**决策树与集成学习**

# 引言

AI时代，机器学习算法成为了研究、应用的热点。当前，最火的两类算法莫过于神经网络算法（CNN、RNN、LSTM等）与树形算法（随机森林、GBDT、Xgboost等），树形算法的基础就是决策树。决策树因其易理解、易构建、速度快的特性，被广泛应用于统计学、数据挖掘、机器学习领域。因此，对决策树的学习，是机器学习之路必不可少的一步。

根据处理数据类型的不同，决策树又分为两类：分类决策树与回归决策树，前者可用于处理离散型数据，后者可用于处理连续型数据

首先从基本情况开始，决策树是一种直观的模型，决策者需要在每个节点进行选择，从而穿过整个“树”。树形归纳是将一组训练样本作为输入，决定哪些从哪些属性分割数据，不断重复这一过程，直到所有训练样本都被归类。在构建树时，我们的目标是用数据分割创建最纯粹的子节点。纯粹性是通过信息增益的概念来衡量的。在实际中，这是通过比较熵或区分当前数据集中的单一样本和所需信息量与当前数据需要进一步区分所需要的信息量。

随机森林是决策树的简单集成，即是输入向量经过多个决策树的过程。对于回归，所有树的输出值是平均的；对于分类，最终要用投票策略决定。

二元决策树是基于属性做一系列二元(是/否)决策。每次决策从下面的两种决策中选择一种，然后又会引出另外两种决策，依次类推直到叶子节点:即最终的结果。也可以理解为是对二叉树的遍历，或者很多层的If-Else嵌套。

# 决策树

决策树是一种基本的**分类和回归**算法

组成：结点、有向边

结点：内部结点：一个特征或属性

叶子结点：一个类

目标函数：正则化的极大似然函数

策略：损失函数最小化

剪枝的目的：使决策树具有更好的泛化能力

决策树的生成：局部最优

决策树的剪枝：全局最优

**决策树学习通常包括三个步骤：特征选择，决策树生成，决策树剪枝。**

## 特征选择

选择对训练数据具有分类能力的特征

### 信息增益

#### 熵

熵 H(X)：随机变量X不确定性的度量



熵越大，随机变量的不确定性越大，并且满足



#### 条件熵

条件熵 H(Y|X)：已知随机变量X的条件下度量随机变量Y的不确定性；

给定随机变量X的条件下，随机变量Y的条件概率分布的熵对X的数学期望。



#### 经验熵/经验条件熵

经验熵/经验条件熵：熵/条件熵中的概率由数据估计（尤其是极大似然估计）得到

#### 信息增益

信息增益：已知特征X的信息使得类Y不确定性减少的程度

给定训练集D的经验熵与已知特征A的信息下D的经验条件熵：g(D,A) = H(D) - H(D|A) 数据集D的不确定性 - 已知特征A的数据集D的不确定性



对训练数据集D，计算每个特征的信息增益，信息增益越大，即在当前特征下类Y的不确定性减少的程度越大，即当前特征具有更强的分类能力（取信息增益最大的特征）

### 信息增益比

### 基尼系数



## 决策树的生成

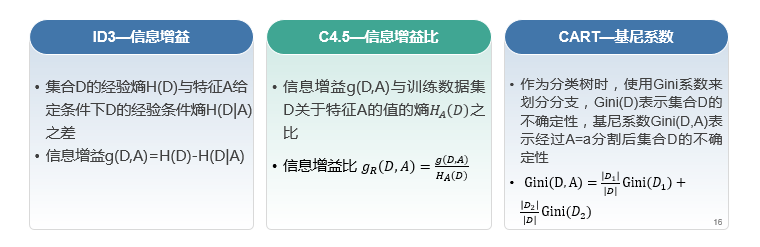
### ID3

### C4.5

### CART

## 决策树的剪枝

三种比较常见的分类决策树分支划分方式包括：ID3, C4.5, CART。



# 回归决策树（Regression Decision Tree）

分类与回归树（Classification And Regression Tree，CART)模型由Breiman等人在1984年提出，是应用广泛的决策树学习方法。CART同样由特征选择、树的生成及剪枝组成，既可以用于分类也可以用于回归。

