《数据挖掘技术》

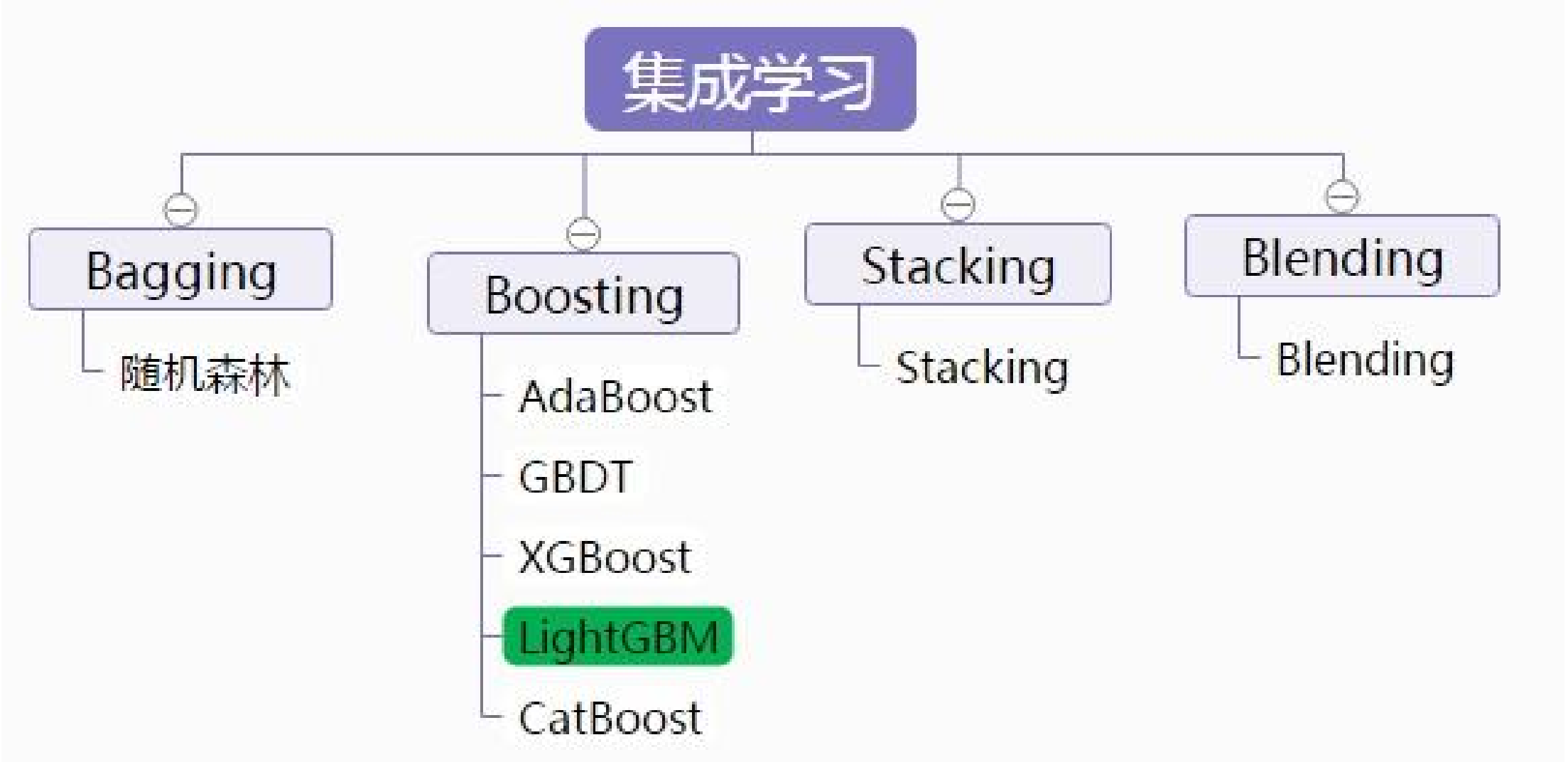
# ⭐ CH08 提升方法

➡️ Created by *Wang JingHui*

➡️ Version: *4.0*

## 主要内容

1. 提升方法AdaBoost算法
2. AdaBoost算法的训练误差分析
3. AdaBoost算法的解释
4. 提升树


集成学习
Bagging
随机森林
Boosting
AdaBoost
GBDT
XGBoost
LightGBM
CatBoost
Stacking
Stacking
Blending
Blending

