# 集成算法(Ensembles)

## 摘要：

伴随着人工智能概念的再一次觉醒，机器学习和深度学习的概念也成为了当下研究的热点内容[1]，其中机器学习作为人工智能领域和深度学习领域中一种重要的研究手段，因此学术界和工业界对机器学习中的各种方法进行了大量的研究和应用，并根据机器学习中常见的任务划分出了回归、分类、聚类和降维的问题[2][11]，并相应提出了诸多各式各样的算法，由于算法之间存在应用领域和本身算法适用性问题，因此遇到实际问题时无法立刻找到最优的解决办法，这个问题就是机器学习领域在常见的no free lunch theorem问题[3]。而集成算法(Ensembles)是一种比较独特的机器学习算法，把简单的算法组织起来，这样就能够集合各种算法的优点，来产生一个综合的机器学习策略，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，这个为实际的应用和研究提供了很好的思路[4]。

本文将对集成学习算法中比较经典和流行的Bagging、Boosting和Stacking算法进行介绍并给出具体的算法设计原理和实现流程，最后简要对其性能上给出一定的比较。同时也会简要介绍一些基于这些算法基础上衍生出来的集成算法，例如Gradient Boosting(GB)、Gradient Boosting Decision Tree(GBDT)以及XGBoost算法的设计原理和实现流程。

## 引言：

就机器学习算法来说，其泛化误差可以分解为两部分，偏差(bias)和方差(variance)。偏差指的是算法的期望预测与真实预测之间的偏差程度，体现了模型本身的拟合能力；方差衡量了训练集的变动导致学习性能的变换，刻画了数据扰动对模型性能的影响。在实际的应用过程中会发现，当模型越复杂时，对训练集的拟合效果就越好，模型的训练偏差就越小。如果换一组数据可能模型的变化就会很大，即模型的方差很大，所以得出的经验就是模型过于复杂会导致过拟合问题。当模型比较简单时，即使换测试数据，最后模型得到的结果差别不会很大，也就是说此时模型的方差很小。因此，模型复杂度与模型的偏差和方差存在一个约束关系，对于机器学习的研究有着一定的指导和参考意义。

集成方法是一种常见的机器学习算法，它构造一组分类器，由一组经过单独训练的分类器(如神经网络或决策树)组成，在对新测试用例进行分类时，这些分类器的预测被组合在一起，然后通过对它们的预测进行加权投票来对新的数据点进行分类。以往的研究表明，集成通常比集成中的任何单个分类器更准确。

最初的集成方法是贝叶斯平均法，随着理论研究的推进，又出现了比较流行分类方式包括Bagging，Boosting，Stacking，这三类分别代表了集成算法中并行、串行和树形结合的几种不同方式[4][5]。Bagging是把各个基模型的结果组织起来，取一个折中的结果。Boosting是根据旧模型中的错误来训练新模型，层层改进。Stacking则是把基模型组织起来，基模型本身进行搭配组合，该方法看起来更灵活，也更复杂[6]。

本文将重点回顾Boosting算法和Bagging算法的原理和算法设计思路，给出具体的实现步骤，并解释为什么集合的性能通常优于任何单个分类器，然后分别介绍这两类算法的优点和不足，以及应用当作需要注意的问题。

## 算法介绍

集成学习方法的设计思想是构造一个集合，在集合中的分类器多种多样(多样性)，相互之间独立(独立性)并且都有着较高的准确率(准确性)，通过加法模型将集合中的弱分类器进行线性组合，最后利用平均法、投票法或学习法这三种组合策略来获得高度准确的分类决策。许多作者已经通过集成方法证明了显著的性能改进[6]。

Bagging (Breiman， 1996c)和Boosting (Freund & Shapire， 1996; Shapire， 1990)是两种比较新但却比较流行的集成方法[1]。

Boosting主要关注降低偏差，因此Boosting能基于泛化性能相当弱的学习器构建出很强的集成。Bagging主要关注降低方差，因此它在不剪枝的决策树、神经网络等学习器上效用更为明显[11]。下面将主要对这两种集成方法中的代表算法进行介绍。

