



浙江工业大学  
ZHEJIANG UNIVERSITY OF TECHNOLOGY



计算机科学与技术学院、软件学院  
College of Computer Science and Technology College of Software



# 机器学习-第八章 集成学习

黄亮 副教授

2022年03月

- 01 集成学习方法概述**
- 02 AdaBoost和GBDT算法**
- 03 XGBoost**
- 04 LightGBM**

# 1.集成学习方法概述

3

## 01 集成学习方法概述

## 02 AdaBoost和GBDT算法

## 03 XGBoost

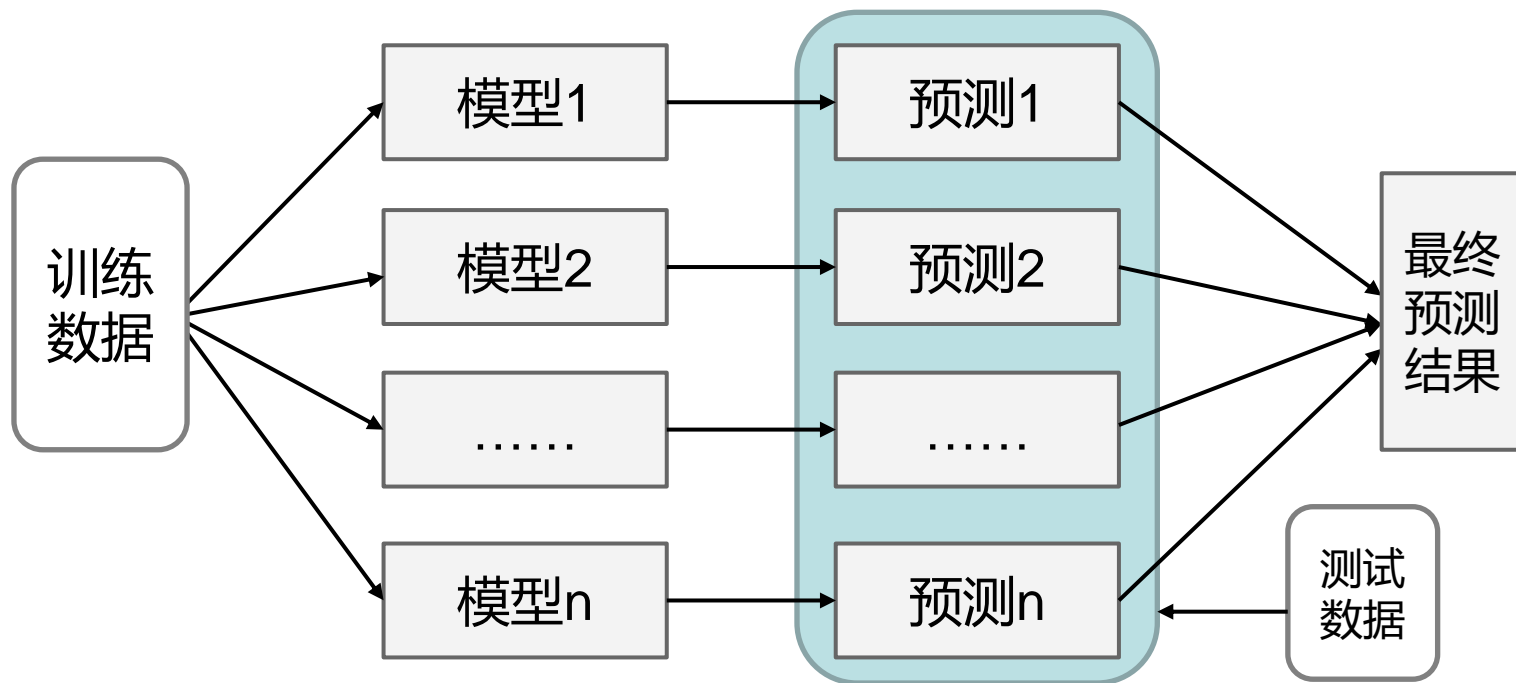
## 04 LightGBM

# 集成学习

4

## Bagging

从训练集中进行**子抽样**组成每个基模型所需要的子训练集，对所有基模型预测的结果进行综合产生最终的预测结果：

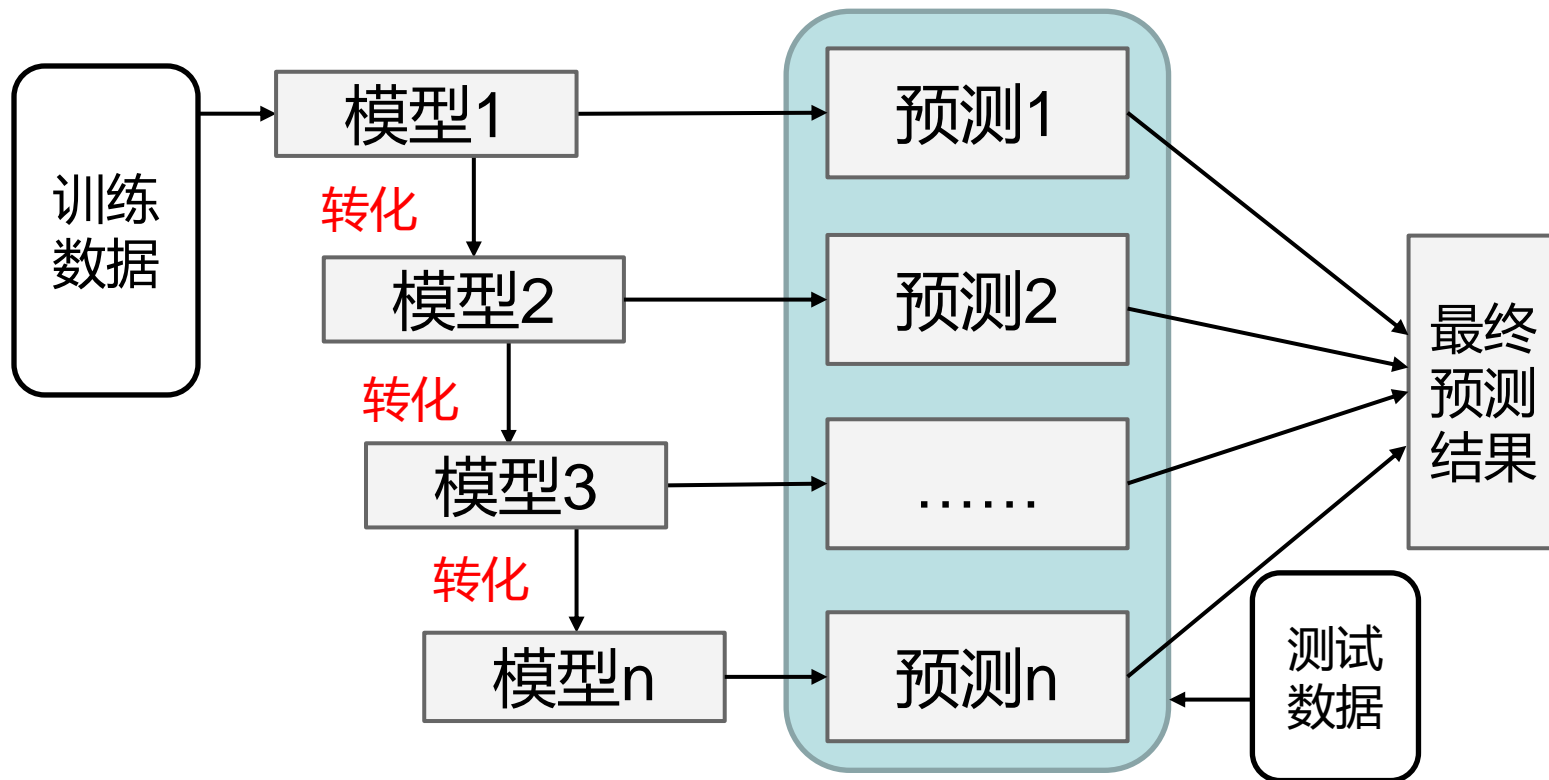


# 集成学习

5

## Boosting

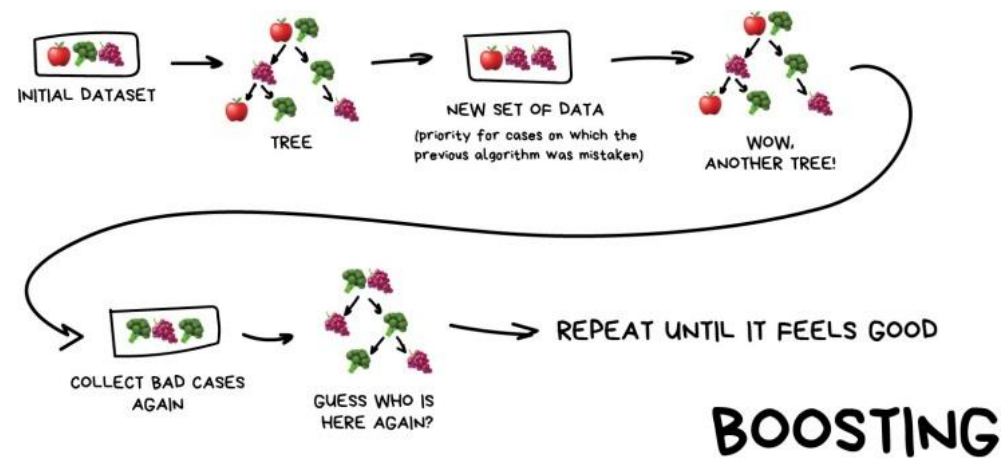
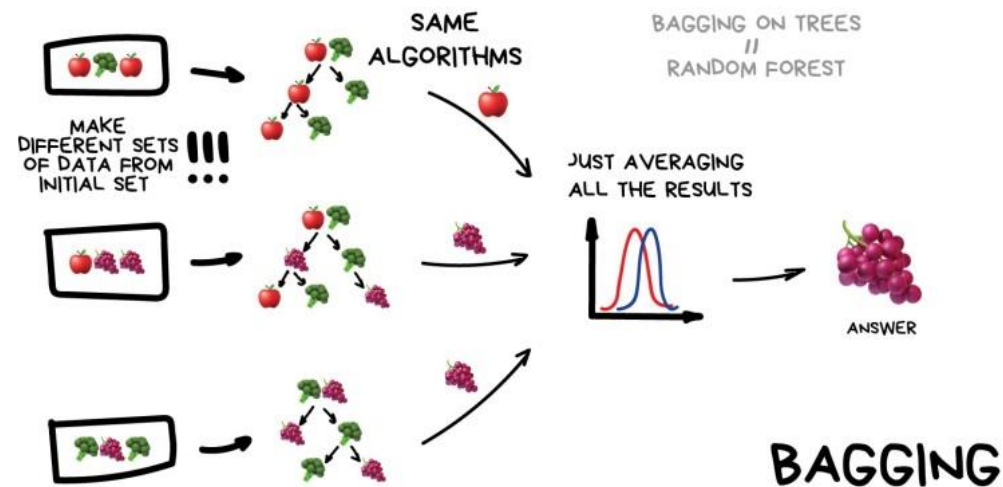
- 训练过程为阶梯状，**基模型**按次序逐个训练（可以**并行**）
- **基模型**的训练集按照某种策略每次都进行一定的转化。
- **综合**所有**基模型**预测的结果，产生最终的预测结果。