# 常用的集成算法概述

集成算法一般分为：Bagging，Boosting，Stacking（我们可以把它简单地看成并行，串行和树型），Blending。

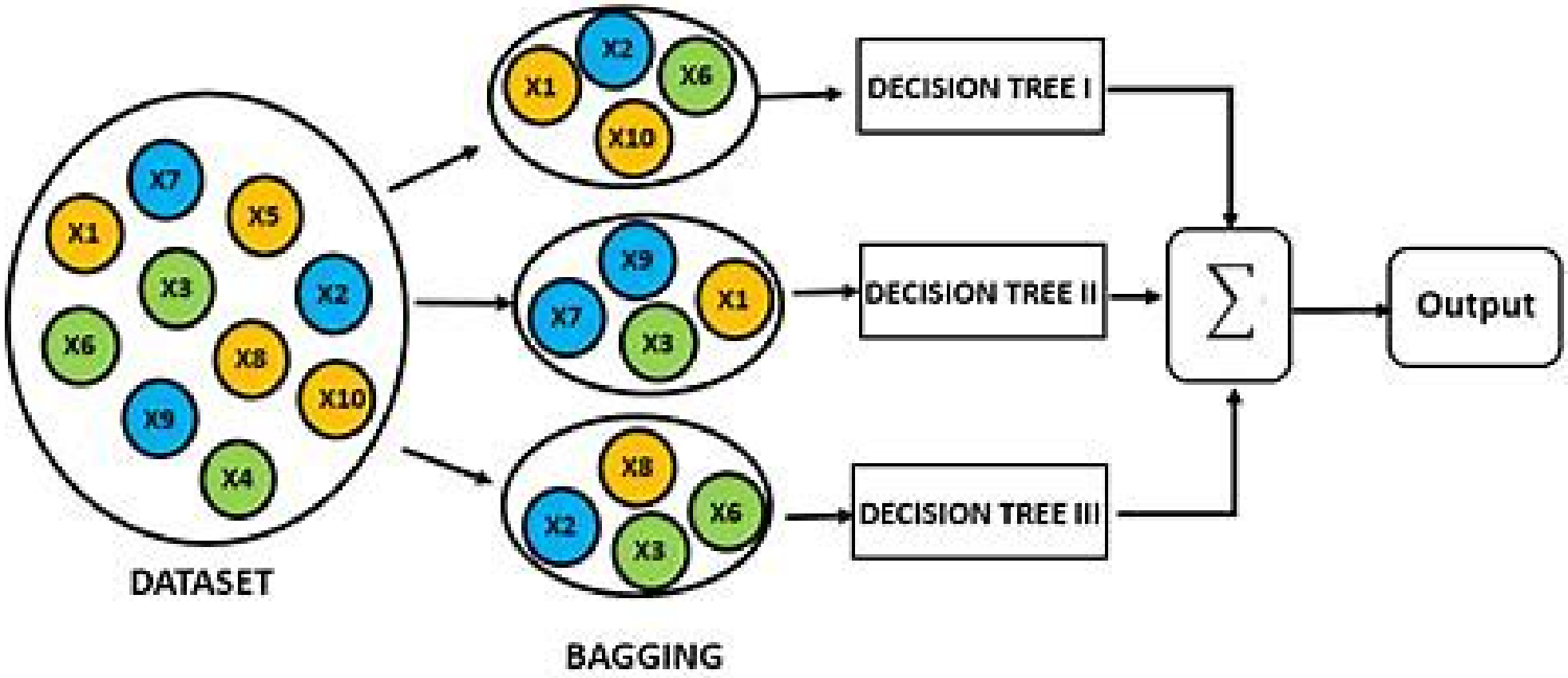
- Bagging是把各个基模型的结果组织起来，取一个折中的结果；
- Boosting是根据旧模型中的错误来训练新模型，层层改进；
- Stacking是把基模型组织起来，注意不是组织结果，而是组织基模型本身，该方法看起来更灵活，也更复杂。