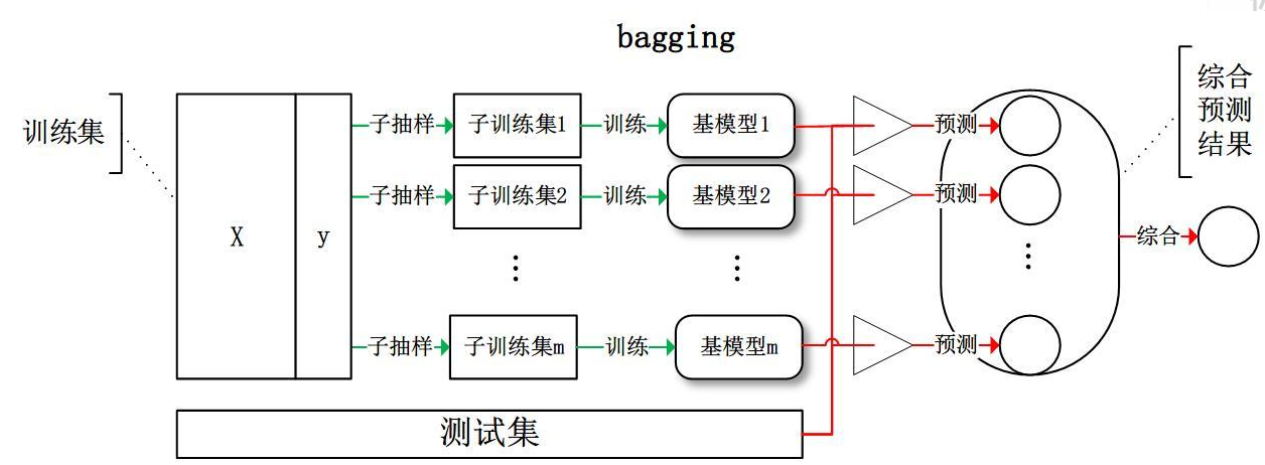
### 3.1 Bagging算法

#### 3.1.1 基本概念

Bagging的全称是bootstrap aggregating，主要借鉴了统计学上的Bootstrap估计方法，Bootstrap是一类非参蒙特卡洛(Monte Carlo)方法[12]，基本原理就是对观测信息进行再抽样，进而对总体的发布特性进行统计推理。Bagging就是按照随机有放回的方式选择训练数据然后构造分类器，最后组合。

下面将以scikit-learn工具包中提供的Bagging集成算法为例来介绍一下Bagging算法需要解决的问题：Bagging Classifier/Regressor是从原始数据集抽选S次(抽取实例，抽取属性)，得到S个新数据集(有的值可能重复，有的值可能不出现)。使用同一模型，训练得到S个分类器，预测时使用投票结果最多的分类。

下面给出的是Bagging算法的结构图。



#### 3.1.2 算法介绍

它是一系列决策树的集成，用随机的方式建立一个决策树的森林。当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行判断，预测时使用投票结果最多的分类，也是少数服从多数的算法。Bagging中各个基算法之间没有依赖，可以并行计算，它的结果参考了各种情况，实现的是在欠拟合和过拟合之间取折中。

Bagging预测器是一种方法，用于生成多个版本的预测器，并使用这些版本获得聚合的预测器。在预测数值结果时，聚合对版本进行平均，在预测类时进行简单投票。通过对学习集进行自举复制并将其作为新的学习集来形成多个版本。利用分类回归树和线性回归中的子集选择对真实数据集和模拟数据集进行了测试，结果表明，套袋法在精度上有较大的提高。关键因素是预测方法的不稳定性。如果对学习集的扰动会导致构建的预测器发生显著变化，那么bagging可以提高预测精度。

