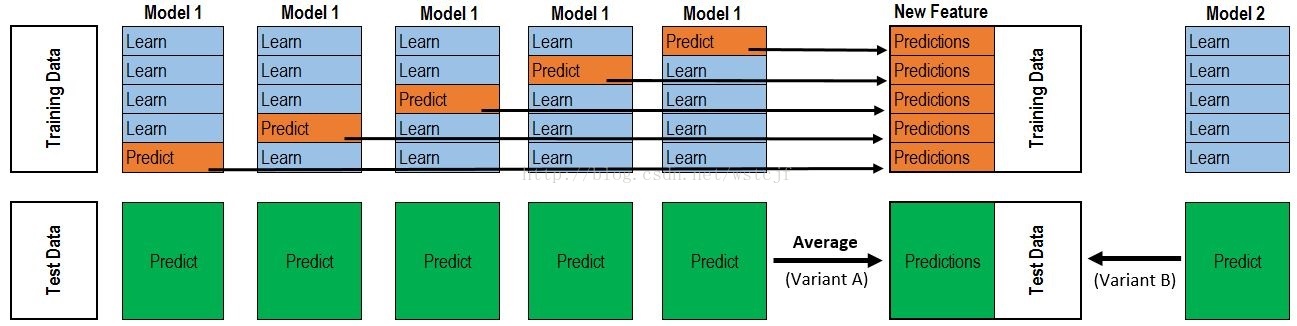
**Ensemble**

**I. Stacking**

**请简述stacking算法的工作原理。**



Stacking方法将数据集平均分为部分，然后随机组合份子集作为训练集，另外一份作为测试集，组合出数据集训练出个模型并对剩下一份训练集作为预测得到新的数据集，在用这份数据集训练出更高级的模型。

**II. Bagging**

**1. 随机森林的工作原理是什么？**

1. 样本随机：从样本集合中用Bootstrap随机选取n个样本；
2. 特征随机：从所有属性中随机选取k个属性，选择最佳分割属性作为节点建立CART决策树；
3. 重复以上两步m次，即建立了m棵CART决策树；
4. 通过m棵CART决策树形成随机森林，通过投票表决记过，决定结果。

**2. 随机森林如何处理缺失值？**

1. 简单粗暴，对于训练集，同一个类别下的数据，如果是分类变量缺失，用众数补上，如果是连续型变量缺失，用中位数补。
2. 先用中位数（对于连续变量）或众数（对于分类变量）补上缺失值，然后构建森林并计算proximity matrix，再回头看缺失值，如果是分类变量，则用没有缺失的观测实例的proximity中的权重进行投票。如果是连续型变量，则用proximity矩阵进行加权平均的方法补缺失值，然后迭代多次。

proximity matrix其实就是任意两个观测实例间的相似度矩阵。原理是如果两个观测实例落在同一棵树的同一个叶子节点的次数越多，则这两个观测实例的相似度越高。这个矩阵进行了归一化，就是除以数的棵树。

**3. 随机森林如何评估特征重要性？**

在随机森林中某个特征X的重要性的计算方法如下：

1. 对于随机森林中的每一颗决策树,使用相应的OOB(袋外数据)数据来计算它的袋外数据误差,记为；
2. 随机地对袋外数据OOB所有样本的特征X加入噪声干扰(就可以随机的改变样本在特征X处的值),再次计算它的袋外数据误差,记为；
3. 假设随机森林中有棵树，那么对于特征X的重要性为

之所以可以用这个表达式来作为相应特征的重要性的度量值是因为若给某个特征随机加入噪声之后，袋外的准确率大幅度降低，则说明这个特征对于样本的分类结果影响很大，也就是说它的重要程度比较高。

**4. 什么是OOB？随机森林中OOB是如何计算的？它有什么优缺点？**

Bagging方法中Bootstrap每次约有的样本不会出现在Bootstrap所采集的样本集合中，当然也就没有参加决策树的建立，把这的数据称为袋外数据OOB，它可以用于取代测试集误差估计方法。