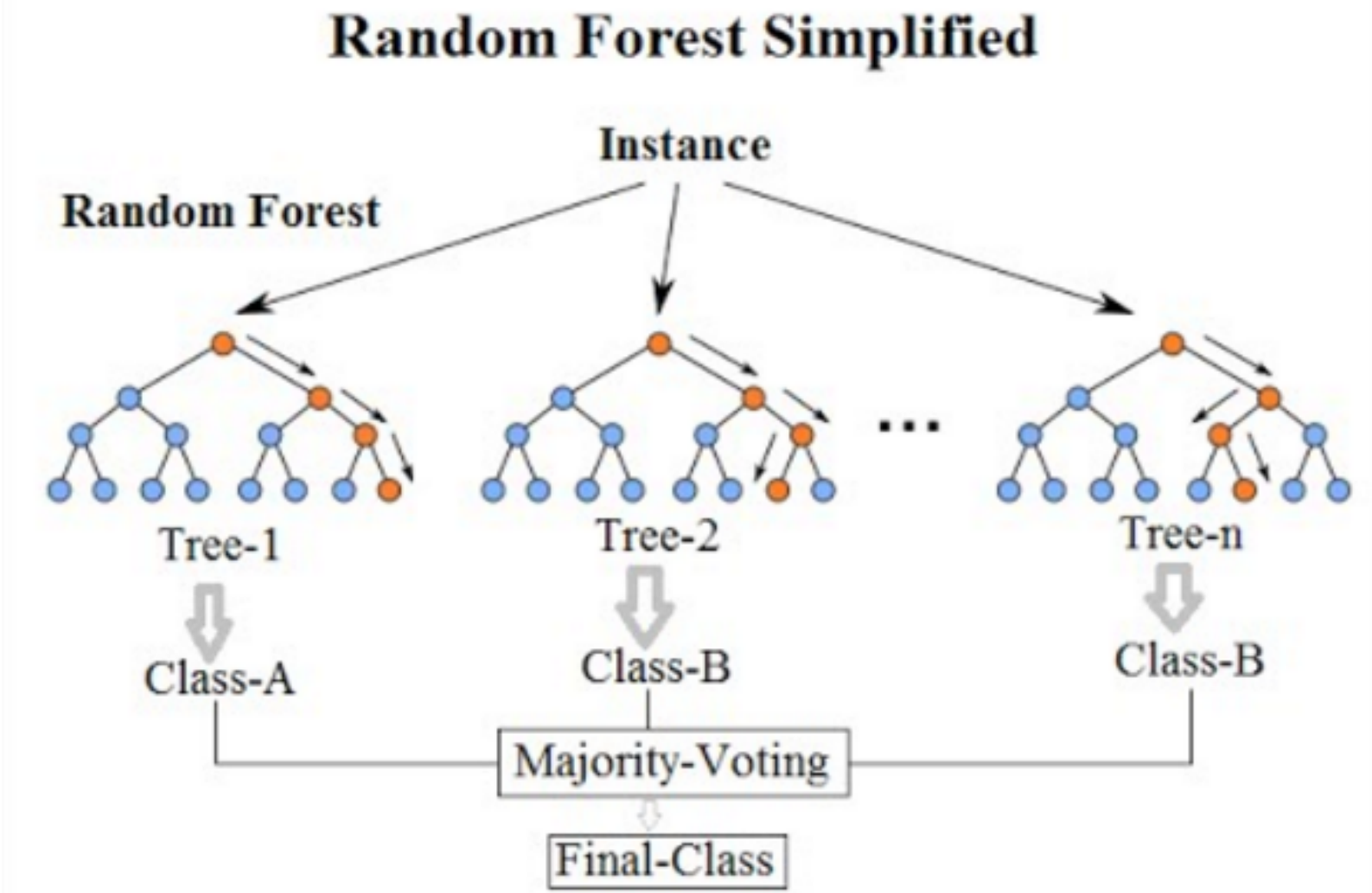
## 1 . Bagging（自举汇聚法）

Bagging的全称是bootstrap averaging，它把各个基模型的结果组织起来，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，具体实现也有很多种类型，以sklearn中提供的Bagging集成算法为例：

- BaggingClassifier/BaggingRegressor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。
- RandomForestClassifier随机森林，对决策树的集成，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。
- VotingClassifier，选择多个不同的基模型，分别进行预测，以投票方式决定结果。

图示：


X7
X1
X5
X3
X2
X6
X8
X10
X9
X4
DATASET
X2
X1
X6
X10
X9
X7
X1
X3
X8
X2
X6
X3
BAGGING
DECISION TREE I
DECISION TREE II
DECISION TREE III
Σ
Output


Random Forest Simplified
Instance
Random Forest
Tree-1
Tree-2
Tree-n
Class-A
Class-B
Class-B
Majority-Voting
Final-Class

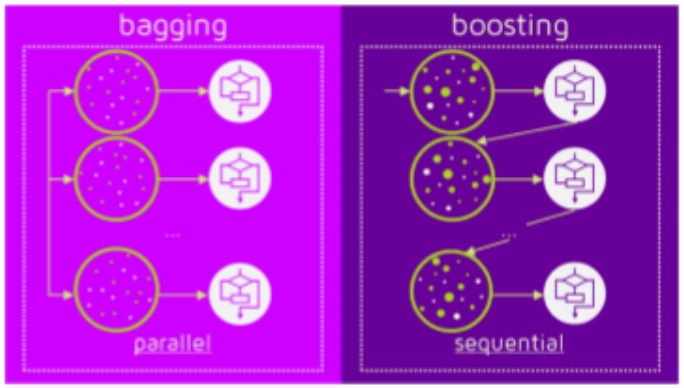
## 2 . Boosting（提升法）

Boosting不断的建立新模型，而新模型更重视上一个模型中被错误分类的样本，最终根据按成功度加权组合得到结果。

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，因此不能使用并行方法训练，并且由于对错误样本的关注，也可能造成过拟合。常见的Boosting算法有：

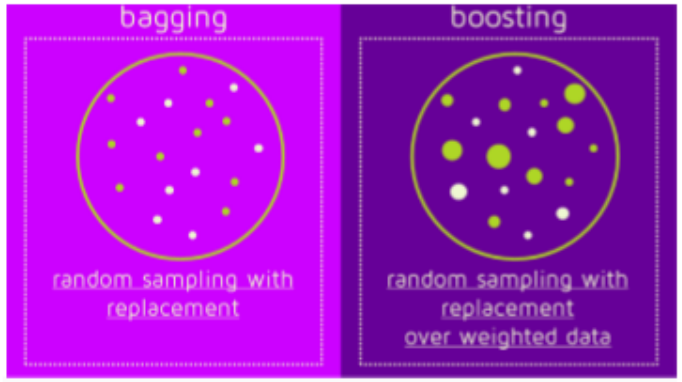
- AdaBoost自适应提升算法，它对分类错误属性的给予更大权重，再做下次迭代，直到收敛。AdaBoost是一个相对简单的Boosting算法，可以自己写代码实现，常见的做法是基模型用单层分类器实现（树桩），桩对应当前最适合划分的属性值位置。
- Gradient Boosting Machine（简称GBM）梯度提升算法，它通过求损失函数在梯度方向下降的方法，层层改进，sklearn中也实现了该算法：GradientBoostingClassifier/GradientBoostingRegressor。