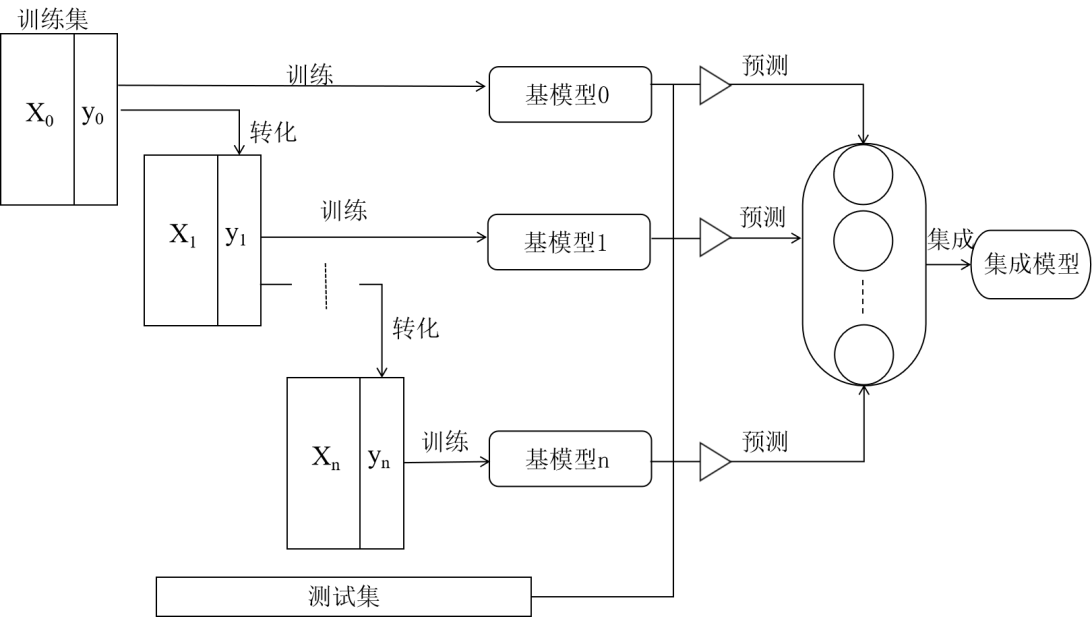
Random Forest (RF)算法，是在以决策树为基学习器构建Bagging集成的基础上，进一步在决策树的训练过程中引入了随机属性选择，属于Bagging算法的一种衍生算法。RF算法在模型训练过程中，采用了将属性集合作为模型训练的一个新增超参数的想法，即从节点属性集合M中随机选取k个属性的子集，然后再从这个子集中选择一个最优属性用于决策树构建。RF算法由于输入数据是随机的从整体的训练样本数据中选取的一部分作为决策树的训练，而且是有放回的选取，此外又引入了随机特征作为构建决策树的方法，这两个随机性的引入使得RF能够有效的避免过拟合现象的出现，而且还具有很好的抗噪声能力。同时RF还具备实现简单，处理高维数据时无需做特征选择等优点，但不足的地方是对噪声比较敏感，对分类问题比较好，但回归问题比较差。

### 3.2 Boosting算法

#### 3.2.1 基本概念

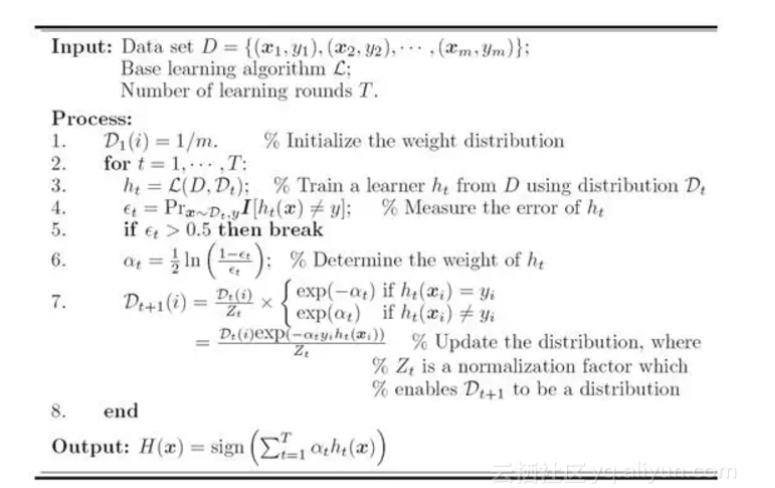
Boosting算法基于错误提升分类器性能的原理，在不断迭代建立新模型的同时，通过调整样本分布特征使新模型更重视上一个模型中被错误分类的样本，提升对数据样本整体的处理精度，迭代执行直到基学习器数目达到指定的值，最后将训练好的基学习器根据按准确率加权组合得到训练结果。

