0、概述

集成学习是将诸多基学习器组合成一个性能较高的学习器的方法。这些基学习器可以是弱学习器，集成学习将它们组合，能够得到一个强学习器（这在理论上可以证明）。

集成学习的集成策略（即基学习器的组合方式）主要有两种：Bagging和Boosting。下面分别介绍这两种集成策略。

**1、Bagging**

（1）Bagging的训练过程是：从训练集中使用boostrap方法（即自助采样法，也就是有放回地采样）随机采出多个子集，用每个子集训练一个基学习器。显然，这些基学习器的训练可以并行执行。

（2）Bagging的预测过程是：对于分类问题，将基学习器们的投票结果作为最终输出；对于回归问题，对基学习器们的输出作平均作为最终输出。

（3）Bagging的特点是：通过降低方差来提升基学习器的性能（多个模型投票或平均能够降低方差）。但是注意，Bagging策略能降低方差的关键是基学习器之间的“独立性”。但实际中无法实现完全独立，只能尽量增加各基学习器之间的“差异性”或者说基学习器们的“多样性”。

**2、Boosting**

（1）Boosting的训练过程是：先从初始的训练集训练出一个基学习器，再根据基学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多的关注，然后基于调整后的样本分布来训练下一个基学习器。如此重复进行，直至基学习器数目达到事先指定的值。显然，这些基学习器的训练需要串行执行。

（2）Boosting的预测过程是：将训练好的基学习器加权结合。

Boosting策略只给出了一般的思路，要想具体地实现它，还需要具体地考虑几个问题：训练集分布的调整如何计算，所有基学习器如何结合以及所有基学习器结合时每个基学习器的权重如何计算。

（3）Boosting的特点是：通过降低偏差来提升基学习器的性能。

集成学习最常用的基学习器就是决策树，基于决策树的集成主要有两类：随机森林和提升树，前者使用Bagging集成策略，后者使用Boosting集成策略。

除了提升树，Boosting还有一种具体的实现方式，那就是AdaBoost。

一、随机森林

随机森林是Bagging的一种实现。顾名思义，它使用决策树作为基学习器，并基于Bagging集成策略。（它在Bagging的基础做了一点小小的改动：Bagging对训练集进行部分采样时只在样本上随机采样，而随机森林还在特征上随机采样，也就是说，各个基学习器使用的样本的特征不完全一样）

我们知道，弱学习器可以集成为强学习器。对于随机森林来说，它使用不剪枝的CART决策树作为基学习器。基学习器很容易过拟合，因而拥有较弱的泛化性能。随机森林基于Bagging策略，并行地训练多个基学习器，使用投票或平均的方法集成这些基学习器，形成一个较强泛化性能的模型。但是这样做有一个前提，就是各基学习器要尽量差异化。实际上，最好是各个基学习器相互独立，但是实际中很难做到完全互相独立，所以随机森林算法只能尽量地差异化各个基学习器。如何差异化各个基学习器呢？一个自然的想法是，对每个基学习器，不使用整个训练集，而是使用训练集的一部分进行训练，训练出的基学习器自然会有差异。随机森林就是这样做的，它对训练集进行样本上和特征上的随机采样，采样得到的不同子集用来训练不同的基学习器。样本上的采样使用有放回随机采样（boostrap采样）；特征上的采样使用无放回的随机采样。

注意，这样的随机采样方法使得各个基学习器使用的子集之间会有一定的“重叠”，从而造成各个基学习器之间不会完全独立。但是不能为了使它们不重叠，就让各基学习器使用完全不相交的子集，因为这样会使子集容量太小，基学习器学到的信息太少，从而使基学习器准确率太低。

总之，Bagging通过对训练集进行部分采样，尽量增加各基学习器的差异程度，从而增强其泛化性能。

对于特征上的随机采样，每个子集的采样特征数一般设定为，其中d是总的特征数；对于样本上的随机采样，一般每个子集的容量为整个训练集的63.2%。

二、AdaBoost

AdaBoost算法是Boosting的一种实现，它是一个二类分类的模型。它没有指定具体的基学习器，所以它不是一个“完全具体”的算法，它有一定的扩展性。但是它指定了基学习器结合的方式（加权相加）以及损失函数（指数损失函数），从而能基于贪心优化方法导出其训练样本的权重调整的具体方式和基学习器权重的具体计算方法。