## The key differences between Bagging and Boosting



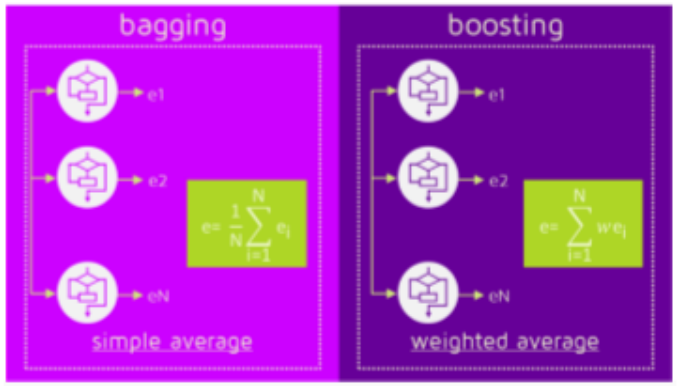

- **Boosting**
  Sequential ensemble methods where the base learners are generated sequentially (e.g. AdaBoost). Each learner is built on top of the previous one.

- **Bagging**
  Parallel ensemble methods where the base learners are generated in parallel.




- **Bagging**
  N new training data sets are produced by random sampling with replacement from the original set. By sampling with replacement some observations may be repeated in each new training data set.

- **Boosting**
  incorrectly predicted observations are weighted and therefore some of them will take part in the new sets more often




- **Bagging**
  The result is obtained by averaging the responses of the N learners (or majority vote).

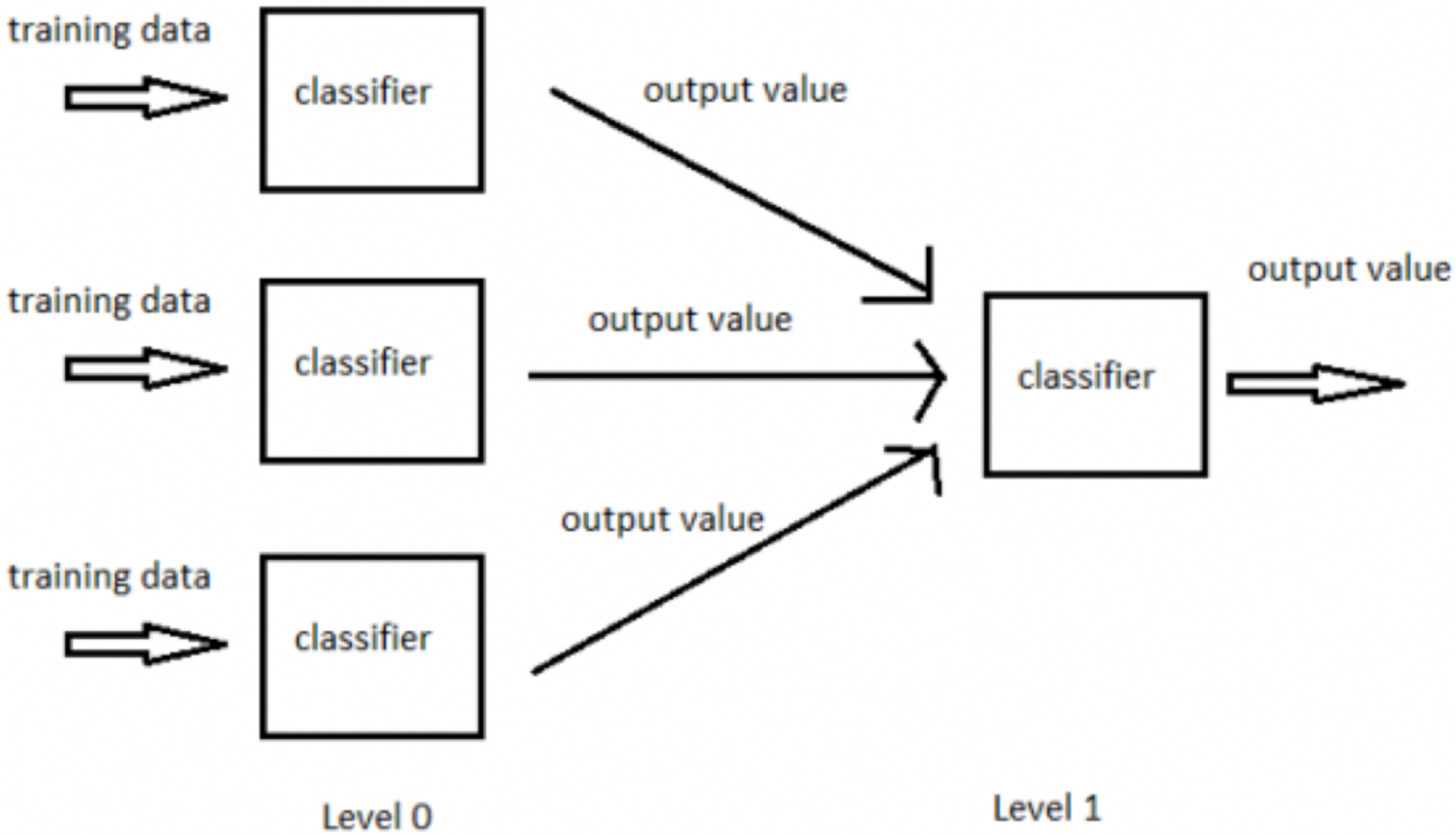
- **Boosting**
  The final boosting ensemble uses a weighted average, more weight to those with better performance on training data.