# 集成学习

6

- 三个臭皮匠顶个诸葛亮

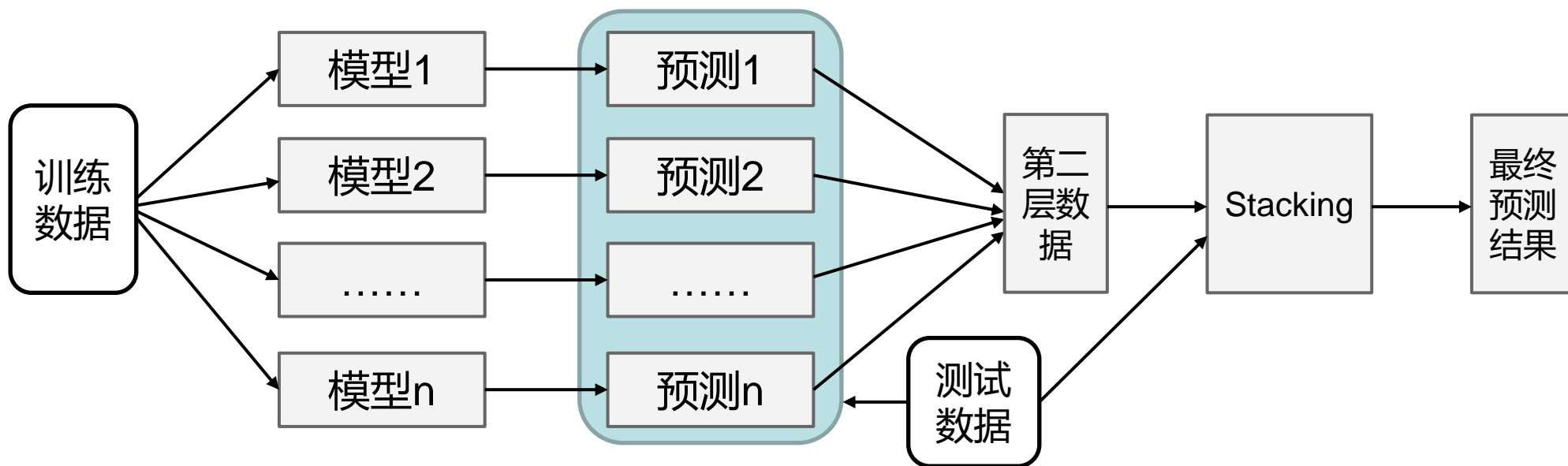


# 集成学习

7

## Stacking

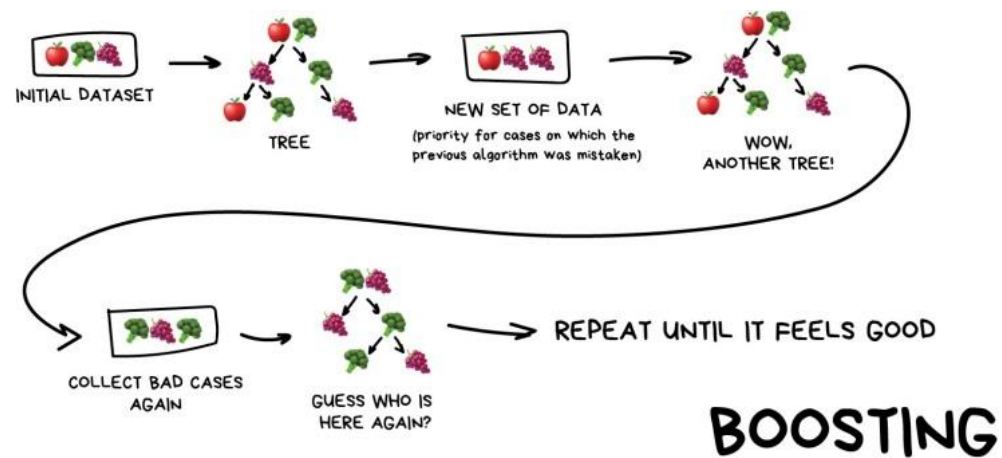
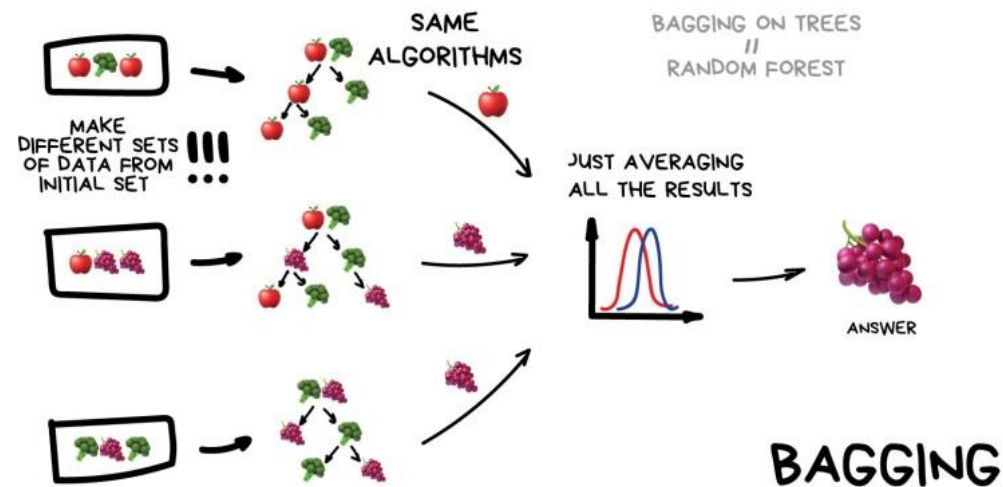
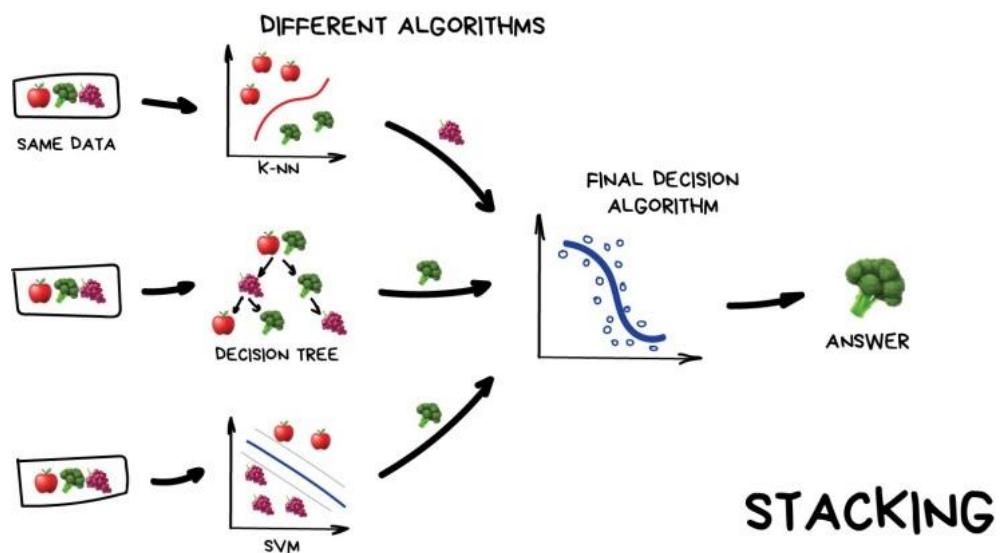
将训练好的所有基模型对训练基进行预测，第 $j$ 个基模型对第 $i$ 个训练样本的预测值将作为新的训练集中第 $i$ 个样本的第 $j$ 个特征值，最后基于新的训练集进行训练。同理，预测的过程也要先经过所有基模型的预测形成新的测试集，最后再对测试集进行预测。



# 集成学习

8

- 三个臭皮匠顶个诸葛亮





# 随机森林

9

## Random Forest (随机森林)

用随机的方式建立一个森林。随机森林算法由很多决策树组成，每一棵决策树之间没有关联。建立完森林后，当有新样本进入时，每棵决策树都会分别进行判断，然后基于投票法给出分类结果。