**CART假设决策树是二叉树**，内部结点特征的取值只有“是”和“否”，左分支是取值为“是”的分支，有分支则相反。这样的决策树等价于递归地二分每个特征。

所谓回归，就是根据特征向量来决定对应的输出值。回归树就是将特征空间划分成若干单元，每一个划分单元有一个特定的输出。因为每个结点都是“是”和“否”的判断，所以划分的边界是平行于坐标轴的。对于测试数据，我们只要按照特征将其归到某个单元，便得到对应的输出值。

划分的过程也就是建立树的过程，每划分一次，随即确定划分单元对应的输出，也就多了一个结点。当根据停止条件划分终止的时候，最终每个单元的输出也就确定了，也就是叶结点。

决策树的生成就是递归地构建二叉决策树的过程，对回归树用平方误差最小化准则，对分类树用基尼指数最小化准则，进行特征选择，生成二叉树。

既然要划分，切分点怎么找？输出值又怎么确定？这两个问题也就是回归决策树的核心。

切分点选择：最小二乘法

输出值：单元内均值

切割点的选择是二元决策树最核心的部分，其基本思路是：遍历所有数据，尝试每个数据作为分割点，并计算此时左右两侧的数据的离差平方和，并从中找到最小值，然后找到离差平方和最小时对应的数据，它就是最佳分割点。

## 原理

假设X和Y分别为输入和输出变量，并且Y是连续变量，给定训练数据集为



其中为输入实例(特征向量)，为特征个数，，为样本容量。

对特征空间的划分采用启发式方法，每次划分逐一考察当前集合中所有特征的所有取值，根据平方误差最小化准则选择其中最优的一个作为切分点。如对训练集中第个特征变量和它的取值，作为切分变量和切分点，并定义两个区域和，为找出最优的和，对下式求解



也就是找出使要划分的两个区域平方误差和最小的特征变量和它的取值。其中，和为划分后两个区域内固定的输出值，方括号内的两个Min意为使用的是最优的和，也就是使各自区域内平方误差最小的和，易知这两个最优的输出值就是各自对应区域内Y的均值（这个可以证明），所以上式可写为



其中，



找到最优的切分点后，依次将输入空间划分为两个区域，接着对每个区域重复上述划分过程，直到满足停止条件为止。这样就生成了一棵回归树，这样的回归树通常称为最小二乘回归树。

## 算法

输入：训练数据集；

输出：回归树.

在训练数据集所在的输入空间中，递归地将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

1. 选择最优切分变量与切分点，求解



遍历变量，对固定的切分变量扫描切分点，选择使上式达到最小值的对。

1. 用选定的对划分区域并决定相应的输出值：



(3) 继续对两个子区域调用步骤(1),(2)，直至满足停止条件.

(4) 将输入空间划分为M个区域，生成决策树：



其中I为指示函数。



# 提升决策树（Boosting Decision Tree）

## 原理

提升树是迭代多棵回归树来共同决策。当采用平方误差损失函数时，每一棵回归树学习的是之前所有树的结论和残差，拟合得到一个当前的残差回归树，残差的意义如公式：残差 = 真实值 - 预测值 。提升树即是整个迭代过程生成的回归树的累加。

## 算法

1. 初始化；
2. 对
3. 计算残差：
4. 拟合残差，学习一个回归树，得到
5. 更新
6. 得到提升决策树



# 梯度提升决策树（Gradient Boosting Decision Tree）

## 原理

提升树是利用加法模型和前向分步算法实现学习的优化过程。当损失函数是平方损失和指数损失函数时，每一步的优化很简单，如当损失函数时平方损失函数时，学习到的决策树即为残差回归树。

但对于一般的损失函数，往往每一步优化没那么容易，如绝对值损失函数和Huber损失函数。针对这一问题，Freidman提出了梯度提升算法：利用最速下降的近似方法，即利用损失函数的负梯度在当前模型的值，作为回归问题中提升树算法的残差的近似值，拟合一个回归树。

GBDT（Gradient Boosting Decision Tree）由多棵回归树组合而成，属于集成学习（Ensemble Learning）的一种。集成学习又主要包括两种方法Bagging和Boosting（提升）。Boosting方法先训练出一个基学习器，然后根据其表现，进行调整，得到下一个基学习器，最终将T个基学习器加权结合。