## 3 . Stacking

Stacking训练一个模型用于组合(combine)其他各个基模型。

- 具体方法是把数据分成两部分，用其中一部分训练几个基模型A1,A2,A3，用另一部分数据测试这几个基模型，把A1,A2,A3的输出作为输入，训练组合模型B。
- 注意，它不是把模型的结果组织起来，而把模型组织起来。理论上，Stacking可以组织任何模型，实际中常使用单层logistic回归作为模型。
- Sklearn中实现了stacking模型：StackingClassifier

# Concept Diagram of Stacking



# 提升方法AdaBoost算法

## 提升方法的基本思路

概率近似正确(PAC, Probably approximately correct)

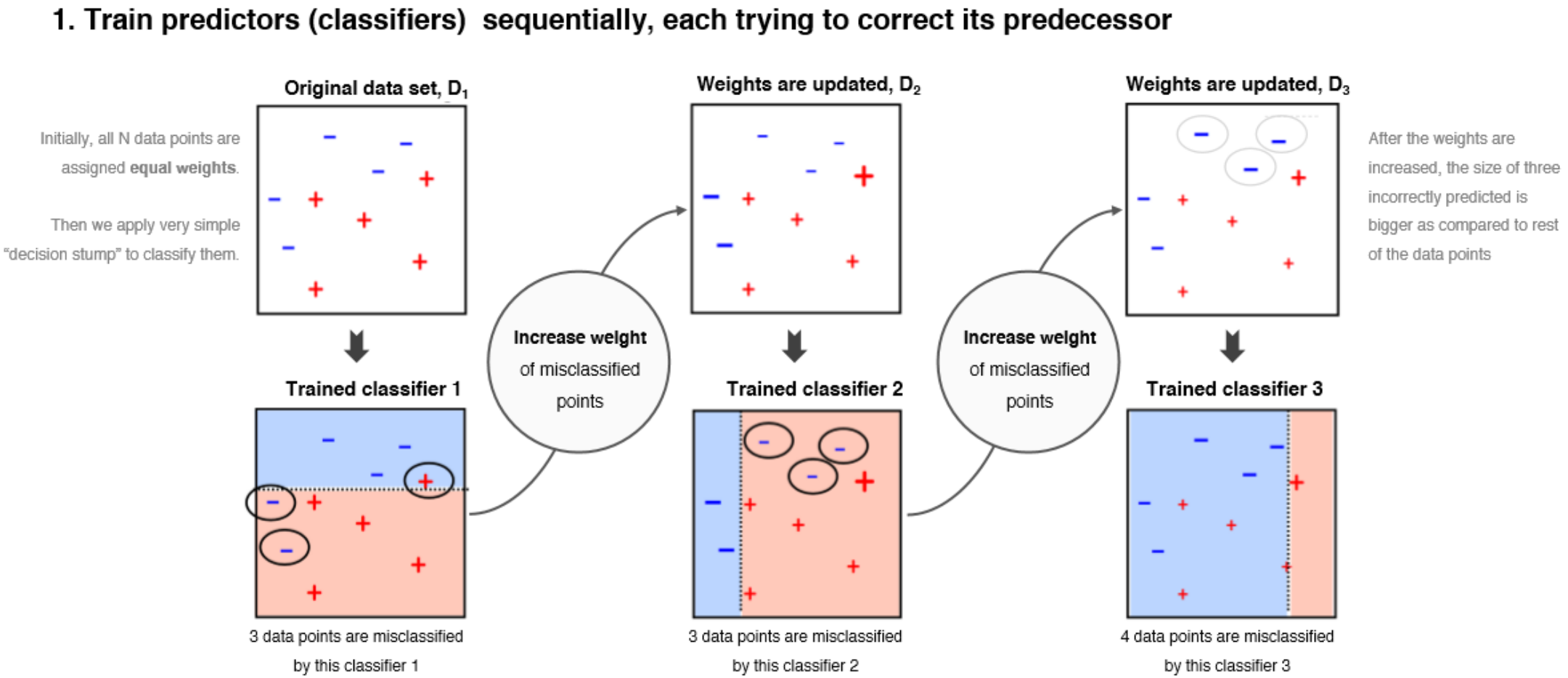
在PAC学习框架下，一个概念是强可学习的充分必要条件是这个概念是弱可学习的。
两个问题