Boosting 算法是一种加法模型(additive training)，由于引入了逐步改进的思想，重要属性会被加权，这也符合人的直觉。一般来说，它的效果会比Bagging好一些。由于新模型是在旧模型的基本上建立的，是一种典型的串行集成算法。由于对错误样本的关注，也可能造成过拟合。下面给出的是Boosting算法的结构图。



#### 3.2.2 算法介绍

Adaptive Boosting(AdaBoost)算法，即自适应增强，是Boosting方法中最具代表性的一种算法，与Bagging算法的不同之处在于AdaBoost模型中子模型必须是串行训练获得，每个新的子模型都是根据已训练出的模型性能来进行训练，每次迭代生成新的子模型使用的训练数据都相同，但是对样本的权重进行了重新设定。AdaBoost会根据当前的错误率，按照增大错误样本权重，减小正确样本权重的原则更新每个样本的权重。它对分类错误属性的给予更大权重，再做下次迭代，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数，则完成对新的弱分类器的训练。不断重复训练和调整权重，直到训练错误率或基学习器的个数满足用户指定的数目为止。Adaboost的最终结果是每个弱学习器加权的结果，如下是AdaBoost算法流程图如下：



其中对基分类器的训练和更新样本分布是AdaBoost算法实现的关键。

在迭代训练基学习器时，使用具有权重分布的训练数据集进行学习，得到基分类器，根据新训练得到的基分类器在训练数据集上计算得到分类误差率：



其中为指示函数，当时值为1，否则等于0。以上公式表面，基分类器在加权的训练数据集上的分类误差率是被误分类样本的权值之和。

权重系数表示基分类器在最终分类器中的重要程度，目的在于使我们得到的基分类器在最终分类器中所占的权值，在确定权重系数时，使用了如下公式：



由公式可以看出，当时，，并且随着的减小而增大，意味着分类误差越小的基分类器在最终的分类器中的作用越大，反之当分类误差越大的基分类器在最后的集成分类器中的作用越小。