### 优点

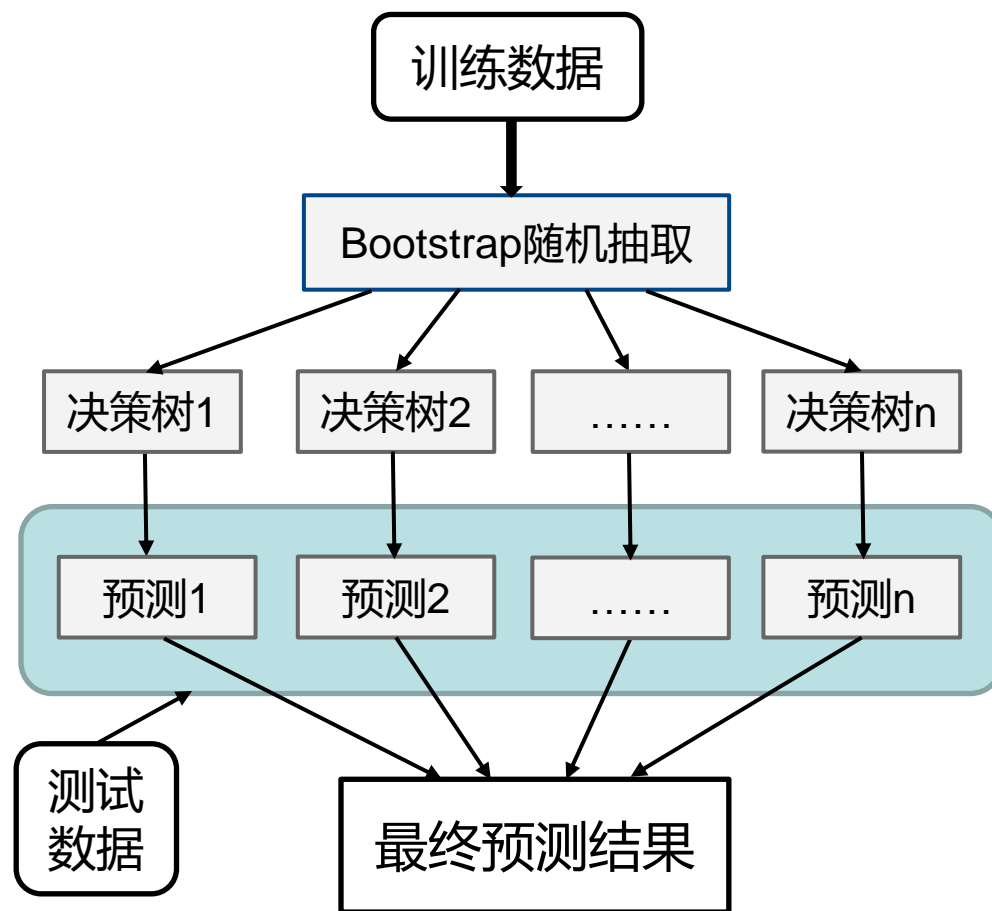
1. 在数据集上表现良好，相对于其他算法有较大的优势
2. 易于并行化，在大数据集上有很大的优势；
3. 能够处理高维度数据，不用做特征选择。

# 随机森林

10

**Random Forest (随机森林)** 是 Bagging 的扩展变体，它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特征选择，因此可以概括 随机森林包括四个部分：

1. 随机选择样本（放回抽样）；
2. 随机选择特征；
3. 构建决策树；
4. 随机森林投票（平均）。



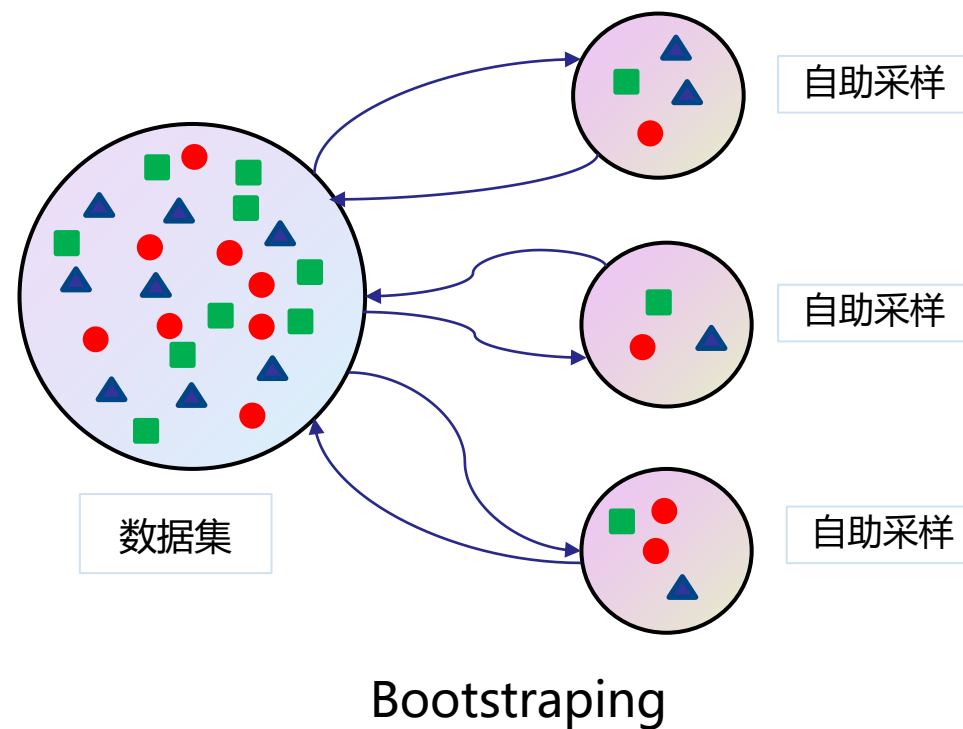
# 随机森林

11

随机选择样本和 Bagging 相同，采用的是 Bootstrapping 自助采样法；

**随机选择**特征是指在每个节点在分裂过程中都是随机选择特征的（区别与每棵树随机选择一批特征）。

这种随机性导致随机森林的偏差会有稍微的增加（相比于单棵不随机树），但是由于随机森林的“平均”特性，会使得它的方差减小，而且方差的减小补偿了偏差的增大，因此总体而言是更好的模型。



## 2.AdaBoost和GBDT算法

12

**01 集成学习方法概述**

**02 AdaBoost和GBDT算法**

**03 XGBoost**

**04 LightGBM**

# AdaBoost算法

13

**AdaBoost** (Adaptive Boosting, 自适应增强), 其自适应在于:

- 前一个基本分类器分错的样本会得到**加强**, 加权后的全体样本再次被用来训练下一个基本分类器。
- 同时, 在每一轮中加入一个**新**的弱分类器, 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。

**后一个模型的训练永远是在前一个模型的基础上完成!**

# Adaboost算法

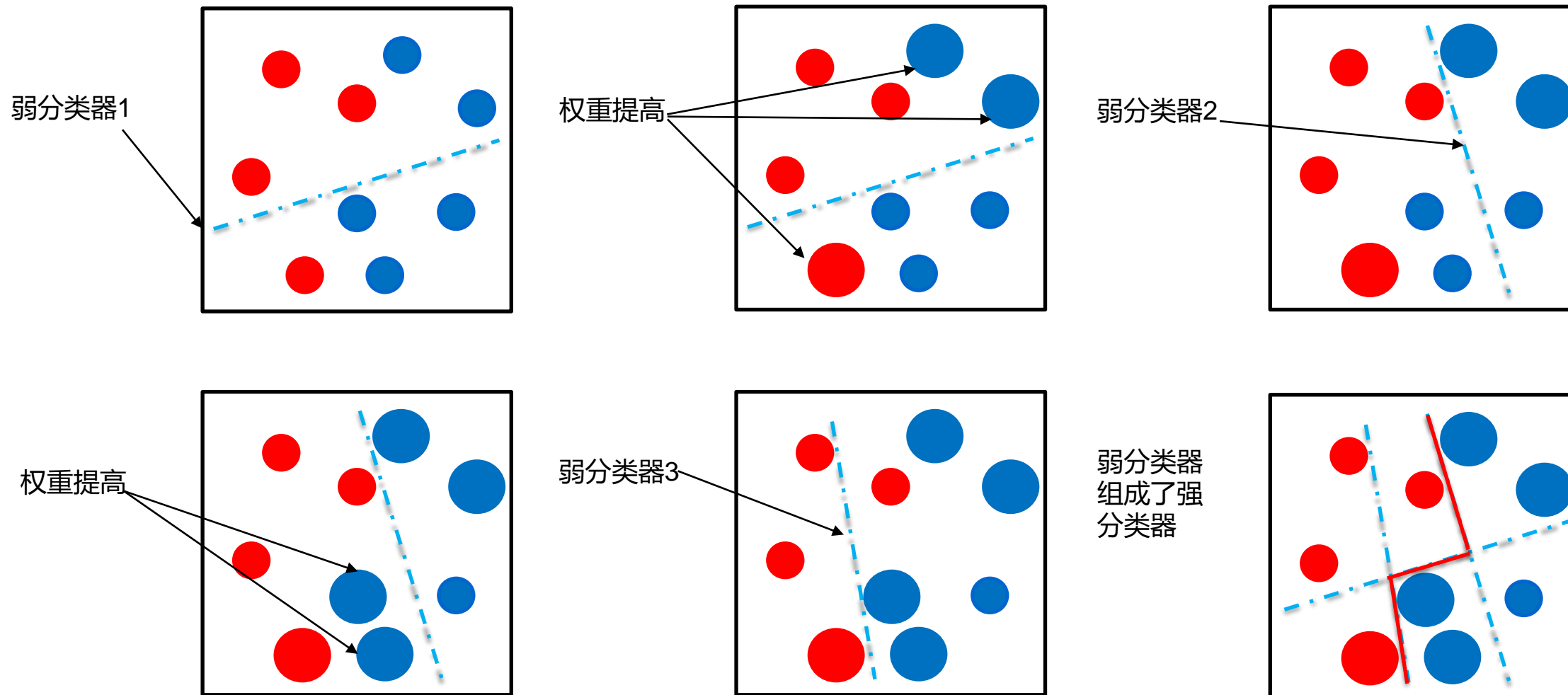
14

## 算法思想

- 初始化训练样本的权值分布，每个样本具有相同权重；
- 训练弱分类器，如果样本分类**正确**，则在构造下一个训练集中，它的权值就会被**降低**；反之提高。用更新过的样本集去训练下一个分类器；
- 将所有弱分类组合成强分类器，各个弱分类器的训练过程结束后，加大分类**误差率小**的弱分类器的权重，降低分类**误差率大**的弱分类器的权重。