1. 在每一轮如何改变训练数据的权值或者概率分布
2. 如何将弱分类器组合成一个强分类器

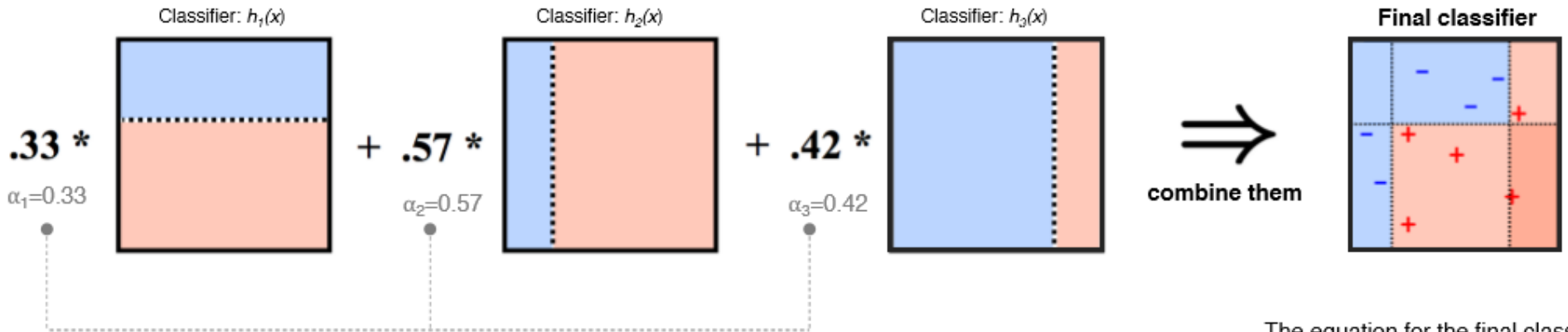
Adaboost解决方案：

1. 提高前一轮被分错的分类样本的权值，降低被正确分类的样本的权值
2. 加权多数表决的方法

# Adaboost算法


1. Train predictors (classifiers) sequentially, each trying to correct its predecessor
Original data set, $D_1$
Initially, all N data points are assigned **equal weights**.
Then we apply very simple "decision stump" to classify them.
Trained classifier 1
3 data points are misclassified by this classifier 1
Increase weight of misclassified points
Weights are updated, $D_2$
Trained classifier 2
3 data points are misclassified by this classifier 2
Increase weight of misclassified points
Weights are updated, $D_3$
Trained classifier 3
4 data points are misclassified by this classifier 3
After the weights are increased, the size of three incorrectly predicted is bigger as compared to rest of the data points

## 2. Combines the several weak classifiers into one strong learner




After it's trained, we compute the output weight (alpha) for that classifier. After each classifier is trained, the classifier's weight is calculated based on its accuracy. More accurate classifiers are given more weight[1]. see next slide for more details.

$$\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$\epsilon_t$ : It's based on the classifier's error rate. It is the weighted sum error for misclassified points.

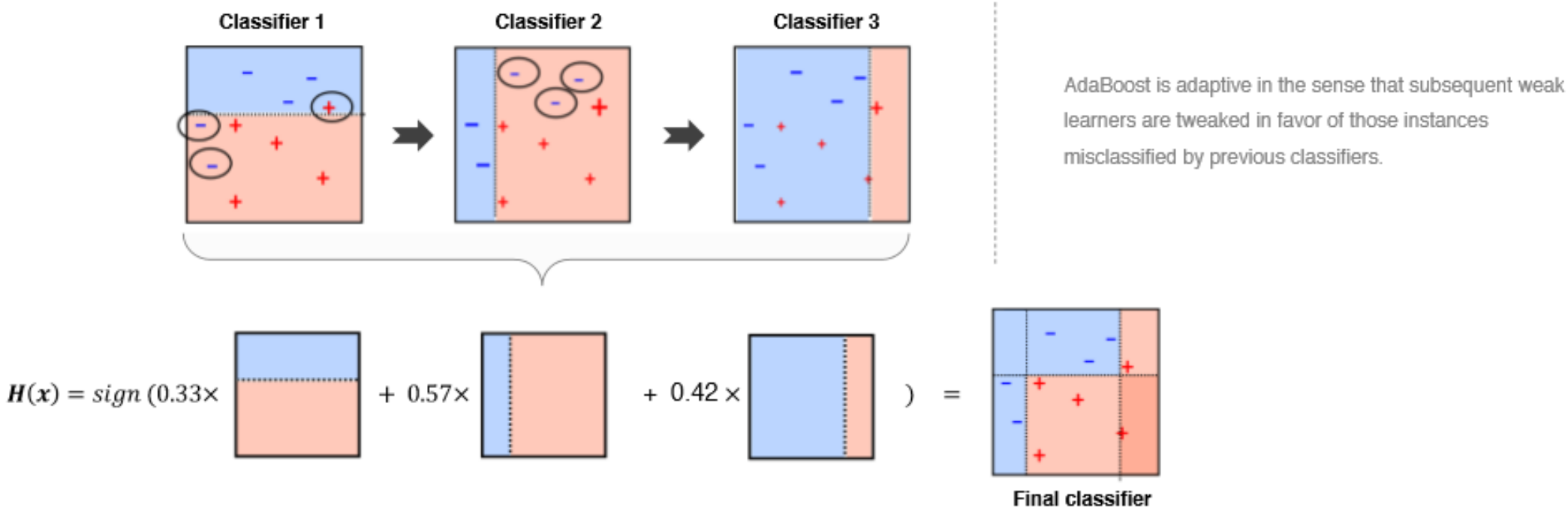
$\epsilon_t$ > 0.5 means classifier is not better than random guessing