AdaBoost模型的形式为：



可以看出，它是把基学习器们加权相加的结果作为最终模型输出，这被称为“加性模型”。

给定训练集，其中，则AdaBoost的目标为：最小化指数损失函数



其中D表示训练集样本对应的总体的分布。等式的左边表示对于给定的分布D，模型的输出H的指数损失；右边表示服从分布D的随机变量x的函数项的期望，f(x)表示分布D上的“God模型”，即完美模型的输出。

当然，我们不可能知道总体的分布D和相应的God模型，但我们有训练集和训练集中的样本标签，因此损失函数可以进行相应的改造。

从AdaBoost的基本思路中我们可以得到一个启示：一般来说，机器学习模型的模型形式和目标函数可以分离，也就是说，每一种模型都可以使用不同的目标函数。这也是后面要介绍的XGBoost系统设计的一个思想，以此提高系统的可扩展性。

这里省略推导步骤，直接给出AdaBoost的算法流程：

给定训练集，其中，并给定基学习器以及基学习器的学习算法，

（1）初始化训练数据的权值分布为均匀分布：



（2）迭代N次，每次迭代训练一个基学习器、计算该基学习器的权重以及更新训练数据的权值分布。具体来说，第t次迭代执行如下操作：

（a）使用分布为Dt的训练数据学习一个基学习器（注意的输出是-1或+1）；

（b）计算在训练数据集上的分类误差率：



注意，这里体现了训练数据的权值分布的意义——它用于计算基学习器的分类误差率。

（c）根据分类误差率，计算的系数：



（d）更新训练数据集的权值分布：



其中，是规范化因子，它的作用是使成为一个规范的概率分布。

（3）对训练好的所有基学习器，将它们加权相加：



得到最终的分类器：



可以看出，基学习器的系数的计算是本着“误差率越高，系数越低”的原则；

而第i个训练样本权值的第t+1次更新是本着“若第t个模型对它预测正确，则减小其权重，若预测错误，则增加其权重”的原则，且增加或减小的程度与第t个模型的分类准确率有关，准确率越低，增加或减小的程度越高。这确实与Boosting的思想是一致的。

再次强调，在指定了加权相加的基学习器结合方式和指数损失函数后，AdaBoost模型的基学习器系数的计算方法和训练数据权值调整方法可以自然地导出，而不需要再特意设定。

有些人可能要问，AdaBoost为什么选择指数损失函数，而不选择其他的损失函

数呢？这可能是为了使它的算法流程中，基学习器系数的计算和训练数据权值的更新在形式上尽量简单吧。

三、提升树

**1、提升树**

提升树也是Boosting的一种实现。与随机森林相同，它也指定CART决策树为基学习器；与AdaBoost类似，它也基于加性模型。

提升树模型的形式为：



其中是第t棵决策树，是决策树的参数，N是树的个数。

可以看出，虽然同为加性模型，但AdaBoost是各基学习器的加权相加，提升树则是直接相加。

提升树采用贪心优化方法（又叫“前向分布算法”），即：不是一次性地求解整体的最优模型，而是每次只求解一个最优的决策树。其优化目标为：



然后把它加到已经训练好的模型（当前模型）上，得到更新的模型：



其中，i表示第几个样本，t表示第几个决策树；表示第t个决策树对第i个样本的预测值，表示前t-1个决策树结合后的模型（当前模型）对第i个样本的预测值。L表示损失函数，它是样本标签和模型预测值的函数。

可以看出提升树与AdaBoost的另一个区别：提升树支持各种损失函数，但AdaBoost指定损失函数为指数损失函数（如果不指定指数损失函数，则AdaBoost不能推导出具体的基学习器系数的计算和训练数据权值的更新方式）。

这里如果指定决策树为回归树，那么可以给出具体的形式：

我们知道，决策树的训练过程相当于将训练集划分成多个子集，每个子集对应于决策树的一个叶节点，决策树利用这些叶节点对样本进行预测——每个叶节点对应一个输出，决策树对某个样本的预测值就是该样本所属叶节点的输出值。对回归树而言，一个叶节点的输出是该叶节点中样本的标签值的平均。