# AdaBoost算法

15

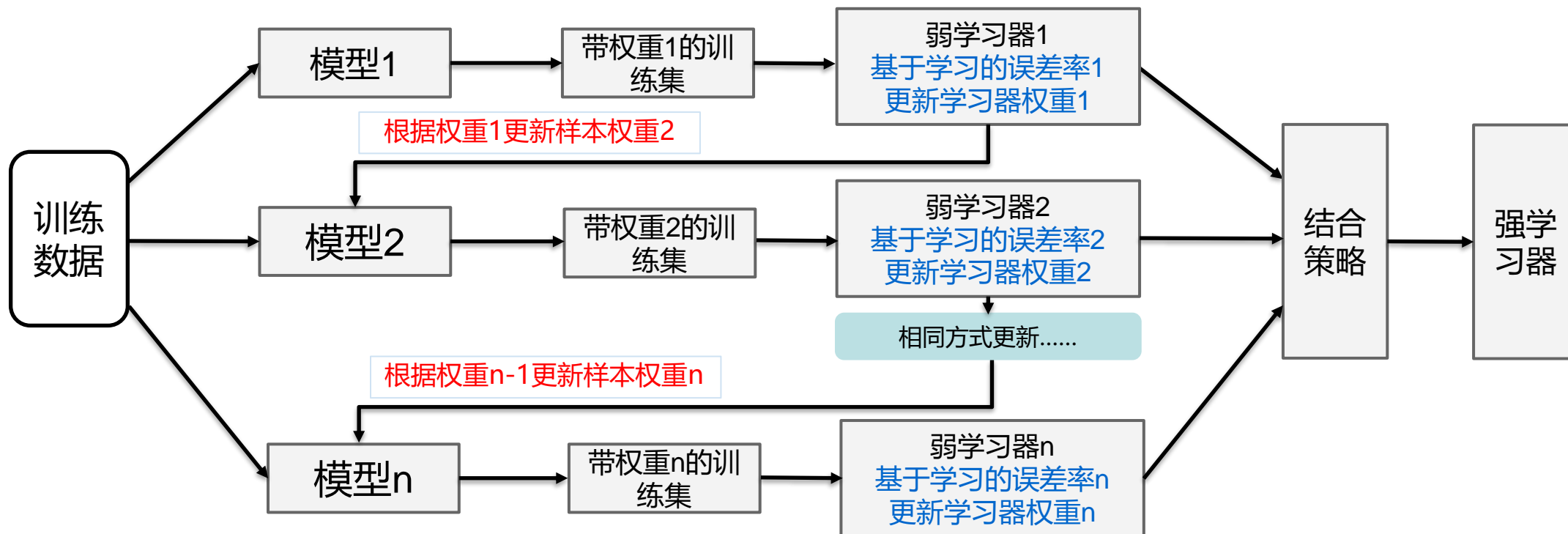


# Adaboost算法

16

算法思想：

后一个模型的训练永远是在前一个模型的基础上完成





# GBDT算法

17

GBDT (Gradient Boosting Decision Tree) 是一种迭代的决策树算法，该算法由多棵决策树组成，GBDT 的核心在于累加所有树的结果作为最终结果，所以 GBDT 中的树都是回归树，不是分类树，它是属于 Boosting 策略。GBDT 是被公认的泛化能力较强的算法。

GBDT 由三个概念组成：

Regression Decision Tree (即 DT)、Gradient Boosting (即 GB)，  
和 Shrinkage (缩减)

# GBDT算法

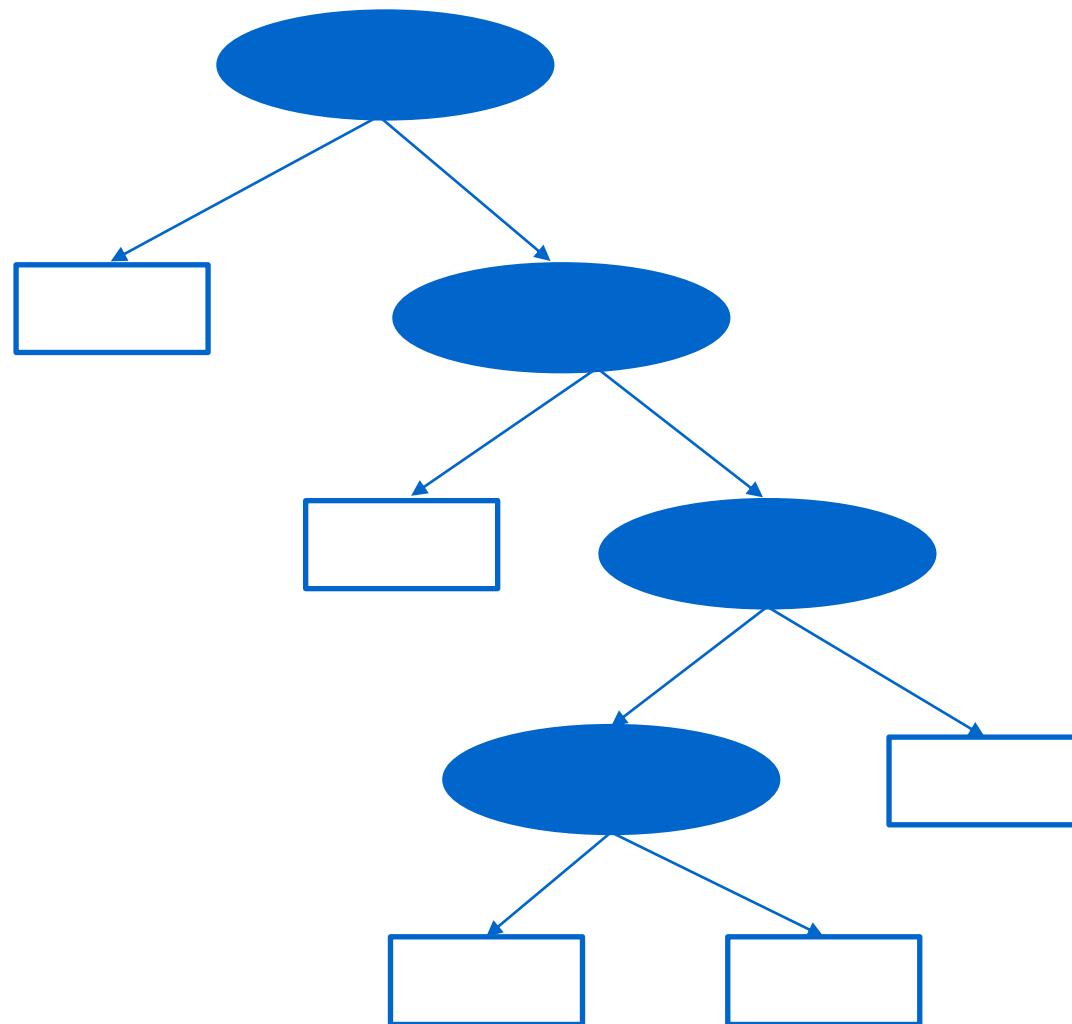
18

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

$$T(x; \theta) = \sum_{j=1}^J c_j I$$

$T$ 为决策树,  $\theta_m$ 为参数,  $M$ 为树的数量

$c$  为常数,  $J$  为叶子节点



# GBDT算法

19

## 前向分步算法:

$$f_0(x) = 0 \quad \leftarrow \text{初始化提升树}$$

迭代 $m$ 次, 包含 $m$ 棵决策树的提升树

$$f_m(x) = f_{m-1}(x) + T(x; \theta_m), m = 1, 2, \dots, M$$

第 $m$ 棵决策树

$$f_M(x) = \sum_{m=1}^M T(x; \theta_m) \quad \Rightarrow \quad \hat{\theta}_m = \arg \min_{\theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \theta_m))$$