当更新训练数据集的权值分布，得到样本的新的权值分布，用于下一轮迭代时，计算新权值的分布如下：



其中是算法引入的一个规范化因子，它的作用在于使成为一个概率分布，公式如下：

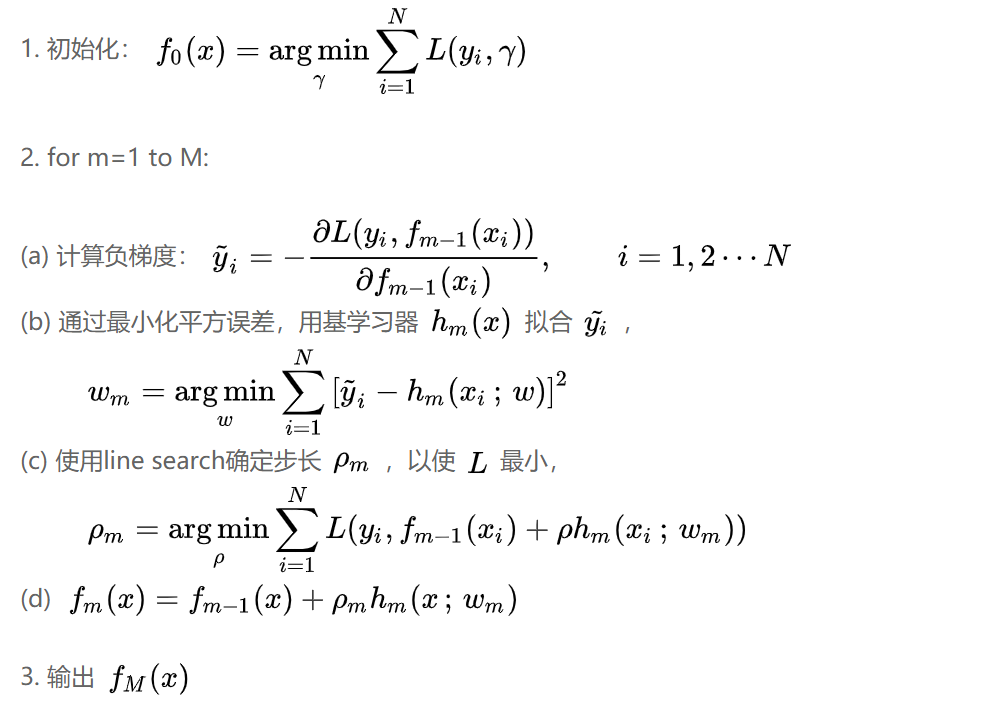


其中表示第t轮迭代时的权值分布，即。由的公式可以看出，被基分类器误分类样本的权值得以扩大，而被正确分类的样本权值得以缩小，因此，在下一轮训练时误分类样本能够起更大的作用，在不改变所给的训练数据情况下，通过不断改变训练数据权值的分布，使得训练数据在基分类器的学习中起到不同的作用，这就是AdaBoost算法的最大特点。

AdaBoost算法的优点在于实现简单，不用担心过拟合问题。常见的实现方式是基模型用单层分类器实现，节点对应当前最适合划分的属性值位置。由于Boosting算法本身都会存在对噪声异常点比较敏感，串行执行速度慢的问题。AdaBoost算法除了这些问题之外，还需要注意的地方就是存在的一个超参数会造成优化过程中陷入局部最优解的情况，不能保证是最优解，因此在具体实现AdaBoost算法的时候需要注意这些问题。

#### 3.2.2.2 GBDT算法

GB算法是根据梯度算法推导得到的，下面是Gradient Boosting算法流程图，主要展示了Gradient过程。



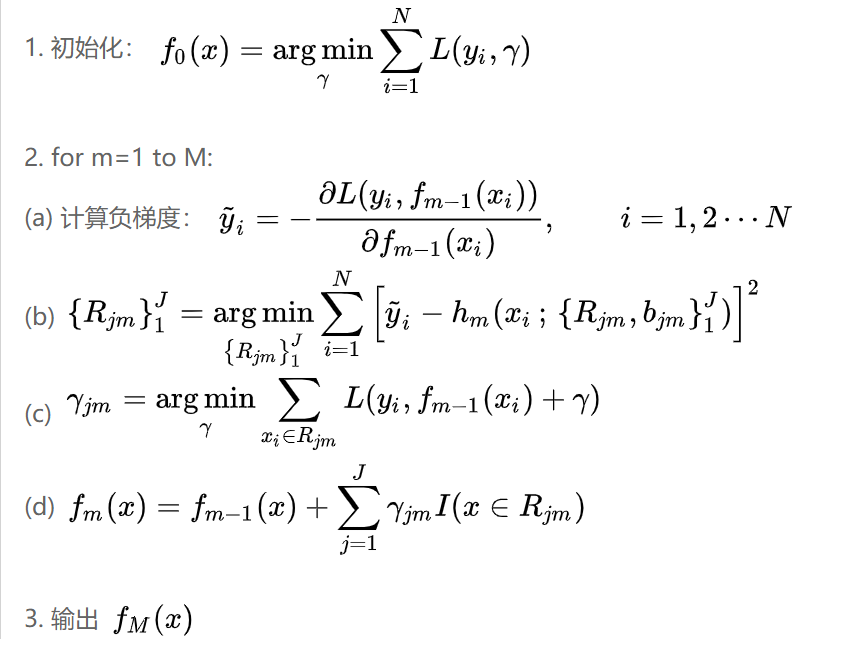
在Gradient Boosting框架中，最常用的基学习器是决策树 (一般是CART)，是因为CART树的叶子节点对应的值是一个实际的分数，而非一个确定的类别，这样有利于实现高效的优化算法，易于实现梯度提升。二者结合就成了著名的梯度提升树 (Gradient Boosting Decision Tree， GBDT)算法。GBDT(Gradient Boost Decision Tree)，是一种基于迭代累加的决策树算法，通过构造一组弱的学习器，并把多棵决策树结果累加作为最终的预测输出，如下所示：



