输出值，将这些输出值相加，即得到样本最终的预测值。

使用两棵树来预测一个人是否喜欢电脑游戏的示意图如下，小男孩和老人的预测值为两棵树预测值的加和。



2. XGBoost所做的改进

2.1. 损失函数从平方损失推广到二阶可导的损失

GBDT的核心在于后面的树拟合的是前面预测值的残差，这样可以一步步逼近真值。然而，之所以拟合残差可以逼近到真值，是因为使用了平方损失作为损失函数，公式如下

如果换成是其他损失函数，使用残差将不再能够保证逼近真值。XGBoost的方法是，将损失函数做泰勒展开到第二阶，使用前两阶作为改进的残差。可以证明，传统GBDT使用的残差是泰勒展开到一阶的结果，因此，GBDT是XGBoost的一个特例。注意：此处省略了严格的推导，详情请参阅陈天奇的论文。

2.2. 加入了正则化项

正则化方法是数学中用来解决不适定问题的一种方法，后来被引入机器学习领域。通俗的讲，正则化是为了限制模型的复杂度的。模型越复杂，就越有可能“记住”训练数据，导致训练误差达到很低，而测试误差却很高，也就是发生了“过拟合”。在机器学习领域，正则化项大多以惩罚函数的形式存在于目标函数中，也就是在训练时，不仅只顾最小化误差，同时模型复杂度也不能太高。

在决策树中，模型复杂度体现在树的深度上。XGBoost使用了一种替代指标，即叶子节点的个数。此外，与许多其他机器学习模型一样，XGBoost也加入了L2正则项，来平滑各叶子节点的预测值。

2.3. 支持列抽样

列抽样是指，训练每棵树时，不是使用所有特征，而是从中抽取一部分来训练这棵树。这种方法原本是用在随机森林中的，经过试验，使用在GBDT中同样有助于效果的提升。

3.为什么XGBoost效果这么好

XGBoost是boosting算法中的一种，其他的还包括AdaBoost等。Boosting方法是目前最好的机器学习方法之一，关于其优良的学习效果，已有理论解释包括偏差-方差分解和Margin理论，但都不完美。下面结合个人理解做一些通俗的讨论。

机器学习就是模型对数据的拟合。对于一组数据，使用过于复杂的模型去拟合，往往会发生过拟合，这时就需要引入正则化项来限制模型复杂度，然而正则化项的选取、正则化系数的设定都是比较随意的，也比较难做到最佳。而如果使用过于简单的模型，由于模型能力有限，很难把握数据中蕴含的规律，导致效果不佳。

Boosting算法比较巧妙，首先使用简单的模型去拟合数据，得到一个比较一般的结果，然后不断向模型中添加简单模型（多数情况下为层数较浅决策树），随着树的增多，整个boosting模型的复杂度逐渐变高，直到接近数据本身的复杂度，此时训练达到最佳水平。

因此，boosting算法要取得良好效果，要求每棵树都足够“弱”，使得每次增加的复杂度都不大，同时树的总数目要足够多。XGBoost中，对每棵树的叶子节点数做了惩罚，从而限制了叶子节点的增长，使得每棵树都是“弱”的，同时还引入了学习速率，进一步降低了每棵树的影响。这样做的代价是，数的总数目会多一些，但从其取得的效果上看，这样做是值得的。

4. 为什么XGBoost运行这么快

4.1. 连续型特征的处理

决策树在训练时需要进行分叉，对于连续型特征，枚举所有可能分叉点将会十分耗时。一种近似方法是只枚举若干个分位点，例如将所有样本根据此特征进行排序，然后均分10份，两份之间断开的数值即为分位点，枚举所有9个分位点后，使用降低损失最多的那个作为分叉点。

4.2. 利用数据稀疏性

数据稀疏有三个原因：缺失数据；某些特征本身就含有大量的0；对离散特征做了one-hot处理。无论是哪种，都会导致样本中出现大量的0。通常，利用稀疏性可以提高运算效率。XGBoost的方法是，每次分叉时，都指定一条默认分支，如果样本的这个特征为0，就走这个默认分支。这样，训练时不必考虑这些0元素，大大提高了运算速度。陈天奇的实验表明，此方法在稀疏数据上可以提高50倍。

4.3. 数据的预排序和分块存储

分叉的时候为了判断分叉点，需要对每个特征进行排序。这个步骤是要重复多次的，因此XGBoost在训练之前预先对数据进行每一列做了排序，并按列存储到内存中。在分布式环境下，可以进行分块存储。

4.4. 减少读写相关，提高Cache命中率

由于预排序的数据是按列存储的，但训练时并不总是按列读取和写回，在需要按行读写的时候，将需要的行预先收集到一块连续内存上，再进行计算。这样由于是连续内存地址，可以提高Cache命中率，从而提高了运算速度。

4.5. 数据量大时，提高硬盘吞吐率

当数据量很大，不能装入内存时，需要将一部分数据放在硬盘里。然而硬盘读写速度慢，会严重影响计算效率。XGBoost使用了两种方法提高吞吐率，一个是对存储的内容进行压缩，读取时再进行解压，这相当于在读取代价和解压代价之间做了一个权衡。另一个方法是做数据分片，即在多块硬盘上存储数据，然后同时读写，从而提高读写速度。

四、随机森林和GBDT的区别：

1. 随机森林采用的bagging思想，而GBDT采用的boosting思想。这两种方法都是Bootstrap思想的应用，Bootstrap是一种有放回的抽样方法思想。虽然都是有放回的抽样，但二者的区别在于：Bagging采用有放回的均匀取样，而Boosting根据错误率来取样（Boosting初始化时对每一个训练样例赋相等的权重1／n，然后用该算法对训练集训练t轮，每次训练后，对训练失败的样例赋以较大的权重），因此Boosting的分类精度要优于Bagging。Bagging的训练集的选择是随机的，各训练集之间相互独立，弱分类器可并行，而Boosting的训练集的选择与前一轮的学习结果有关，是串行的。
2. 组成随机森林的树可以是分类树，也可以是回归树；而GBDT只能由回归树组成。
3. 组成随机森林的树可以并行生成；而GBDT只能是串行生成。
4. 对于最终的输出结果而言，随机森林采用多数投票等；而GBDT则是将所有结果累加起来，或者加权累加起来。
5. 随机森林对异常值不敏感；GBDT对异常值非常敏感。
6. 随机森林对训练集一视同仁；GBDT是基于权值的弱分类器的集成。
7. 随机森林是通过减少模型方差提高性能；GBDT是通过减少模型偏差提高性能。

五、GBDT与xgboost的区别

1.传统GBDT以CART作为基分类器，xgboost还支持线性分类器，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。

2.传统GBDT在优化时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，同时用到了一阶和二阶导数。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。

3.Xgboost在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的score的L2模的平方和。从Bias-variance tradeoff角度来讲，正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合，这也是xgboost优于传统GBDT的一个特性。

4.Shrinkage（缩减），相当于学习速率（xgboost中的eta）。xgboost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把eta设置得小一点，然后迭代次数设置得大一点。（补充：传统GBDT的实现也有学习速率）

5.列抽样（column subsampling）。xgboost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是xgboost异于传统gbdt的一个特性。

6.缺失值的处理。对于特征的值有缺失的样本，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。

8.可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。