The equation for the final classifier

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

- $h_t(x)$ is the output of weak classifier 't'.
- $\alpha_t$ is the weight applied to classifier 't' as determined by AdaBoost

So the final output is just a linear combination of all of the weak classifiers $h_t(x)$, and then we make our final decision simply by looking at the sign of this sum[2].




AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers.

**算法8.1**

- 输入：训练数据集$T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}, x \in \mathcal{X} \subseteq \mathbb{R}^n$, 弱学习方法
- 输出：最终分类器$G(x)$

**步骤**

1. 初始化训练数据的权值分布

$$D_1 = (w_{11}, \cdots, w_{1i}, \cdots, w_{1N}), w_{1i} = \frac{1}{N}, i = 1, 2, ..., N$$

2. $m = 1, 2, ..., M$

(a). 使用具有权重分布$D_m$的训练数据集学习，得到基本分类器

$$G_m(x) : X \to \{-1, +1\}$$

(b). 求$G_m$在训练集上的分类误差率

$$e_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

(c). 计算$G_m(x)$的系数，使用自然对数。

$$\alpha_m = \frac{1}{2}\log\frac{1-e_m}{e_m}$$

(d). 更新训练数据集的权重分布

$$D_{m+1} = (w_{11}, \cdots, w_{1i}, \cdots, w_{1N})$$

$$w_{m+1,i} = \frac{w_{m+1}}{Z_m}\exp(-\alpha_m y_i G_m(x_i))$$

这里$Z_m$是规范化因子

$$Z_m = \sum_{i=1}^{N} w_{mi}\exp(-\alpha_m y_i G_m(x_i))$$

3. 构建基本分类器的线性组合

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

得到最终分类器

$$G(x) = sign(f(x)) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$$

AdaBoost是个集成学习算法， 因为在他的输入中包含了**弱学习算法**。

# AdaBoost例子

## 例子8.1

# AdaBoost 算法的训练误差分析

## AdaBoost的训练误差界

## 二分类问题AdaBoost的训练误差界

- AdaBoost的训练误差是以指数速率下降的。

# AdaBoost 算法的解释

## 加法模型

$$f(x)=\sum_{m=1}^{M}\beta_m b(x;\gamma_m)$$

其中，$b(x;\gamma_m)$为基函数，$\beta_m$为基函数系数，$\gamma_m$为基函数参数。

在给定训练数据及损失函数$L\left(y, f\left(x\right)\right)$的条件下，学习加法模型$f\left(x\right)$成为经验风险极小化问题

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i; \gamma_m\right)\right)$$

学习加法模型，从前向后每一步只学习一个基函数及其系数，即每步只优化

$$\min_{\beta, \gamma} \sum_{i=1}^{N} L\left(y_i, \beta b\left(x_i; \gamma\right)\right)$$

## 前向分步算法

输入：训练数据集$T = (x_1, y_1), (x_2, y_2), ..., (x_N, y_N), x_i \in \mathcal{X} \subseteq R^n, y_i \in \{-1, 1\}$， 损失函数$L(y, f(x))$; 基函数集合$\{b(x; \gamma)\}$

输出：加法模型$f(x)$

步骤：

1. 初始化$f_0(x) = 0$
2. 对$m = 1, 2, \ldots, M$
   - 极小化损失函数 $(\beta_m, \gamma_m) = \arg\min\limits_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$
   - 更新$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$
3. 得到加法模型$f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$