真实值

损失函数

备注：损失函数选择：如分类用指数损失函数,回归使用平方误差损失。

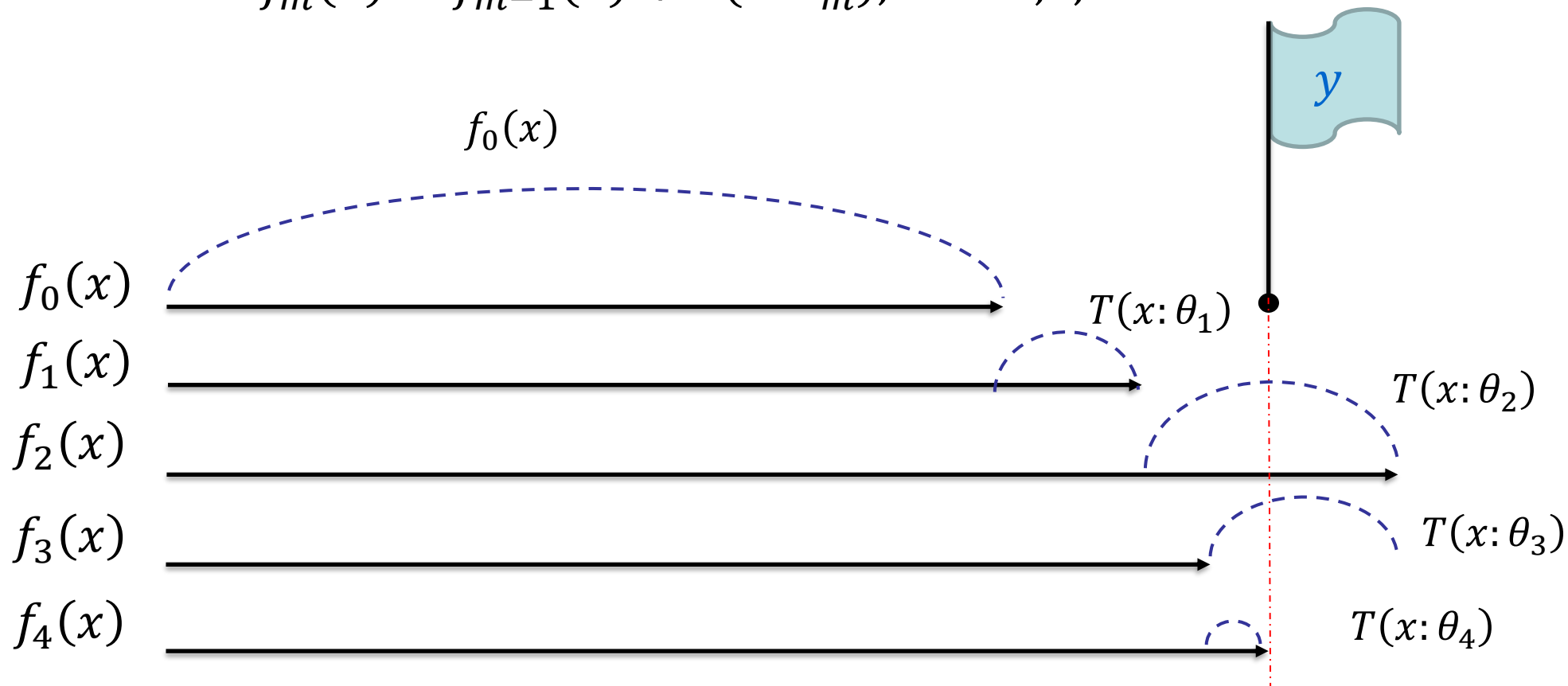
# GBDT算法

20

## 前向分步算法:



$$f_m(x) = f_{m-1}(x) + T(x; \theta_m), m = 1, 2, \dots, M$$



# GBDT算法

21

## 前向分步算法:

回归使用平方误差损失  $f_m(x) = f_{m-1}(x) + T(x: \theta_m)$

$$\begin{aligned} L(y, f(x)) &= L(y, f_m(x)) \\ &= L(y, f_{m-1}(x) + T(x: \theta_m)) \\ &= (y - f_m(x))^2 \\ &= [\underline{y - f_{m-1}(x)} - T(x: \theta_m)]^2 \\ &= [r - T(x: \theta_m)]^2 \end{aligned}$$

其中,  $r = y - f_{m-1}(x)$ , 为残差, 所以第 $m$ 棵决策树 $T(x: \theta_m)$ 是对该残差的拟合

注意: 提升树算法中的基学习器CART树是回归树

# GBDT算法

22

## 回归树问题的提升算法:

输入: 训练数据集  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$

输出: 提升树  $f_M(x)$

(1) 初始化  $f_0(x) = 0$

(2) 对  $m = 1, 2, \dots, M$

(a) 计算残差  $r_{mi} = y_i - f_{m-1}(x_i), i = 1, 2, \dots, N$

(b) 拟合残差  $r_{mi}$  学习一个回归树, 得到  $T(x; \theta_m)$

(c) 更新  $f_m(x) = f_{m-1}(x) + T(x; \theta_m)$

(3) 得到回归提升树

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

# GBDT算法

23

x	y
1	5.56
2	5.7
3	5.91
4	6.4
5	6.8
6	7.05
7	8.9
8	8.7
9	9
10	9.05

$$\min_s \left[ \min_{c_1} \sum (y_i - c_1)^2 + \min_{c_2} \sum (y_i - c_2)^2 \right]$$



$$R_1 = \{x | x \leq s\}, R_2 = \{x | x > s\}$$



$$c_1 = \frac{1}{N_1} \sum_{x_i \in R_1} y_i, c_2 = \frac{1}{N_2} \sum_{x_i \in R_2} y_i$$

1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5



$$m(s) = \min_{c_1} \sum_{x_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2} (y_i - c_2)^2$$



$$s = 1.5, R_1 = \{1\}, R_2 = \{2, 3, \dots, 10\}, c_1 = 5.56, c_2 = 7.5$$

s	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5
m(s)	15.72	12.07	8.36	5.78	3.91	1.93	8.01	11.73	15.74

# GBDT算法

24

x	y	y-f(x)
1	5.56	-0.68
2	5.7	-0.54
3	5.91	-0.33
4	6.4	0.16
5	6.8	0.56
6	7.05	0.81
7	8.9	-0.01
8	8.7	-0.21
9	9	0.09
10	9.05	0.14

$$s = 6.5, R_1 = \{1, 2, \dots, 6\}, R_2 = \{7, 8, \dots, 10\}, c_1 = 6.24, c_2 = 8.91$$

$$f_1(x) = T_1(x) = \begin{cases} 6.24 & x < 6.5 \\ 8.91 & x \geq 6.5 \end{cases}$$

$$L(y, f_1(x)) = \sum_{i=1}^{10} (y_i - f_1(x_i))^2 = 1.93$$

$$T_2(x) = \begin{cases} -0.52 & x < 3.5 \\ 0.22 & x \geq 3.5 \end{cases}$$

$$f_2(x) = f_1(x) + T_2(x) = \begin{cases} \begin{cases} 5.72 & x < 3.5 \\ 8.91 & x \geq 3.5 \end{cases} & x < 6.5 \\ 8.91 & x \geq 6.5 \end{cases}$$

$$L(y, f_2(x)) = \sum_{i=1}^{10} (y_i - f_2(x_i))^2 = 0.79$$



# GBDT算法

25

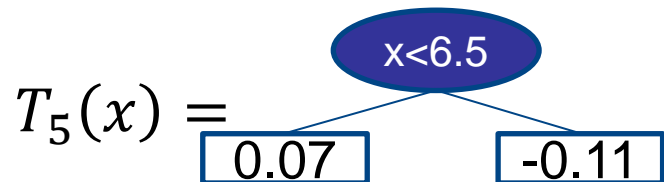


$$L(y, f_3(x)) = 0.47$$

$$f_6(x) = f_5(x) + T_6(x) = T_1(x) + \dots + T_6(x) =$$



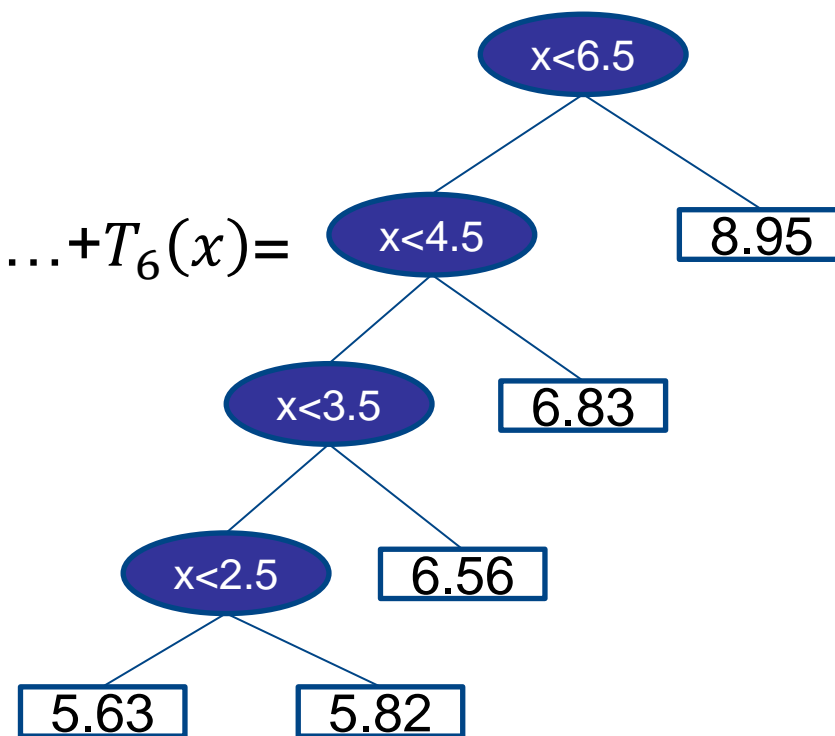
$$L(y, f_4(x)) = 0.30$$



$$L(y, f_5(x)) = 0.23$$

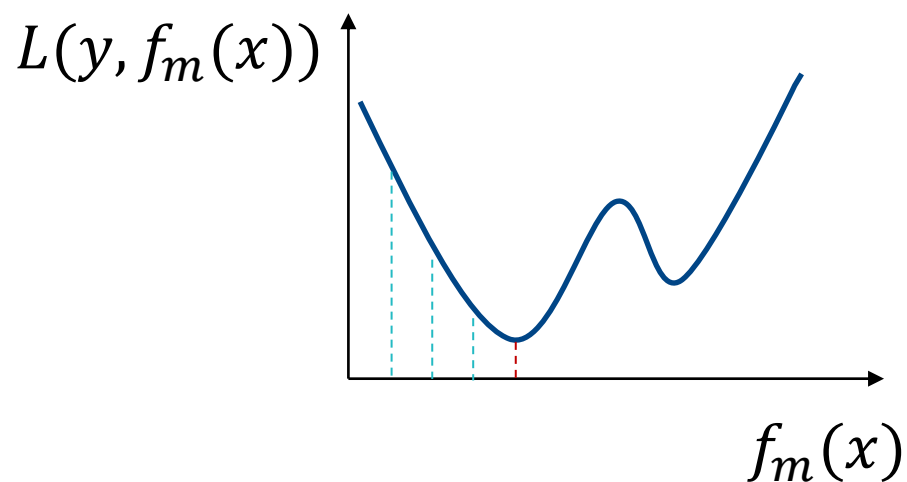
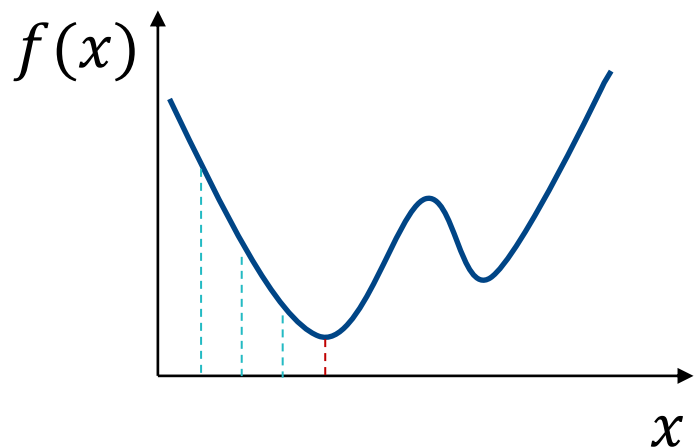


$$L(y, f_6(x)) = 0.17$$



# GBDT算法

26



损失函数的负梯度在当前模型的值作为提升树的残差的近似值来拟合回归树

# GBDT算法

27

## 回归树的梯度提升算法:

输入: 训练数据集  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ , 损失函数  $L(y, f(x))$