设树T的各叶节点对应的子集为，对应的输出为，则决策树可以表示为：



其中I表示指示函数。

进一步，如果指定损失函数为MSE损失，则损失函数可以写成：



其中r是当前模型（前t-1次决策树训练后得到的结合模型）拟合数据的残差。也就是说，每个决策树训练时，以该决策树输出与当前模型的残差的MSE最小化为目标。

这样，我们就得到了回归提升树的具体算法：

给定训练集，注意，在讨论决策树模型时，“T”有时表示训练集，有时表示决策树模型，注意根据上下文区分。

（1）初始化；

（2）迭代N次，第t次迭代的操作如下：

（a）计算当前模型对各样本的的残差：；

（b）训练一个回归树来拟合残差，得到；

（c）更新模型

（3）得到最终的提升树模型

可以看出，AdaBoost与提升树是Boosting策略的两种不同实现方式，二者的区别在于：

（1）AdaBoost是二类分类模型，提升树则可以用于任意类分类和回归；

（2）AdaBoost没有指定基学习器的具体形式，提升树指定了CART树作为基学习器；

（3）AdaBoost的损失函数指定为指数损失函数，而提升树支持各种损失函数；

（4）AdaBoost对各基学习器的结合方式是加权相加，提升树对各基学习器的结合方式则是不带权的相加。

**2、梯度提升树**

梯度提升树（GBM或GBDT）是在提升树基础上实现的一种改进模型。

对于提升树，前面已经说过，它支持各种损失函数。当损失函数是MSE时，每一步优化就是拟合当前模型的残差就好了，当损失函数是指数损失函数时，每步的优化也具有较简单的形式，但是对于一般的损失函数而言（比如加了正则化的损失函数），往往每步优化并不那么容易。因此梯度提升树就被提出来了，它适用于一般的损失函数，它是一种近似算法。

梯度提升树的思路是每一步优化中，用损失函数对模型的负梯度在当前模型上的值 来拟合一个回归树。

为什么用负梯度来拟合当前的基学习器呢？实际上，它是结合了“梯度下降法”。前面已经说过，对于梯度提升树（以及其他Boosting模型），我们贪心地训练它：每一次训练一棵树，不断地减小目标函数值。我们希望用尽量少的树，因此每棵树都最大程度的减小当前目标函数值；但我们又不希望树太复杂，因此每棵树的学习能力有限，不能使目标函数一下子降到最小，所以需要多次训练，逐步优化。对于经验损失，我们的目标很明确，就是降低模型输出与样本标签之间的误差，所以用残差（对于不同的目标函数，残差具有不同形式）拟合就可以；但是对于一般的损失函数，无法在一次训练时简单地确定当前最优目标（注意，整体最优目标我们是清楚的，就是整体损失函数最小化，我们只是无法简单地确定当前这一次训练的目标），因此只能结合梯度下降法考虑。这时，每一棵树相当于更新的参数，用负梯度来拟合当前的树，就相当于沿着梯度下降的方向前进，但并非像纯梯度下降法那样，是“完全地前进”，因为训练一棵树是有误差的。

注意，梯度提升树已经不属于贪心训练了，它更应该被强调为基于梯度下降法的训练。因为它无法保证每一步是当前阶段最优的（梯度下降的方向和梯度的值，不一定对于当前这棵树来说，是能最小化整体目标的方向和步长），但是既然我们已经无法简单地确定每一步的最优目标，那也没有更好的办法了。

**3、XGBoost**

XGBoost是提升树模型的一个扩展。提升树在实际中的应用只能使用均方误差（即MSE）损失函数，而XGBoost不仅支持各种各样的损失函数，还支持对目标函数正则化。

XGBoost不只有这一个优点，它还在应用层面上对传统的提升树算法做了很多改进，比如通过并行加速计算。XGBoost已经被实现为一个应用广泛的系统，并有Python的接口。

下面介绍一下XGBoost的理论推导过程，这部分内容主要参考了陈天奇做的介绍XGBoost理论细节的PPT。注意，他使用的符号与上面介绍提升树的内容中使用的符号的风格不太一致，但是也清晰易懂。

