



北京交通大学
BEIJING JIAOTONG UNIVERSITY



1

机器学习

集成学习

鲍鹏
北京交通大学

本章目录

2

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

1.个体与集成

3

01 个体与集成

02 集成学习方法概述

03 AdaBoost和GBDT算法

04 XGBoost

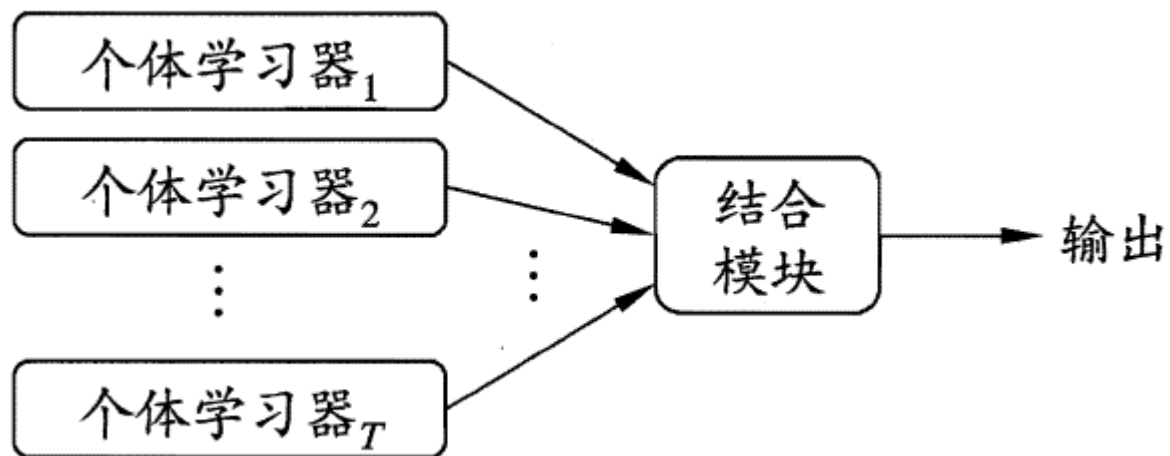
05 LightGBM

1.个体与集成

4

- 个体与集成

➤ 集成学习(ensemble learning)：通过构建并结合多个学习器来完成学习任务。



1.个体与集成

5

• 个体与集成

- 在二分类问题中，假定三个分类器在三个测试样本上的表现如下图所示，其中√表示分类正确，×表示分类错误，集成的结果通过投票产生。

	测试例1	测试例2	测试例3
h_1	√	√	×
h_2	×	√	√
h_3	√	×	√
集成	√	√	√

(a) 集成提升性能

	测试例1	测试例2	测试例3
h_1	√	√	×
h_2	√	√	×
h_3	√	√	×
集成	√	√	×

(b) 集成不起作用

	测试例1	测试例2	测试例3
h_1	√	×	×
h_2	×	√	×
h_3	×	×	√
集成	×	×	×

(c) 集成起负作用

- 集成个体应：“好而不同”

1.个体与集成

6

• 个体与集成-简单分析

- 考虑二分类问题，假设基分类器的错误率为：

$$P(h_i(x) \neq f(x)) = \epsilon$$

- 假设集成通过简单投票法结合 T 个分类器，若有超过半数的基分类器正确则分类就正确：

$$F(x) = \text{sign}\left(\sum_{i=1}^T h_i(x)\right), \quad \text{sign} = \begin{cases} 0, & \sum_{i=1}^T h_i(x) < 0 \\ 1, & \sum_{i=1}^T h_i(x) > 0 \end{cases}$$

1.个体与集成

7

• 个体与集成-简单分析

➤ 假设基分类器的错误率相互独立，则由Hoeffding不等式可得集成的错误率为：

$$\begin{aligned} P(F(x) \neq f(x)) &= \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \\ &\leq \exp\left(-\frac{1}{2}T(1-2\epsilon)^2\right) \end{aligned}$$