袋外数据(OOB)误差的计算方法如下：

对于已经生成的随机森林，用袋外数据测试其性能,假设袋外数据总数为O，用这O个袋外数据作为输入，带进之前已经生成的随机森林分类器，分类器会给出O个数据相应的分类，因为这O条数据的类型是已知的，则用正确的分类与随机森林分类器的结果进行比较,统计随机森林分类器分类错误的数目，设为X，则袋外数据误差大小=X/O；这已经经过证明是无偏估计的，所以在随机森林算法中不需要再进行交叉验证或者单独的测试集来获取测试集误差的无偏估计。

OOB的概率为，当很大时

**5. 随机森林的随机性体现在哪里？**

对于森林中的单棵树，分类强度越大越好。对于森林中的多棵树，树之间的相关度越小越好。随机森林的随机性有两个方面：数据的随机性选取，以及待选特征的随机选取。

1. 数据的随机选取

第一，从原始的数据集中采取有放回的抽样，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。

第二，利用子数据集来构建子决策树，将这个数据放到每个子决策树中，每个子决策树输出一个结果。最后，如果有了新的数据需要通过随机森林得到分类结果，就可以通过对子决策树的判断结果的投票，得到随机森林的输出结果了。

1. 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。这样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。

**6. 随机森林的优缺点是什么？**

1. 优点：

在当前的很多数据集上，相对其他算法有着很大的优势，表现良好；

它能够处理很高维度特征的数据，并且不用做特征选择；

在训练完后，它能够给出哪些特征比较重要；

在创建随机森林的时候，对泛化误差使用的是无偏估计，模型泛化能力强；

训练速度快，容易做成并行化方法；

在训练过程中，能够检测到特征间的互相影响；

实现比较简单；

对于不平衡的数据集来说，它可以平衡误差；

如果有很大一部分的特征遗失，仍可以维持准确度。

1. 缺点：

随机森林已经被证明在某些噪音较大的分类或回归问题上会过拟；

对于有不同取值的属性的数据，取值划分较多的属性会对随机森林产生更大的影响，所以随机森林在这种数据上产出的属性权值是不可信的。

**III. Boosting**

**1. Boosting算法的原理是什么？**

一般来说，找到弱学习算法要相对容易一些，然后通过反复学习得到一系列弱分类器，组合这些弱分类器得到一个强分类器。Boosting算法要涉及到两个部分，加法模型和前向分步算法。加法模型就是说强分类器由一系列弱分类器线性相加而成。一般组合形式如下：

其中，就是一个个的弱分类器，是弱分类器学习到的最优参数，就是弱学习在强分类器中所占比重，是所有和的组合。这些弱分类器线性相加组成强分类器。

前向分步就是说在训练过程中，下一轮迭代产生的分类器是在上一轮的基础上训练得来的。也就是可以写成这样的形式：

由于采用的损失函数不同，Boosting算法也因此有了不同的类型，AdaBoost就是损失函数为指数损失的Boosting算法。

**2. AdaBoost算法的工作原理是什么？**

输入：训练数据集，其中，，，迭代次数

训练：

1. 初始化训练样本的权值分布：。
2. 对于

使用具有权值分布的训练数据集进行学习，得到弱分类器

计算在训练数据集上的分类误差率：

计算在强分类器中所占的权重：

更新训练数据集的权值分布（这里，是归一化因子，为了使样本的概率分布和为1）：

1. 输出：得到最终分类器

**3. 如何推导AdaBoost算法的弱学习权值？**

假设已经经过轮迭代，得到，根据前向分步，可以得到：

由于AdaBoost是采用指数损失，由此可以得到损失函数：

这时候，是已知的，可以作为常量移到前面去:

其中，，这个就是每轮迭代的样本权重，依赖于前一轮的迭代重分配。化简一下：

化简Loss：

重写一下，

这样就得到了化简之后的损失函数。对求偏导，令得到：

**4. AdaBoost有何优缺点？**

优点：

1. 容易理解，实现简单
2. 易编码
3. 分类精度高
4. 可以使用各种回归模型构建基分类器，非常灵活
5. 作为二元分类器是，构造简单、结果可理解、少参数
6. 相对来说，不宜过拟合

缺点：

1. 只能串行
2. 对异常值敏感：boosting对异常值敏感

**5. Gradient Boosting的工作原理是什么？**

梯度提升算法假设一个真值，然后通过加权和形式的得到近似函数，被称为基学习器：

根据经验风险最小化原理，可以通过对训练集样本平均损失求极小值得到。开始时，模型由一个常函数组成，然后利用贪心策略增量扩展模型：

其中是基学习器。

不幸的是，在任意损失函数L的每个步中选择最佳函数h，一般来说这是一个计算上不可行的优化问题。因此我们可以通过一些限制简化问题。简化的办法就是使用梯度下降法解决最小化问题。假设我们考虑连续的情况，比如是一个定义在上可微函数的集合，我们可以根据下式更新模型：

其中导数是关于求导所得。然后在离散的情况下，比如函数集合是有限的，我们将选择最接近损失函数梯度的候选函数，其中可以借助上式的线性搜索进行计算。注意，这是一个启发式的方法，因此不能得到精确解，但是得到的近似解已经足够精确了。

通用梯度提升方法的伪代码如下:

输入：训练集，可微损失函数，迭代次数

算法如下:

1. 使用常量初始化模型：

使用损失函数的负梯度在当前模型上的值近似代替残差：

使用基学习器拟合近似的残差值，即用训练集进行拟合

解一维优化问题计算乘子：

更新模型：

1. 输出

**6. GBDT的工作原理是什么？**

当基学习器是一个包含个节点的回归树时，可以写成：

简单点说就是，对于回归树，如果被归到某个叶子节点，那么在这个中得到的预测值就是叶子节点的值。一般用叶子节点上的平均值近似节点的值

是不相交的区域，集合覆盖预测值的空间，可以认为是回归树的系数。

回归提升树的预测函数：

令，这个值是叶子节点的最理想的常数更新值，也可以认为兼顾了下降方向和下降步长。

综上，整理一下回归提升树的整个算法流程：

输入：训练数据集，迭代次数

1. 初始化：，表示样本量

计算损失函数在当前模型的负梯度值：

根据学习得到了第棵回归树，对应的叶节点区域为

得到本轮最佳拟合决策树为：

得到本次迭代得到的强学习器：

1. 结束迭代之后的强学习器为：

发现损失函数在当前模型的负梯度值刚刚好是，也就是常说的残差，下面给出证明：

令当前的预测函数模型为，下一棵要学习的回归树为，则下一步要学习的预测函数为：

回归提升树的损失函数为平方损失：

对求关于的偏导：

要使经验风险极小化，令偏导为0：

也就是说，构建的回归树要去拟合前面的残差，得证。可以这么理解，GBDT是在函数空间的梯度下降求最优值，当损失函数为平方损失时，恰好去拟合了残差。

**7. 原始的boosting算法开始时，为每一个样本赋上一个权重值，然后根据预测结果改变权重，那么GBDT算法中并未有权重的改变，哪里体现了boosting思想？**

Gradient Boosting与Boosting区别在于，每一计算的是为了减少上一次的残差，下一个模型主要在残差减少的梯度方上建立模型，使得残差往梯度方向上减少。虽然不同，但是GBDT算法会更关注那些梯度比较大的样本，和Boosting思想类似。

**8. GBDT与传统Boosting（AdaBoost）的区别是什么？**

1. GBDT也属于Boosting算法，但与传统Boosting有区别，它通过拟合上一步的残差，传统意义上说不能并行，只能用CART回归树，降低偏差。
2. 迭代思路不同：传统boosting对训练样本进行加权，GBDT则是拟合残差，下一棵树沿残差梯度下降的方向进行拟合。