首先，与提升树相同，它属于加性模型：

假设有K个树，则模型输出为：

接下来，定义模型的目标函数为：



其中，第一部分是模型在训练集上的损失函数，第二部分是衡量模型复杂度的正则化项。

损失函数是训练集中样本真实标签与模型输出的函数，它可以定义为任何适合我们所研究的数据的函数，当它为均方误差函数时，它就等价于传统的提升树。

有了目标函数，我们要做的就是优化它。XGBoost的优化与提升树的优化思路相同，即贪心优化，每轮学习一个最优的树，使得将它加到先前学好的模型上以后，更新的模型具有到该轮为止最小的目标函数：



对第t轮优化来说，是已知的常量，故将其作为constant，并在优化问题上忽略之。

对于这个目标函数，我们如何优化呢？考虑到每次增加的新树的输出相对先前训练好的结合模型的输出来说是一个微小的量，这里对目标函数中的训练损失函数进行二阶泰勒展开近似：

设：



（其中gi表示上一轮迭代的损失函数对上一轮迭代后的结合模型的一阶偏导，而hi表示二阶偏导）

则目标函数变为：



对于第t轮优化来说，是已知的常量，故将其合并到constant，并在优化过程中省略掉。

接下来我们定义正则化项的具体形式：



其中T表示该决策树的叶节点的个数，表示该决策树的第j个叶节点的输出值。

然后我们将正则化项的具体形式带入目标函数，并将损失函数部分的求和按照决策树叶节点重新分组，得到：



该目标函数的左边项是关于所有wj的二次函数的求和，当决策树的结构固定了，那么它的T就是固定的，此时只需考虑左边项，也就是优化每个叶节点的输出。

左边项的最小值等于对每一个wj的二次函数求最小，然后再把它们加起来。

设，则目标函数变为：



根据二次函数的性质，使目标函数最小的wj是

，

最小的目标函数值为



注意这是当树结构固定时的情况，因此，每一轮的优化问题就转化为：枚举所有的单棵树结构，计算它的Obj，然后比较每个树的Obj的大小，选择使Obj最小的树结构，计算其，即为所求。

但在实际中，这样做有一个问题，就是很难枚举所有的树结构。所以这里我们仍然考虑使用贪心算法构造最优的树结构，也就是说，对于每一个节点，设定一个目标，根据目标决定其如何分裂。这与一般的（单棵）决策树的训练思路是一致的。那么如何决定目标呢？

根据总体目标，我们自然地想到，可以相应地定义节点分裂带来的增益：



对于这个增益，我们希望它越大越好。其中第一项和第二项分别表示该节点的左子节点和右子节点的效益，第三项表示该节点不分裂的效益，第四项表示该节点分裂后，带来的模型复杂度的增加。

这样，我们需要做的就是遍历每一个特征，并对每个特征扫描分割点，找到使Gain最大的特征和分割点。

以上就是XGBoost的优化问题的推导。注意，整个过程中我们使用了两次贪心优化思想，一次是用每轮训练一棵树代替一次性训练所有树；一次是训练每一棵树时，不是直接按照一个最终的目标求解一整棵树，而是在每次节点分裂时按照一个“阶段目标”求解最优分裂。

总结一下AdaBoost、提升树、梯度提升树（GBM）和XGBoost的主要异同点：

（1）这四个模型都是基于Boosting策略的集成模型；

（2）梯度提升树（GBM）和XGBoost都是基于提升树的改进的模型。二者都使用了近似优化；

（3）AdaBoost不指定具体的基学习器，提升树、梯度提升树和XGBoost指定CART树为基学习器；

（4）AdaBoost指定损失函数为指数损失，提升树支持各种损失函数，但实际应用中只有均方误差（MSE）损失函数和指数损失函数比较容易实现，梯度提升树通过近似优化，支持实际中实现各种损失函数，XGBoost不仅支持实际中实现各种损失函数，还支持实现正则化；

（5）提升树、梯度提升树对单棵决策树的训练过程使用一般的决策树训练方法，而XGBoost对单棵决策树的训练也作出改进，以与其整体目标保持一致。