这里的K就是树的棵树，F表示所有可能的CART树。与AdaBoost相比，GBDT每一次的计算都是为了减少上一次的残差，进而在残差减少(负梯度)的方向上建立一个新的模型，进而拟合一棵CART回归树。从数学意义上讲，GBDT是在函数空间中利用梯度下降法进行优化的一种集成算法。每个新的模型的建立是为了使得之前的模型的残差往梯度下降的方法，与传统的Boosting中关注错误样本加权有着很大的区别。

 GBDT的性能在RF的基础上又进一步提升，优点主要来自于它的非线性变换比较多，因此具有较强的表达能力，而且不需要做复杂的特征工程和特征变换。不足的的地方也是来自于Boosting算法固有的几个问题，串行速度慢。

下面是GBDT算法流程图：



Gradient Boosting算法理论上可以选择多种不同的学习算法作为基学习器，但实际应用当作使用最多的无疑是决策树。主要原因是决策树有很多优良的特征，比如能够灵活处理各种类型的数据，包括连续值和离散值，而且对缺失值不敏感。此外，不需要做标准化/归一化，有着很好的解释性。但存在的缺点就是不稳定，容易造成过拟合，模型方差比较大，泛化能力比较低，而且很多时候准确率不如其他算法。然而GBDT在继承了GB和决策树诸多优点的基础上，对其缺点也进行了改进。由于GBDT采用的树都是复杂度低的树，所以方差很小，通过梯度提升的方法集成多个决策树，最终能够很好的解决过拟合的问题。

#### 3.2.2.3 XGBoost算法

Boosting共有的缺点为训练是按顺序的而且难以并行，这样在大规模数据上可能导致速度过慢，所幸近年来XGBoost(eXtreme Gradient Boosting)和LightGBM的出现都极大缓解了这个问题。XGBoost最大的特点在于，它能够自动利用CPU的多线程进行并行，同时在算法上加以改进提高了精度。XGBoost可以说是加强版本的Boosting算法[13]，其中Extreme是极致的意思，主要体现在工程设计层面，包括并发的程序执行，贪心的排序操作等，因此XGBoost算法在各大比赛中展现了强大的威力，下面将对XGBoost算法的原理和推导过程展开详细介绍。

XGBoost 是一个监督模型，对应的模型是一堆CART树，在原理方面的改进主要体现在损失函数上，在原损失函数的基础上添加了正则化项并进行了二阶泰勒公式展开，这类似于对每棵树进行了剪枝并限制了叶结点上的分数来防止过拟合，目标函数模型如下所示：



这个目标函数包含两部分，第一部分就是损失函数，第二部分就是正则项，这里的正则化项由K棵树的正则化项相加而来。下面对XGBoost的训练过程展开具体讨论。

由于XGBoost是一种加法模型，目标不再是直接优化整个目标函数，而是需要分步骤优化目标函数，首先优化第一棵树，然后在优化第二颗，直到优化完K棵树。在添加第t棵CART树时，是在前面已有的t-1棵树的基础上，找出能够使目标函数最小的那棵CART树，其目标函数展开推导如下：



其中：





其中和对目标函数进行二阶泰勒展开的一阶导和二阶导，后面的const是常数项。表示第t棵CART树的某个叶子节点的值，这里将叶子节点的值作为了模型的参数。由于CART树的定义形式比较多，下面给出一种常见形式：



其中T表示的是树的叶子节点数，表示的是由T个叶子节点值组成的一个T维向量，q(x)表示将样本预测到某个叶子节点的映射，代表了CART树的结构。则表示这棵树对样本x的预测值了。有了这个定义，则可以给出XGBoost使用的正则化项定义，形式如下：



其中出现的和λ是XGBoost自己定义的，表示对叶子的惩罚力度，表示叶子结点的权重，后面一项整体表示L2惩罚项。



由于目标函数是为了实现最小化，常数项显然没有什么作用，因此省去常数项。代表一个集合，集合中每一个值代表一个训练样本，整个集合代表被第t棵CART树分到第j个叶子节点上的训练样本。