输出: 提升树  $f_M(x)$

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

(2) 对  $m = 1, 2, \dots, M$

(a) 对  $i = 1, 2, \dots, N$  计算

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

(b) 拟合  $r_{mi}$  学习一个回归树, 得到  $T(x; \theta_m)$

(c) 计算步长,  $\sigma_m = \arg \min_{\sigma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \sigma T(x; \theta_m))$

(d) 更新  $f_m(x) = f_{m-1}(x) + \sigma_m T(x; \theta_m)$

# 3.XGBoost

28

**01 集成学习方法概述**

**02 AdaBoost和GBDT算法**

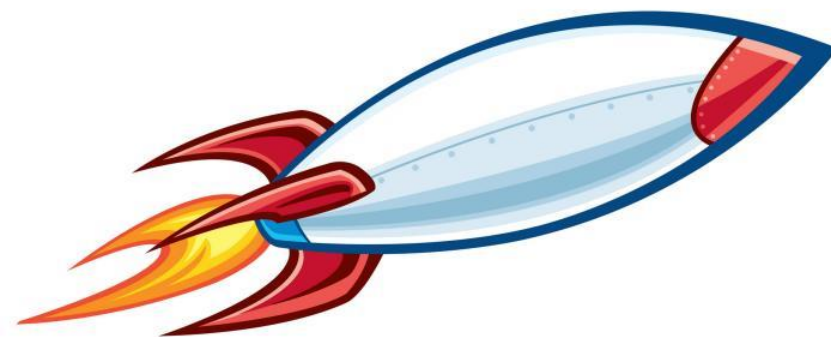
**03 XGBoost**

**04 LightGBM**

# 3.XGBoost

29

**XGBoost** 是大规模并行 boosting tree 的工具，它是目前最快最好的开源 boosting tree 工具包，比常见的工具包快 10 倍以上。XGBoost 和 GBDT 两者都是 boosting 方法，除了工程实现、解决问题上的一些差异外，最大的不同就是目标函数的定义。



# 3.XGBoost

30

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

泰勒展开

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

$\Delta x = f_t(x_i)$ , 表示第  $t$  棵树的输出  
结果等于  $t - 1$  时刻新增加的结果



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

$$g_i = \frac{\partial L(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}$$

$$h_i = \frac{\partial^2 L(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)^2}}$$

备注： $n$ 阶泰勒公式： $f(x) = f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \dots + \frac{f^{(n)}(0)}{n!}x^n + R_n(x) \dots$

$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$  称为  $f(x)$  在点  $x_0$  处的  $n$  阶泰勒余项。

# 3.XGBoost

31

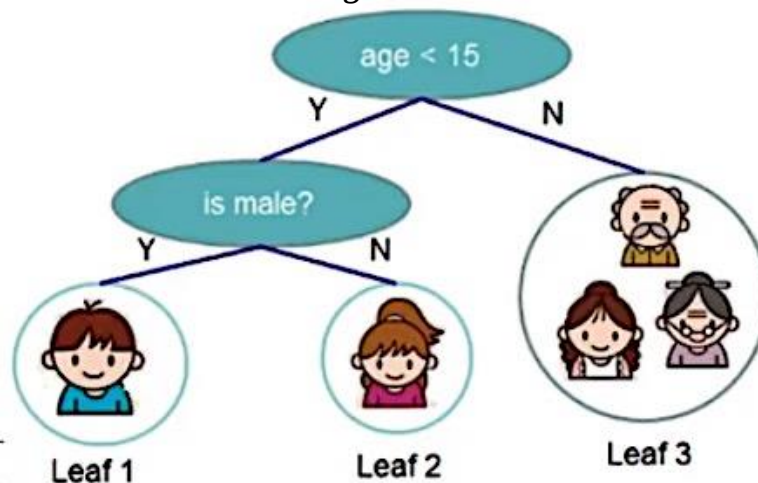
保留  $t - 1$  轮的模型预测结果

加入新的预测函数

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

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

gamma    lambda



$$\Omega = \gamma 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

# 3.XGBoost

32

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

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

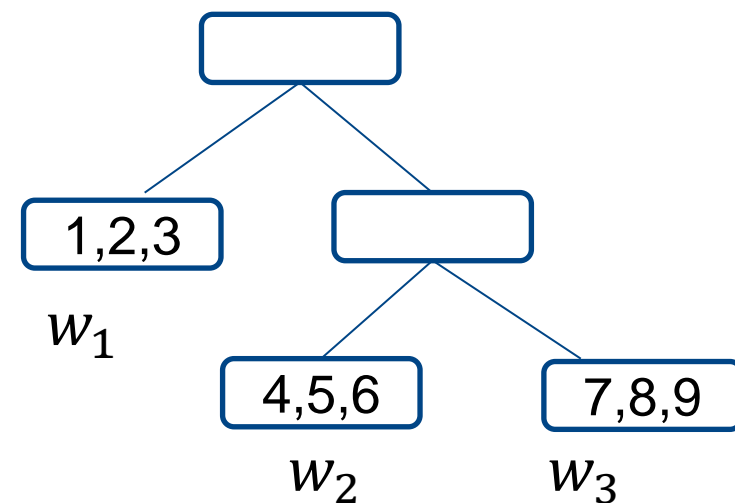
$$= \sum_{i=1}^n \left[ g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + C$$

$w_{q(x_i)}$  第t棵树的叶子结点值

$$= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + C$$

$$= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T + C$$

$$f_t(x) = w_{q(x)}, \quad w \in R^T, q: R^d \rightarrow \{1, 2, \dots, T\}$$





# 3.XGBoost

33

$$J(f_t) = \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T + C$$

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

$$J(f_t) = \sum_{j=1}^T \left[ G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T + C$$






$$= -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

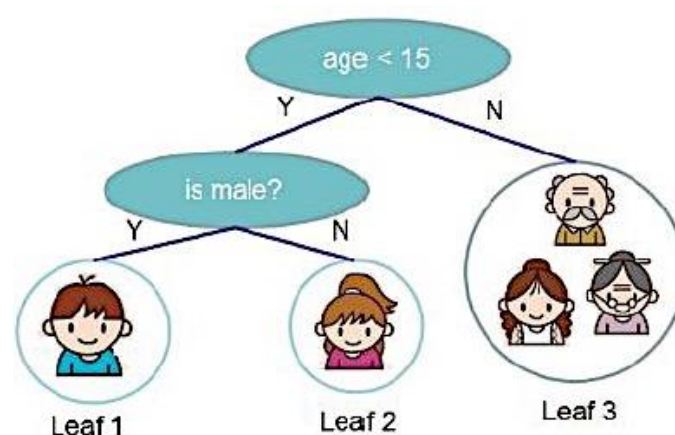
$$\frac{\partial J(f_t)}{\partial w_j} = G_j + (H_j + \lambda) w_j == 0$$

$$w_j = -\frac{G_j}{H_j + \lambda}$$

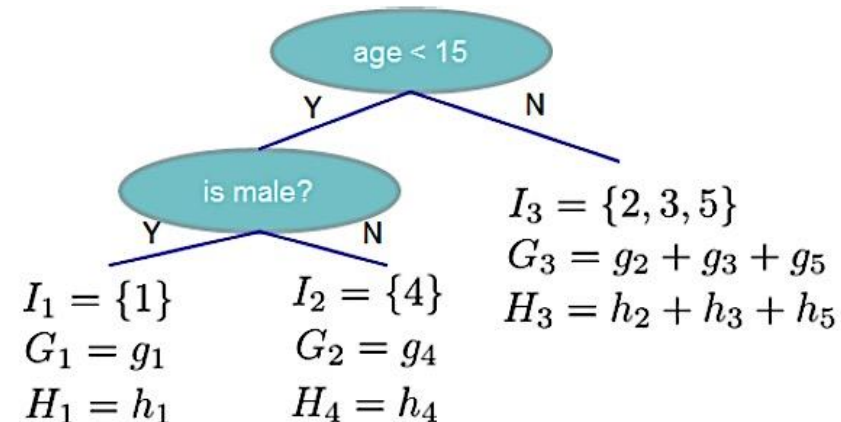
# 3.XGBoost

34

样本号	梯度数据
1 	$g_1, h_1$
2 	$g_2, h_2$
3 	$g_3, h_3$
4 	$g_4, h_4$
5 	$g_5, h_5$



$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$



↓

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

分数越小，代表这个树的结构越好

# 3.XGBoost

35

$$obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

$$gain(\emptyset) = gain(before) - gain(after)$$

$$= \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

加入新叶子节点引入的复杂度代价



分裂后左子树分数



分裂后右子树分数



分裂前左、右子树的分数：  
不分割可以拿到的分数

# 3.XGBoost

36

## XGBoost的分裂方式

使用贪心方法，选增益（*gain*）最大的分裂方式

贪心方法，众多*gain*中找到最大值做为最优分割节点（split point），因此模型会将所有样本按照（一阶梯度）从小到大排序，通过遍历，查看每个节点是否需要分割，计算复杂度是：决策树叶子节点数 - 1。

# 4.LightGBM

37

**01 集成学习方法概述**

**02 Adaboost和GBDT算法**

**03 XGBoost**

**04 LightGBM**

# 4.LightGBM

38

**LightGBM** 由微软提出，主要用于解决 GDBT 在海量数据中遇到的问题，以便其可以更好更快地用于工业实践中，其相对 XGBoost 具有训练速度快、内存占用低的特点。

LightGBM与XGBoost相比，主要有以下几个优势：

- 1) 更快的训练速度
- 2) 更低的内存消耗
- 3) 更好的准确率
- 4) 分布式支持，可快速处理海量数据

# 4.LightGBM

39

## LightGBM 的主要改进

LightGBM与XGBoost相比, 主要有以下几个改进:

- 基于梯度的单边采样算法 (Gradient-based One-Side Sampling, GOSS) ;
- 互斥特征捆绑算法 (Exclusive Feature Bundling, EFB) ;
- 直方图算法 ( Histogram ) ;
- 基于最大深度的 Leaf-wise 的垂直生长算法;

**LightGBM = XGBoost + GOSS + EFB+ Histogram**

# 4.LightGBM

40

## 基于梯度的单边采样算法 (Gradient-based One-Side Sampling, GOSS)

主要思想是通过对样本采样的方法来减少计算目标函数增益时候的复杂度。

GOSS 算法保留了梯度大的样本，并对梯度小的样本进行随机抽样，为了不改变样本的数据分布，在计算增益时为梯度小的样本引入一个常数进行平衡。

**如果一个样本的梯度很小，说明该样本的训练误差很小，或者说该样本已经得到了很好的训练(well-trained)。**



# 4.LightGBM

41

## 基于梯度的单边采样算法 (Gradient-based One-Side Sampling, GOSS)

输入：训练数据，迭代步数 $d$ ，大梯度数据的采样率 $a$ ，小梯度数据的采样率 $b$ ，损失函数和若学习器的类型（一般为决策树）

输出：训练好的强学习器

- (1) 根据样本点的梯度的绝对值对它们进行降序排序；
- (2) 对排序后的结果选取前 $a*100\%$ 的样本生成一个大梯度样本点的子集；
- (3) 对剩下的样本集合 $(1-a)*100\%$ 的样本，随机的选取 $b*(1-a)*100\%$ 个样本点，生成一个小梯度样本点的集合；
- (4) 将大梯度样本和采样的小梯度样本合并；
- (5) 将小梯度样本乘上一个权重系数 $\frac{1-a}{b}$ ；
- (6) 使用上述的采样的样本，学习一个新的弱学习器；
- (7) 不断地重复 (1) ~ (6) 步骤直到达到规定的迭代次数或者收敛为止。

# 4.LightGBM

42

		bin1				bin2			bin3	
样本序号	$i$	1	2	3	4	5	6	7	8	
样本的特征取值	$x_i$	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0	可能的候选点分裂点个数 等于样本取值个数减一
样本的一阶导	$g_i$	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07	
样本的二阶导	$h_i$	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03	

排序完了之后，我们就选出 $a * data\_num$ 个梯度大的，然后从剩下的那些样本里面选出 $b*data\_num$ 个梯度小的：  
这里是8个样本，所以 $a*8=2$ ， $b*8=2$ ， $\frac{1-a}{b} = 3$ 。  
即先选出2个梯度大的样本，然后从剩下的里面随机选出2个梯度小的样本  
这里选取两个大的（6号、7号），然后随机选择两个小的（2号、4号）

bin序号	bin1	bin2	bin3
bin样本之和	$N_i$	$1*3+1$	1
bin内所有样本的一阶导之和	$g_i$	$0.03*3$	$0.05*3+0.7$
bin内所有样本的二阶导之和	$h_i$	$0.04*3$	$0.12*3+0.02$

# 4.LightGBM

43

## 互斥特征捆绑算法 (Exclusive Feature Bundling, EFB)

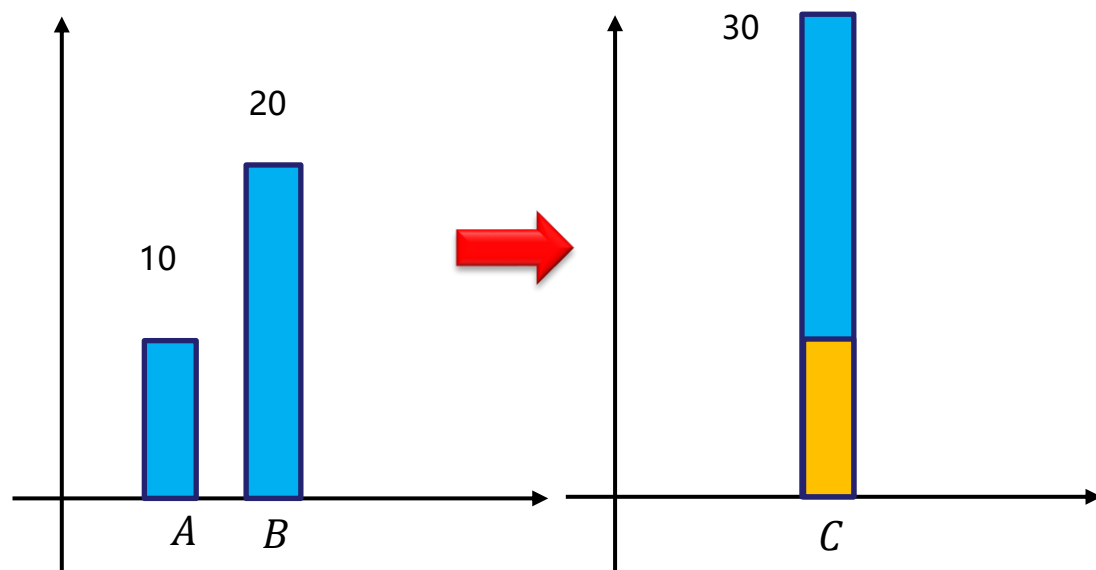
- 高维特征往往是稀疏的，而且特征间可能是相互排斥的（如两个特征不同时取非零值）
- 如果两个特征并不完全互斥（如只有一部分情况下是不同时取非零值），可以用互斥率表示互斥程度。
- EFB算法指出如果将一些特征进行融合绑定，则可以降低特征数量。

论文给出特征合并算法，其关键在于原始特征能从合并的特征中分离出来。

# 4.LightGBM

44

## 互斥特征捆绑算法 (Exclusive Feature Bundling, EFB)



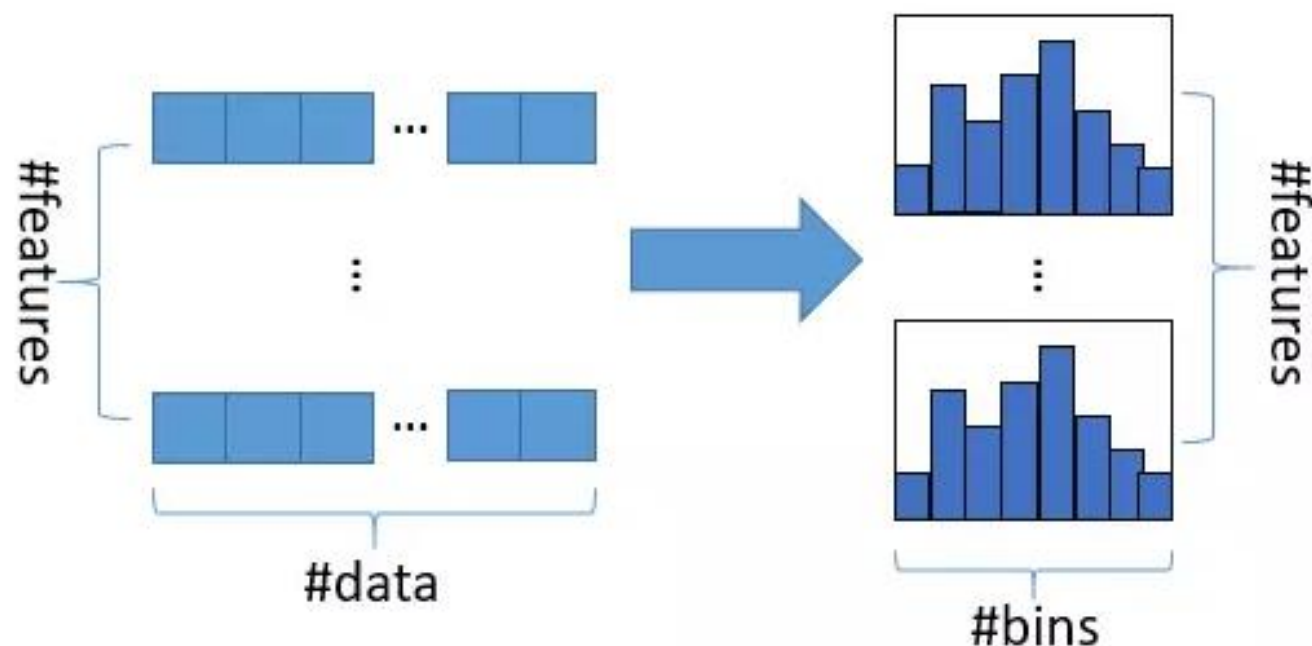
假设 Bundle中有两个特征值, A取值为 $[0,10]$ , B取值为 $[0,20]$ , 为了保证特征A、B的互斥性, 我们可以给特征B添加一个偏移量转换为 $C[10,30]$ , Bundle后的特征其取值为 $[0,30]$ , 这样便实现了特征合并。

# 4.LightGBM

45

## 直方图算法

直方图算法的基本思想是将连续的特征离散化为 $k$ 个离散特征，同时构造一个宽度为 $k$ 的直方图用于统计信息（含有 $k$ 个 bin）。利用直方图算法我们无需遍历数据，只需要遍历 $k$ 个 bin 即可找到最佳分裂点。



# 4.直方图算法

样本序号	$i$	1	2	3	4	5	6	7	8
样本的特征取值	$x_i$	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0
样本的一阶导	$g_i$	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07
样本的二阶导	$h_i$	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03

可能的候选点分裂点个数  
等于样本取值个数减一

	bin1			bin2			bin3	
$i$	1	2	3	4	5	6	7	8
$x_i$	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0
$g_i$	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07
$h_i$	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03

bin序号	bin $i$	bin1	bin2	bin3
bin样本之和	$N_i$	3	3	2
bin内所有样本的一阶导之和	$g_i$	0.1	0.79	0.67
bin内所有样本的二阶导之和	$h_i$	0.29	0.12	0.06