首先GBDT是通过采用加法模型（即基函数的线性组合），以及不断减小训练过程产生的残差来达到将数据分类或者回归的算法。

GBDT通过多轮迭代，每轮迭代产生一个弱分类器，每个分类器在上一轮分类器的残差基础上进行训练。对弱分类器的要求一般是足够简单，并且是低方差和高偏差的。(偏差：描述的是预测值（估计值）的期望与真实值之间的差距。方差：描述的是预测值的变化范围，离散程度，也就是离其期望值的距离。)因为训练的过程是通过降低偏差来不断提高最终分类器的精度。

加法模型



损失函数



1. 初始化；
2. 对
3. 计算：



1. 计算：



1. 得到最终模型



梯度提升

前面介绍的加法模型的最重要的地方就是求解

对于一个样本来说，其损失函数的泰勒展开为：



因为，所以，如果令



则



即，增加一个新的模型时，其损失函数能够保证是不增的。

梯度提升决策树

## 算法

1. 初始化



即估计一个使损失函数极小化的常数值，它是一个只有一个根节点的的树

1. 对
2. 计算负梯度：



1. 拟合残差，学习一个回归树，得到第棵树的叶子节点区域，
2. 对，计算



1. 更新
2. 得到提升决策树



# 随机森林（Random Forest）

随机：数据采样随机+特征选择随机

森林：并行训练多个决策树

## 算法

设总的样本数为，每个样本的特征数为。

1. 从个总样本中以有放回抽样的方式，取样次，形成个随机训练集；
2. ，对第个随机样本集，随机选出个输入特征，训练一个决策树；
3. 将这个

## 随机森林优点

* 能够处理具有多维特征的数据，并且不用做特征选择；
* 并行训练多棵决策树，运算速度快；
* 便于可视化展示与分析

# 集成学习（Ensemble Learning）

## 原理

在机器学习中，已经有了许许多多的模型，比如SVM，逻辑回归等等，这些算法有各自的优缺点，并且每种算法也可以产生非常多的不同的模型。如果把这些算法都结合起来，取其所长，或者说把一些比较弱的分类器结合起来形成一个强分类器（Boosting），可以得到更优的模型。把一堆算法融合起来这个方法叫Blending；根据融合方式不同，如果是线性融合则称为Linear Blending, 如果在一堆算法的输出结果上再做一个其他的算法融合，这叫做Stacking。

集成学习(Ensemble Learning) 本身不是一个单独的机器学习算法，而是通过构建并结合多个机器学习器来完成学习任务。集成学习可以用于分类问题集成，回归问题集成，特征选取集成，异常点检测集成等等。其基本思想是对于同一个训练集数据，通过训练若干个基本学习器，通过一定的结合策略，最终形成一个强学习器，以达到博采众长的目的。

集成学习有两个主要的问题需要解决，第一是如何得到若干个基本学习器，第二是如何选择一种结合策略，将这些基本学习器集合成一个强学习器。

对于第一个问题，一般有两种选择。第一种是所有的基本学习器都是一个种类的，或者说是同质的。比如都是决策树基本学习器，或者都是神经网络基本学习器。第二种是所有的基本学习器不全是一个种类的，或者说是异质的。比如我们有一个分类问题，对训练集采用支持向量机基本学习器，逻辑回归基本学习器和朴素贝叶斯基本学习器来学习，再通过某种结合策略来确定最终的分类强学习器。

目前来说，**同质基本学习器**的应用是最广泛的，一般我们常说的集成学习的方法都是指的同质基本学习器。而**同质基本学习器使用最多的模型是CART决策树和神经网络**。同质基本学习器按照基本学习器之间是否存在依赖关系可以分为两类，第一个是基本学习器之间不存在强依赖关系，一系列基本学习器可以并行生成，代表算法是**Bagging和随机森林（Random Forest）**系列算法；第二个是基本学习器之间存在强依赖关系，一系列基本学习器都需要串行生成，代表算法是**Boosting系列算法**。

集成学习算法大致上可以分成三类：

* Bagging算法：训练多个基本分类器，然后取平均。



* Boosting算法：从弱学习器开始加强，通过加权来进行训练



* Stacking算法：聚合多个分类或者回归模型，一般是分阶段来做，前一阶段的输出结果作为下一阶段的输入。

集成学习算法

## 算法

### Bagging算法

Bagging算法的全称是**Bootstrap Aggregation**，引导聚集算法，又称装袋算法，并行训练一堆学习器。

Bagging算法的基本原理：从数据集中有放回的随机取样，每次抽取一定比例的数据作为训练样本，剩下的数据作为测试样本。重复以上动作次，然后对每次训练一个基本学习器，最后把这些基本学习器组合到一起，成为集合算法。最具代表性的算法是**随机森林**。