**9. GBDT正则化的方式有哪些？**

1. 和AdaBoost类似的正则化项，即步长。定义为，对于前面的弱学习器的迭代有

如果加上了正则化项，则有:

的取值范围为。对于同样的训练集学习效果，较小的意味着需要更多的弱学习器的迭代次数。通常用步长和迭代最大次数一起来决定算法的拟合效果。

1. 对于弱学习器即CART回归树进行正则化剪枝。
2. 通过子采样比例：取值为。注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在之间。由于使用了子采样，程序可以通过采样分发到不同的任务去做Boosting的迭代过程，最后形成新树，从而减少弱学习器难以并行迭代的弱点。

**10. GBDT如何训练组合特征？为什么使用GBDT而非RF？**

通过训练颗决策树，则样本落到每颗树的结点可以用一个one-hot向量表现，将所有的决策树得到的向量拼起来就可以得到全新的特征。由于树的每条路径是通过最小化均方差等方法最终分割出来的区分行路径，根据该路径得到的特征都相对有区分性。

GBDT前面的决策树的特征分裂主要体现对多数样本有区分度的特征，后面的决策树主要体现的是经过前面的决策树，残差仍较大的少数样本。优先选用在整体上有区分度的特征，再选用针对少数样本有区分度的特征，思路更加合理。

**11. GBDT如何用于多分类？**

使用多分类log损失作为损失函数：

对应的概率为（就是softmax）：

对于多分类问题，在构建基学习器时，要为每个类别创建一棵回归树，

因此，每次迭代，以概率角度来计算当前残差。

叶子节点值近似为：

**12. GBDT如何计算特征重要性？**

特征的全局重要度通过特征在单颗树中的重要度的平均值来衡量：

其中，是树的数量。特征在单颗树中的重要度的如下：

其中，为树的叶子节点数量，即为树的非叶子节点数量（构建的树都是具有左右孩子的二叉树），是和节点相关联的特征，是节点分裂之后平方损失的减少值。

**13. GBDT有何优缺点？**

优点：

1. 调参少的情况下，准确率也高（相比于SVM）。
2. 灵活处理各种数据，包括连续和离散，无需归一化处理（相比于LR）。
3. 模型非线性变换多，特征不用经过复杂处理即可表达复杂信息。
4. 从一定程度上可以防止过拟合，小步而非大步拟合。

缺点：

1. 一般来说传统的GBDT只能串行，但是也可以通过子采样比例（0.5~0.8）实现某种意义上的并行，但一般这就不叫GBDT了。
2. 对异常值敏感，但是可以采取一些健壮的损失函数缓解，如Huber/Quantile损失函数。

**14. XGBoost的工作原理是什么？**

由于GBDT除了L2损失函数之外，对其它的损失函数推导比较复杂。而XGBoost采用对损失函数进行二阶Taylor展开来近似，简化了求解与推导。

目标：

其中为损失函数，为正则项，包括L1，L2。

由于函数在点的泰勒展开形式为：

利用泰勒展开三项，做一个近似，我们可以很清晰地看到，最终的目标函数只依赖于每个数据点的在误差函数上的一阶导数和二阶导数。

定义树的复杂度：对于的定义做一下细化，把树拆分成结构部分和叶子权重部分。，结构函数把输入映射到叶子的索引号上面去，而给定了每个索引号对应的叶子分数是什么。定义这个复杂度包含了一棵树里面节点的个数，以及每个树叶子节点上面输出分数的L2模平方。当然这不是唯一的一种定义方式，不过这一定义方式学习出的树效果一般都比较不错。