➤ 上式显示，理想条件下，随着集成分类器数目的增加，集成的错误率将指数级下降，最终趋向于0。

1.个体与集成

8

• 个体与集成-小结

- 上面的分析有一个关键假设：基学习器的误差相互独立
- 现实任务中，个体学习器是为解决同一个问题训练出来的，显然不可能互相独立
- 事实上，个体学习器的“准确性”和“多样性”本身就存在冲突
- 如何产生“好而不同”的个体学习器是集成学习研究的核心

2.集成学习方法概述

9

01 个体与集成

02 集成学习方法概述

03 AdaBoost和GBDT算法

04 XGBoost

05 LightGBM

2.集成学习方法概述

10

- **集成学习方法分类**

- 集成学习大致可分为两大类：

- Bagging与随机森林

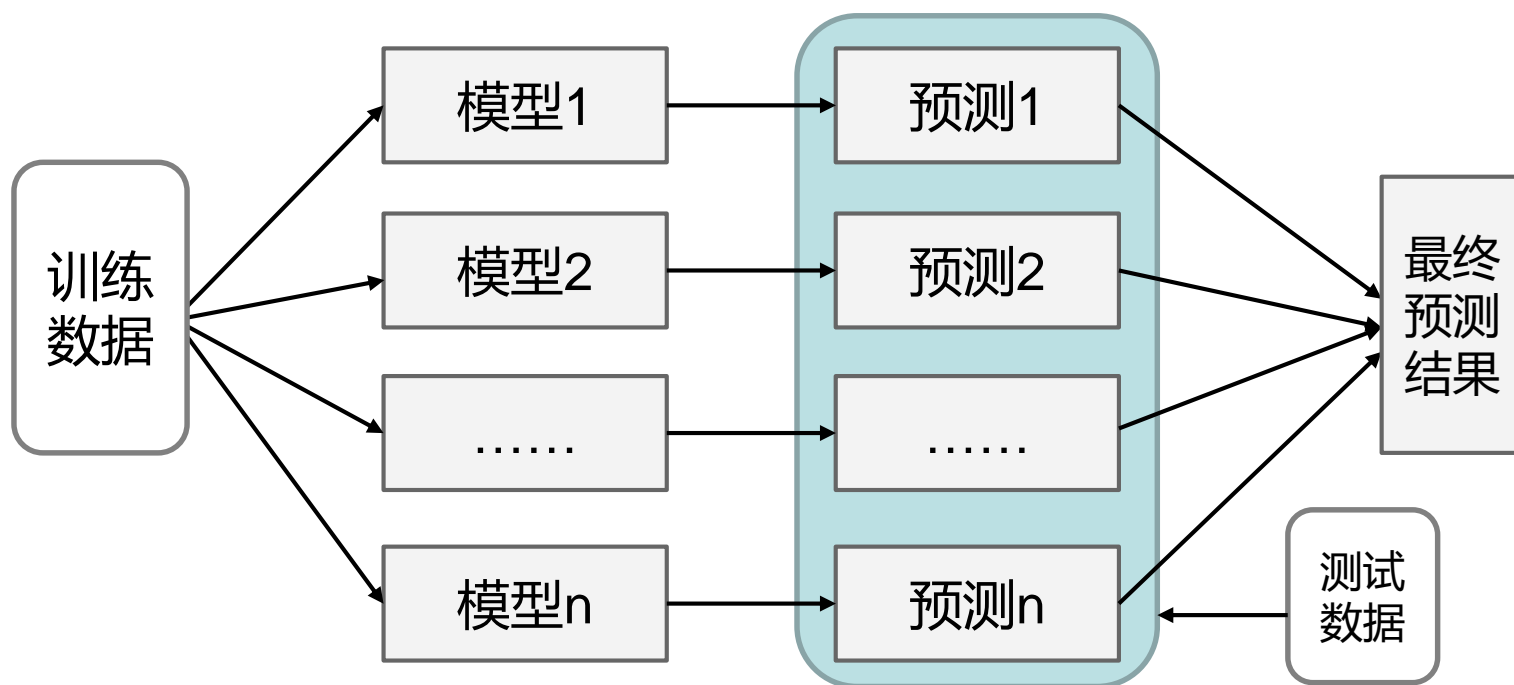
- Boosting : Adaboost、 GDBT、 XGBoost

2.集成学习方法概述

11

- Bagging (**B**ootstrap **a**ggregating , 自举汇聚算法)

- 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：



2.集成学习方法概述

12

• Bagging算法

输入: 训练集 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
基学习算法 \mathcal{L} ;
训练轮数 T .

过程:

1: **for** $t = 1, 2, \dots, T$ **do**
2: $h_t = \mathcal{L}(D, \mathcal{D}_{bs})$
3: **end for**

训练T个基学习器

输出: $H(\mathbf{x}) = \arg \max_{y \in \mathcal{Y}} \sum_{t=1}^T \mathbb{I}(h_t(\mathbf{x}) = y)$

对T个基学习器进行结合

2.集成学习方法概述

13

- **Bagging算法的特点**

- Bagging通过降低基学习器的方差，改善了泛化误差；
- 其性能依赖于基学习器的稳定性；
- 由于每个样本被选中的概率相同，因此Bagging并不侧重于训练数据集中的任何特定实例；
- 时间复杂度低：假定基学习器的计算复杂度为 $O(m)$ ，采样与投票/平均过程的复杂度为 $O(s)$ ，则Bagging的复杂度大致为 $T(O(m) + O(s))$ 。