对于随机采样一般采用的是自助采样法（Bootstap Sampling），即对于包含个样本的原始训练集，每次先随机采集一个样本放入采样集，接着把该样本放回，也就是说下次采样时该样本仍有可能被采集到，这样采集次，最终可以得到个样本的采样集，由于是随机采样，这样每次的采样集是和原始训练集不同的，和其他采样集也是不同的，这样得到多个不同的弱学习器。

对于无放回的采样的算法称为Pasting算法。

随机森林是Bagging的一个特化进阶版，所谓的特化是因为随机森林的弱学习器都是**决策树**。所谓的进阶是随机森林在Bagging的样本**随机采样**基础上，又加上了**特征的随机选择**，其基本思想没有脱离Bagging的范畴。

### Pasting算法

不放回取样

### Boosting算法

Boosting算法是将“弱学习算法“提升为“强学习算法”的过程，主要思想是“三个臭皮匠顶个诸葛亮”。 其主要思想是将弱分类器组装成一个强分类器。在PAC（probably approximately correct，概率近似正确）学习框架下，则一定可以将弱分类器组装成一个强分类器。

Boosting算法的工作机制是首先从训练集用初始权重训练出一个弱学习器1，根据弱学习器的学习误差率表现来更新训练样本的权重，使得之前弱学习器1学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器2中得到更多的重视。然后基于调整权重后的训练集来训练弱学习器2。如此重复进行，直到弱学习器数达到事先指定的数目T，最终将这T个弱学习器通过集合策略进行整合，得到最终的强学习器。

训练基本学习器的训练数据不再关注抽样的数据，而是关注由前一次训练之后产生的数据偏差。也就是说下一个模型接收的训练数据是上一个模型训练之后与目标值做对比之后产生的偏差数据。



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：



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。

典型的代表算法是**Adaboost（Adaptive Boosting），提升决策树，梯度提升决策树，XGboost**

#### Adaboost算法

**Adaboost，全称为Adaptive Boosting，意为自适应增强。**AdaBoost通过改变训练数据的权值，也就是样本的概率分布，将关注点放在被错误分类的样本上，减小上一轮被正确分类的样本权值，提高那些被错误分类的样本权值。然后，再根据所采用的一些基本机器学习算法进行学习。

AdaBoost采用加权多数表决的方法，加大分类误差率小的弱分类器的权重，减小分类误差率大的弱分类器的权重。

算法：

1. 初始化训练样本的权值分布：





1. 对于
2. 使用具有权值分布的训练数据集进行学习，得到弱分类器；
3. 计算在训练数据集上的分类误差率：



1. 计算在强分类器中所占的权重：



注意到，错误率越小则分类器的权重越大，并且当时，即错误率大于0.5，该分类器的权重为负数

1. 更新训练数据集的权值分布：





1. 构建基本分类器的线性组合



1. 得到最终的分类器



公式的推导：

假设经过了轮迭代，得到的模型为，由前向分布



AdaBoost是采用指数损失，由此可以得到损失函数：



因为是已知的，所以前面一项是常数，即



其中



这就是每轮迭代的样本权重。

继续简化损失函数



因为



所以



为了求出损失函数的极小值，对求偏导



得到

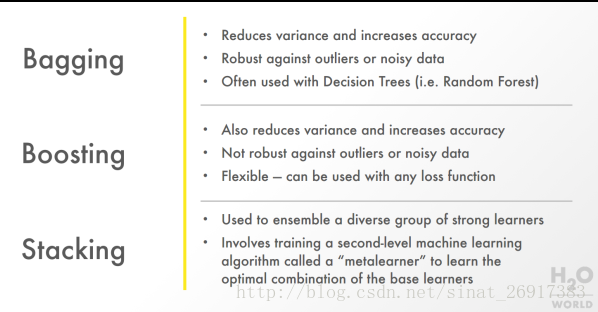


#### 提升决策树

#### [梯度提升决策树](#_梯度提升决策树（Gradient_Boosting_Decision)

### Stacking算法

将多种学习器堆叠在一起



## Bagging，Boosting二者之间的区别

### Bagging和Boosting的区别：

　　1）样本选择上：

　　Bagging：训练集是在原始集中有放回选取的，从原始集中选出的各轮训练集之间是独立的。

　　Boosting：每一轮的训练集不变，只是训练集中每个样例在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。