其中G和H分别可以表示为：

和

当上面的公式和含义确立了之后，由于各个叶子节点的值之间是相互独立的，因此可以通过求导得出各个叶子节点的最佳值以及对应的目标函数值，如下所示：



最优解：



代表了这棵树结构有多好，值越小表示树的结构越好。通过分析推导过程可以看出，这个值仅仅和树的结构有关，能够衡量结构的好坏，与叶子节点的值无关，因此通过上面一系列的推导，我们可以找出结构最佳的树结构，这个定出来之后，就可以用来生成最优的决策树，由于XGBoost在目标函数中考虑了树的复杂度，因此在树的生成过程中不需要进行单独的剪枝操作。

XGBoost与GBDT在boosting策略上类似，区别在于GBDT旨在通过不断加入新的树最快速度降低残差，而XGBoost则可以人为定义损失函数(可以是最小平方差、logistic loss function、hinge loss function或者人为定义的loss function)，只需要知道该loss function对参数的一阶、二阶导数便可以进行boosting，其进一步增大了模型的泛化能力，其贪婪法寻找添加树的结构以及loss function中的损失函数与正则项等一系列策略也使得XGBoost预测更准确。XGBoost在目标函数中引入的正则项，效果体现在对每棵回归树的复杂度进行了惩罚，由于系统的复杂度可以用树的深度，内部节点个数，叶子节点个数，叶节点分数等来衡量，这样做就把树模型复杂度的问题加到了优化目标当中，在项目实测中使用发现，XGBoost的训练速度要远远快于传统的GBDT实现。此外，这样还可以使得XGBoost训练出来的模型不容易产生过拟合。

## 模型的评价

当机器学习模型被训练出来之后往往会对模型进行。

平均法(加权(个体学习器性能相差较大)，简单(性能相近))，投票法(绝对多数(超过半数标记。否则拒绝预测)，相对多数，加权投票)，学习法(通过另一个学习器来进行结合，Stacking算法)

## 结论

首先，虽然Bagging几乎总是比单个分类器更准确，但有时它远不如升压准确。另一方面，提升可以产生比单个分类器更不精确的集合——尤其是在使用神经网络的时候。分析表明[7]，提升方法的性能取决于被测数据集的特性。事实上，进一步的研究结果表明，增加集合体可能会使噪声数据集过大，从而降低集合体的性能。最后，与之前的研究一致，我们的工作表明，集成电路性能的大部分增益来自前几个分类器的组合;然而，在增强决策树时，最多可以看到25个分类器的较大收益。

## 参考文献

1. LeCun Y， Bengio Y， Hinton G. Deep learning[J]. nature， 2015， 521(7553): 436.
2. Goodfellow I， Bengio Y， Courville A， et al. Deep learning[M]. Cambridge: MIT press， 2016.
3. Robert C. Machine learning， a probabilistic perspective[J]. 2014.
4. Dietterich T G. Ensemble methods in machine learning[C]//International workshop on multiple classifier systems. Springer， Berlin， Heidelberg， 2000: 1-15.
5. Dietterichl T G. Ensemble learning[J]. 2002.
6. Dietterich T G. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging， boosting， and randomization[J]. Machine learning， 2000， 40(2): 139-157.
7. Opitz D， Maclin R. Popular ensemble methods: An empirical study[J]. Journal of artificial intelligence research， 1999， 11: 169-198.
8. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection[C]//Ijcai. 1995， 14(2): 1137-1145.
9. Breiman L. Bagging predictors[J]. Machine learning， 1996， 24(2): 123-140.
10. 李航. 统计学习方法[J]. 2012.
11. 周志华. 机器学习[M]. Qing hua da xue chu ban she， 2016.
12. Hall P. The bootstrap and Edgeworth expansion[M]. Springer Science & Business Media， 2013.
13. <https://xgboost.readthedocs.io/en/latest/tutorials/index.html>
14. https://xgboost.readthedocs.io/en/latest/tutorials/model.html
15. <https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf>
16. https://arxiv.org/pdf/1603.02754v1.pdf