2.集成学习方法概述

14

- **随机森林 (Random Forest , RF)**

- 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。

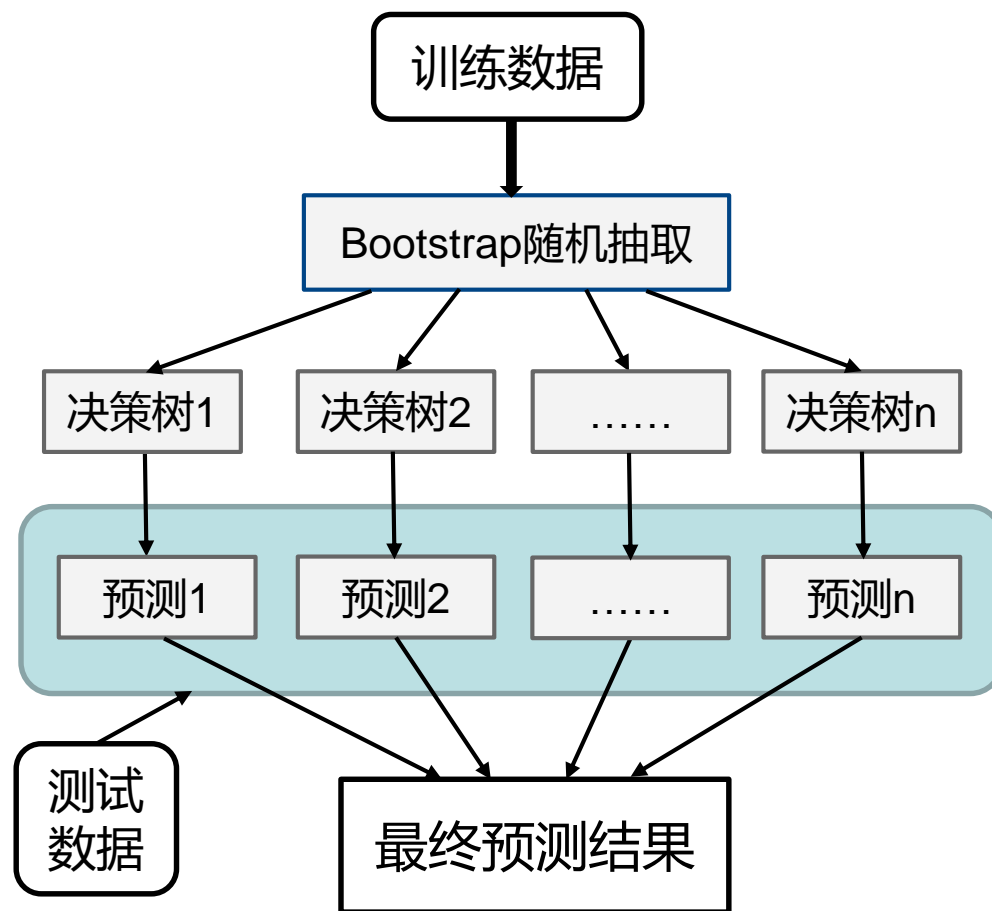
2.集成学习方法概述

15

• 随机森林 (Random Forest)