## 前向分布算法与AdaBoost

**定理:** AdaBoost算法是前向分步加法算法的特例，模型由基本分类器组成的加法模型，损失函数是指数函数。

# 提升树

提升方法实际采用加法模型（即基函数的线性组合）与前向分步算法。

## 提升树模型

以决策树为基函数的提升方法称为提升树。

决策树$T(x;\Theta_m)$

提升树模型可以表示成决策树的加法模型

$$f_M(x)=\sum_{m=1}^{M}T(x;\Theta_m)$$

## 提升树算法

不同的问题， 主要区别在于损失函数不同：

1. 平方误差用于回归问题
2. 指数损失用于分类问题
3. 一般损失函数

## 回归问题的提升树算法

输入：训练数据集

输出：提升树$f_M(x)$

步骤：

1. 初始化$f_0(x)=0$
2. 对$m=1,2,\ldots,M$
    i. 计算残差$r_{mi}=y_i-f_{m-1}(x_i), i=1,2,\ldots,N$
    ii. **拟合残差**$r_{mi}$学习一个回归树，得到$T(x;\Theta_m)$
    iii. 更新$f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$
3. 得到回归问题提升树

$$f(x)=f_M(x)=\sum_{m=1}^{M}T(x;\Theta_m)$$

# 梯度提升(GBDT)

输入： 训练数据集$T = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N), x_i \in x \subseteq \mathbb{R}^n, y_i \in y \subseteq \mathbb{R}$；损失函数$L(y, f(x))$

输出：回归树$\hat{f}(x)$

步骤：

1. 初始化

$$f_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

2. 对$m = 1, 2, \ldots, M$

(a) $i = 1, 2, \ldots, N$

$$r_{mi} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

(b) 对$r_{mi}$拟合一个回归树，得到第$m$棵树的叶节点区域$R_{mj}, j = 1, 2, \ldots, J$

(c) $j = 1, 2, \ldots, J$

$$c_{mj} = \arg\min_{c} \sum_{xi \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)$$

(d) 更新

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

3. 得到回归树

$$\hat{f}(x) = f_M(x) = \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

# XGBoost

XGBoost全名叫（eXtreme Gradient Boosting）极端梯度提升，经常被用在一些比赛中，其效果显著。它是大规模并行boosted tree的工具，它是目前最快最好的开源boosted tree工具包。

## XGBoost目标函数

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{i=1}^{k} \Omega(f_k)$$

# XGBoost目标函数演化

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{i=1}^{k} \Omega(f_i) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_k) + constant$$

其中$t$表示第$t$轮，$f_t$表示第$t$轮生成的树模型，$\Omega(f_i)$表示正则项，$constant = \sum_{i=1}^{t-1} \Omega(f_i)$

泰勒展开：

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$$

$$L \simeq \sum_{i=1}^{n} \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

因为$l(y_i, \hat{y}_i^{(t-1)})$的值由以前的过程决定，本轮不变，constant也不影响本轮训练，得到：

$$L \simeq \sum_{i=1}^{n} \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

目标函数只依赖于每个数据点的误差函数的一阶导数和二阶导数。

对树的复杂度$\Omega(f_t)$进行定义：

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

$T$为叶子的个数，$q$是叶子权重部分。

## XGBoost与GDBT的区别

1. XGBoost生成CART树考虑了树的复杂度，GDBT未考虑，GDBT在树的剪枝步骤中考虑了树的复杂度。
2. XGBoost是拟合上一轮损失函数的二阶导展开，GDBT是拟合上一轮损失函数的一阶导展开，因此，XGBoost的准确性更高，且满足相同的训练效果，需要的迭代次数更少。
3. XGBoost与GDBT都是逐次迭代来提高模型性能，但是XGBoost在选取最佳切分点时可以开启多线程进行，大大提高了运行速度。
4. 关于XGboost的不足之处主要有：每轮迭代时，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间；预排序方法的时间和空间的消耗都很大

## 工具包参考

https://xgboost.readthedocs.io/en/latest/

# LightGBM

LightGBM is a new gradient boosting tree framework, which is highly efficient and scalable and can support many different algorithms including GBDT, GBRT, GBM, and MART. LightGBM is evidenced to be several times faster than existing implementations of gradient boosting trees, due to its fully greedy tree-growth method and histogram-based memory and computation optimization. It also has a complete solution for distributed training, based on the DMTK framework. The distributed version of LightGBM takes only one or two hours to finish the training of a CTR predictor on the Criteo dataset, which contains 1.7 billion records with 67 features, on a cluster of 16 machines.

# LightGBM简介

LightGBM是一个梯度Boosting框架，使用基于决策树的学习算法。它可以说是分布式的，高效的，有以下优势：

- 更快的训练效率
- 低内存使用
- 更高的准确率
- 支持并行化学习
- 可以处理大规模数据
- 与常见的机器学习算法对比，速度是非常快的

## 工具包参考

https://lightgbm.readthedocs.io/en/latest/

https://www.microsoft.com/en-us/research/project/lightgbm/

# 参考


1. [^1]: A Short Introduction to Boosting
2. [^2]: Boosting the margin: A new explanation for the effectiveness of voting methods
3. [^3 ]: Multi-class AdaBoost
4. [^4]: Improve boosting algorithms using confidence-rated predictions
5. [^5]: Machine Learning Techniques: Lecture 11: Gradient Boosted Decision Tree
6. [^6]: Logistic regression, AdaBoost and Bregman distances
7. [^7]: A decision-theoretic generalization of on-line learning and an application to boosting
8. 
9. https://zhuanlan.zhihu.com/p/126968534

# Enjoy your machine learning!

**https://github.com/wjssx/Statistical-Learning-Slides-Code**

E-mail: csr_dsp@sina.com