　　2）样例权重：

　　Bagging：使用均匀取样，每个样例的权重相等

　　Boosting：根据错误率不断调整样例的权值，错误率越大则权重越大。

　　3）预测函数：

　　Bagging：所有预测函数的权重相等。

　　Boosting：每个弱分类器都有相应的权重，对于分类误差小的分类器会有更大的权重。

　　4）并行计算：

　　Bagging：各个预测函数可以并行生成

　　Boosting：各个预测函数只能顺序生成，因为后一个模型参数需要前一轮模型的结果。

### 决策树与这些算法框架进行结合所得到的新的算法：

* 1. Bagging + 决策树 = 随机森林
  2. AdaBoost + 决策树 = 提升树
  3. Gradient Boosting + 决策树 = GBDT

## 差异化模型（Diverse Model）

差异化的模型可以用下面几种方法生成：

* 在相同的数据集上，使用不同的算法
* 在相同的数据集上，使用相同的算法，但是使用不同的参数
* 在不同的数据集上，使用相同的算法
* 对于初值敏感的算法，可以使用不同初值得到不同的模型

## 结合策略

假定得到的个弱学习器是

### 平均法

对于数值类的回归预测问题，通常使用的结合策略是平均法，也就是说，对于若干个弱学习器的输出进行平均得到最终的预测输出。

最简单的平均是算术平均，也就是说最终预测是



如果每个个体学习器有一个权重，则最终预测是



其中是个体学习器的权重，通常有



### 投票法

对于分类问题的预测，通常使用的是投票法。最简单的投票法是相对多数投票法，也就是我们常说的少数服从多数，也就是个弱学习器的对样本的预测结果中，数量最多的类别为最终的分类类别。如果不止一个类别获得最高票，则随机选择一个做最终类别。

稍微复杂的投票法是绝对多数投票法，也就是我们常说的要票过半数。在相对多数投票法的基础上，不光要求获得最高票，还要求票过半数。否则会拒绝预测。

更加复杂的是加权投票法，和加权平均法一样，每个弱学习器的分类票数要乘以一个权重，最终将各个类别的加权票数求和，最大的值对应的类别为最终类别。

### 学习法

上面两种方法都是对弱学习器的结果做平均或者投票，相对比较简单，但是可能学习误差较大，于是就有了学习法这种方法，对于学习法，代表方法是Stacking，当使用Stacking的结合策略时， 我们不是对弱学习器的结果做简单的逻辑处理，而是再加上一层学习器，也就是说，我们将训练集弱学习器的学习结果作为输入，将训练集的输出作为输出，重新训练一个学习器来得到最终结果。

在这种情况下，我们将弱学习器称为初级学习器，将用于结合的学习器称为次级学习器。对于测试集，我们首先用初级学习器预测一次，得到次级学习器的输入样本，再用次级学习器预测一次，得到最终的预测结果。

# 二元决策树的过度拟合

二元决策树同普通最小二乘法一样，都存在拟合过度的情况，判断是否拟合过度有两种方法：

1. 观察结果图。
2. 比较决策树终止节点的数目与数据的规模。设生成决策树的最大深度是，即最多会有个叶子节点，而数据集中一共才有个数据，也就是说有很多节点是只包括一个数据的。

# 二元决策树深度的选择

一般是通过不同深度二元决策树的交叉验证(前面已讲过原理)来确定最佳深度的，基本思路：

1）确定深度列表：

2）设置采用几折交叉验证

3）计算每折交叉验证时的样本外数据的均方误差

4）绘图，观察结果

# 优缺点

## 优点：

对复杂、高度非线性的关系非常实用。它们通常能达到非常高的表现性能，比多项式回归更好。

易于使用理解。虽然最后的训练模型会学会很多复杂的关系，但是训练过程中的决策边界易于理解。

## 缺点：

由于训练决策树的本质，它们更易于过度拟合。一个完整的决策树模型会非常复杂，并包含很多不必要的结构。虽然有时通过“修剪”和与更大的随机森林结合可以减轻这一状况。

利用更大的随机森林，可以达到更好地效果，但同时会拖慢速度，需要更多内存。