➤ 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，随机森林包括四个部分：

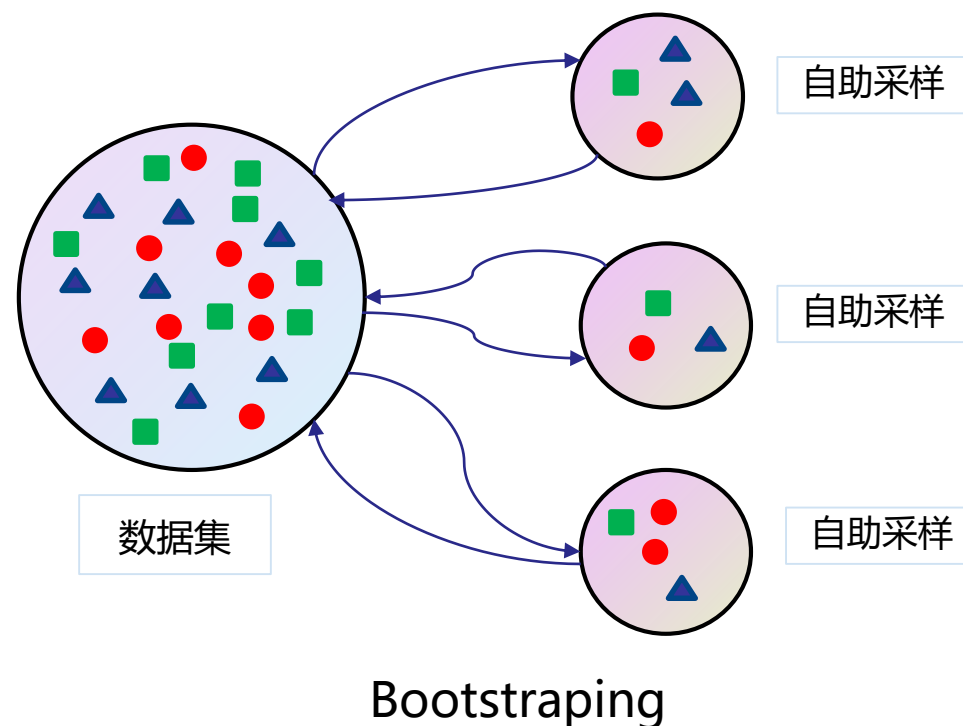
1. 随机选择样本（放回抽样）；
2. 随机选择特征；
3. 构建决策树；
4. 随机森林投票（平均）。