可能的候选点分裂点个数等于bins个数减一

# 4.LightGBM

47

## 直方图加速

在构建叶节点的直方图时，我们还可以通过父节点的直方图与相邻叶节点的直方图相减的方式构建，从而减少了一半的计算量。即：**一个叶子节点的直方图可以由它的父亲节点的直方图与其兄弟的直方图做差得到**。如节点分裂成两个时，右边叶子节点的直方图等于其父节点的直方图减去左边叶子节点的直方图。从而大大减少构建直方图的计算量。



# 4.LightGBM

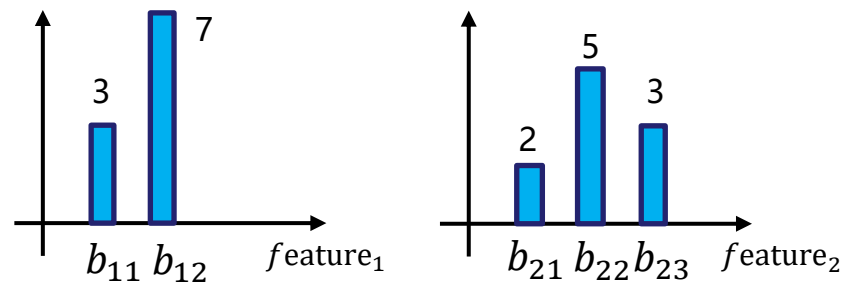
48

## 直方图算法

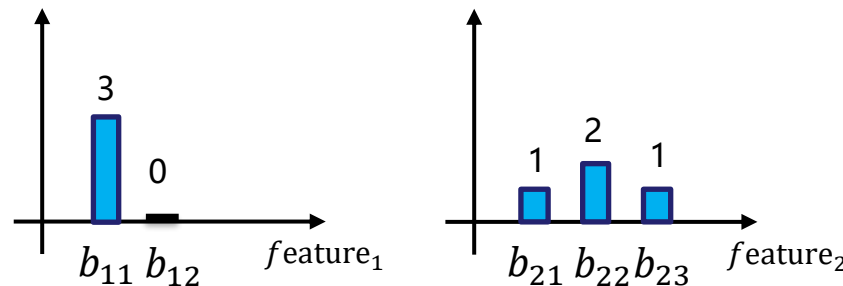
直方图算法还可以进一步加速：一个叶子节点的直方图可以由它的父亲节点的直方图与其兄弟的直方图做差得到。

父节点10个样本，2个特征。

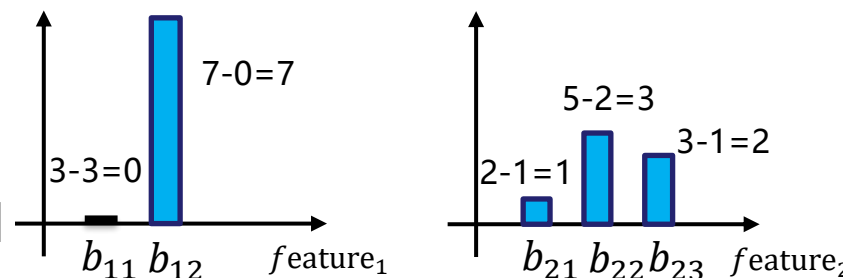
①父节点特征的直方图



②左子节点特征的直方图



③右子节点特征的直方图

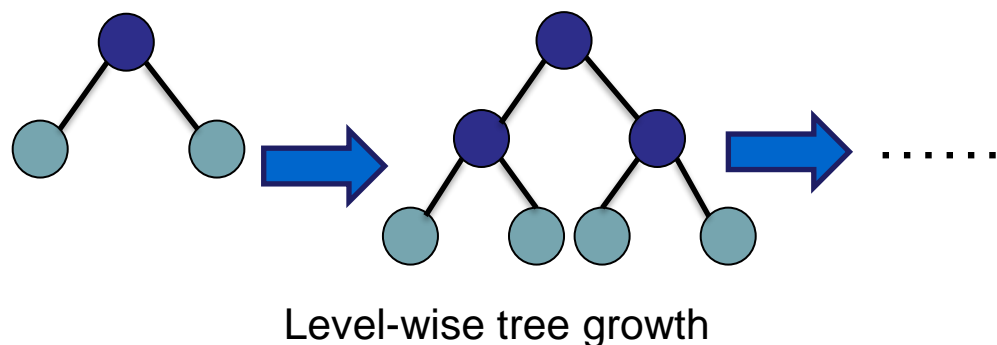




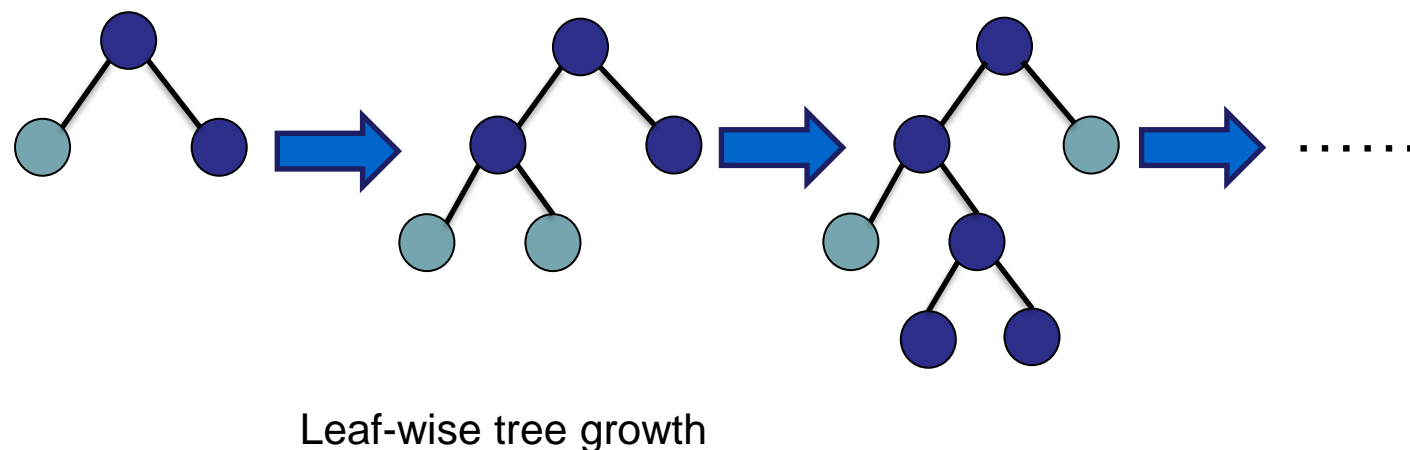
# 4.LightGBM

49

## 基于最大深度的 Leaf-wise 的垂直生长算法



**XGBoost**通过Level-wise tree growth策略来生长树。同一层所有节点都做分裂，最后剪枝

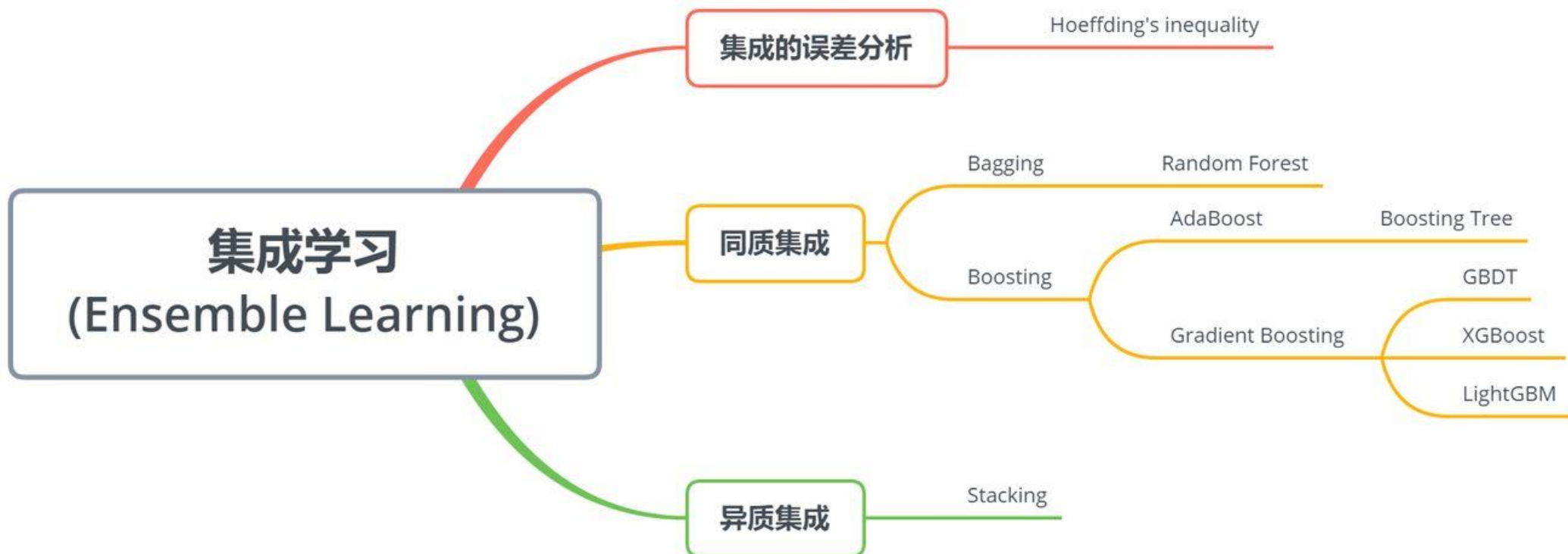


**LightGBM** 通过 leaf-wise (best-first)策略来生长树。它将选取具有最大 delta loss 的叶节点来生长。

建树过程的两种方法：Level-wise和Leaf-wise

# 决策树

50



1. 《统计学习方法》，清华大学出版社，李航著，2019年出版
2. 《机器学习》，清华大学出版社，周志华著，2016年出版
3. Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag, 2006
4. [1] Chen T , Tong H , Benesty M . XGBoost: Extreme Gradient Boosting[J]. 2016.
5. <https://zhuanlan.zhihu.com/p/149522630>

谢谢!

摄影：机械工程学院 何迪