其中为叶子个数，为的L2模平方。

在这种新的定义下，可以把目标函数进行如下改写，其中I被定义为每个叶子上面样本集合：

定义

最终公式可以化简为

通过对求导等于0，可以得到

然后把最优解代入得到：

枚举不同树结构的贪心法：每一次尝试去对已有的叶子加入一个分割

对于每次扩展，还是要枚举所有可能的分割方案，如何高效地枚举所有的分割呢？我假设我们要枚举所有这样的条件，对于某个特定的分割我们要计算左边和右边的导数和。我们可以发现对于所有的，我们只要做一遍从左到右的扫描就可以枚举出所有分割的梯度和和。然后用上面的公式计算每个分割方案的分数就可以了。观察这个目标函数，大家会发现第二个值得注意的事情就是引入分割不一定会使得情况变好，因为我们有一个引入新叶子的惩罚项。优化这个目标对应了树的剪枝，当引入的分割带来的增益小于一个阀值的时候，可以剪掉这个分割。大家可以发现，当我们正式地推导目标的时候，像计算分数和剪枝这样的策略都会自然地出现，而不再是一种因为启发式而进行的操作了。

**15. XGBoost为什么不使用三阶导数？**

1. 使用三阶导数要求损失函数三阶可导，相比于二阶导数，对损失函数的要求更高了，难以寻找满足要求的损失函数。
2. 三阶导数计算比较复杂，时间复杂度高。

**16. GBDT和XGBoost的区别是什么？**

1. 基分类器的选择：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. 二阶泰勒展开：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工具支持自定义损失函数，只要函数可一阶和二阶求导。
3. 方差-方差权衡：XGBoost在目标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分数的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）。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，一般把学习速率设置得小一点，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. 缺失值处理：XGBoost考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率。即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：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（第次迭代的损失函数里包含了前面次迭代的预测值）。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(块)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. 线程缓冲区存储：按照特征列方式存储能优化寻找最佳的分割点，但是当以行计算梯度数据时会导致内存的不连续访问，严重时会导致cache miss，降低算法效率。可先将数据收集到线程内部的buffer（缓冲区），主要是结合多线程、数据压缩、分片的方法，然后再计算，提高算法的效率。
9. 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。大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。

**17. 为什么XGBoost和GBDT在调参时为什么树的深度很少就能达到很高的精度？**

对于Boosting来说，每一步都会在上一轮的基础上更加拟合原数据，所以可以保证偏差，所以对于每个基分类器来说，问题就在于如何选择方差更小的分类器，即更简单的分类器，所以选择了深度很浅的决策树。

**IV. 综合问题**

**1. Bagging与Boosting的区别是什么？**

1. Bagging采用均匀取样，而Boosting根据错误率取样。
2. Bagging的各个预测函数没有权重，而Boosting是有权重的。
3. Bagging的各个预测函数可以并行生成，而Boosting的各个预测函数只能顺序生成。

**2. RF与GBDT之间有什么异同点？**

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：

组成随机森林的树可以分类树也可以是回归树，而GBDT只由回归树组成。

组成随机森林的树可以并行生成，而GBDT是串行生成。

随机森林的结果是多数表决表决的，而GBDT则是多棵树累加之和。

随机森林对异常值不敏感，而GBDT对异常值比较敏感。

随机森林是减少模型的方差，而GBDT是减少模型的偏差。

随机森林不需要进行特征归一化，而GBDT则需要进行特征归一化。

**3. 为什么说bagging是减少variance，而boosting是减少bias?**

对于Bagging算法来说，由于会并行地训练很多不同的分类器的目的就是降低这个方差，因为采用了相互独立的基分类器多了以后，的值自然就会靠近。所以对于每个基分类器来说，目标就是如何降低这个偏差，所以我们会采用深度很深甚至不剪枝的决策树。

对于Boosting来说，每一步都会在上一轮的基础上更加拟合原数据，所以可以保证偏差，所以对于每个基分类器来说，问题就在于如何选择方差更小的分类器，即更简单的分类器，所以选择了深度很浅的决策树。