2.集成学习方法概述

16

- 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。

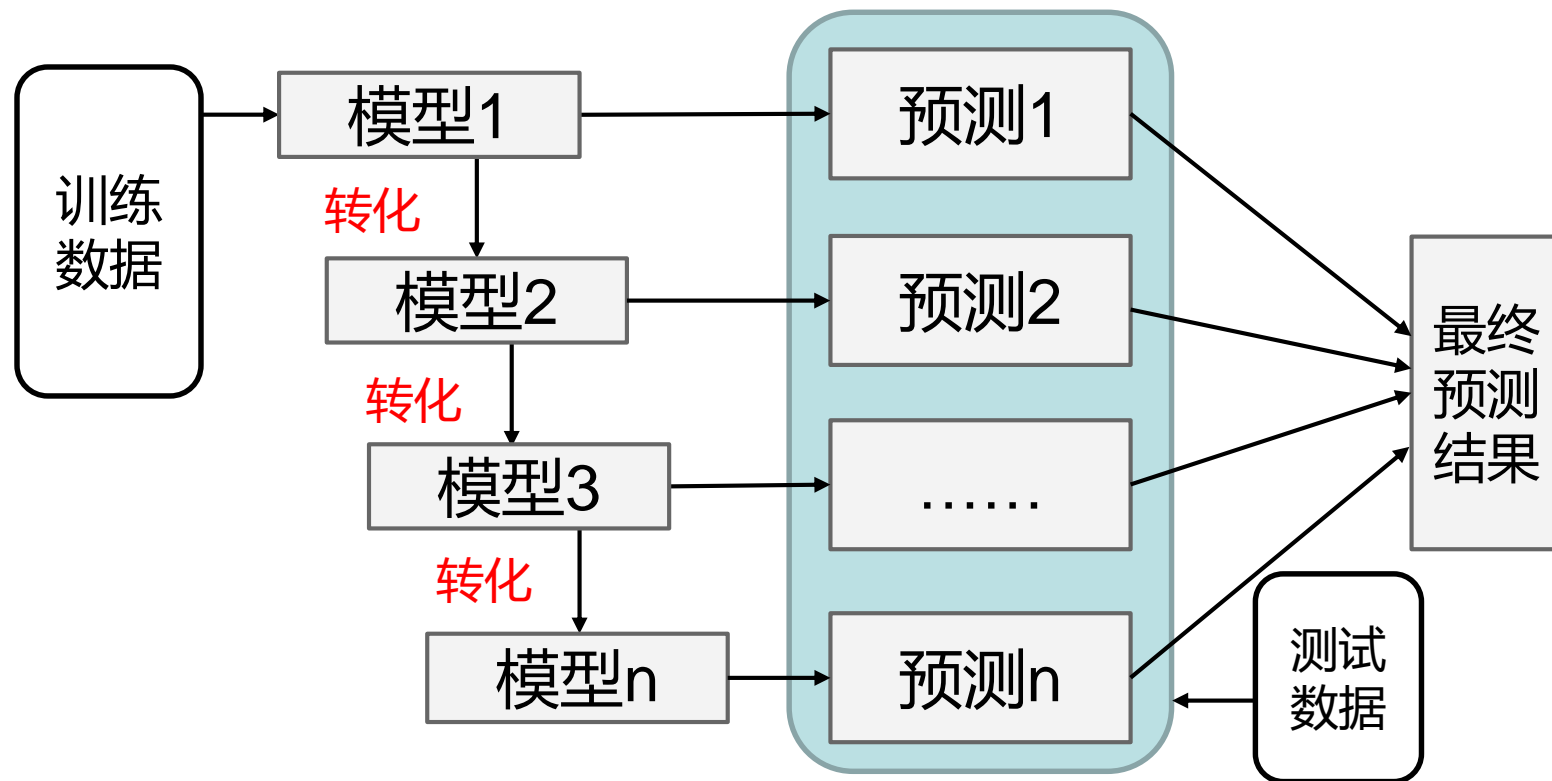


2.集成学习方法概述

17

• Boosting

- 训练过程为阶梯状；
- 基模型按次序——进行训练（实现上可以做到并行）；
- 基模型的训练集按照某种策略每次都进行一定的转化；
- 对所有基模型预测的结果进行线性综合产生最终的预测结果。

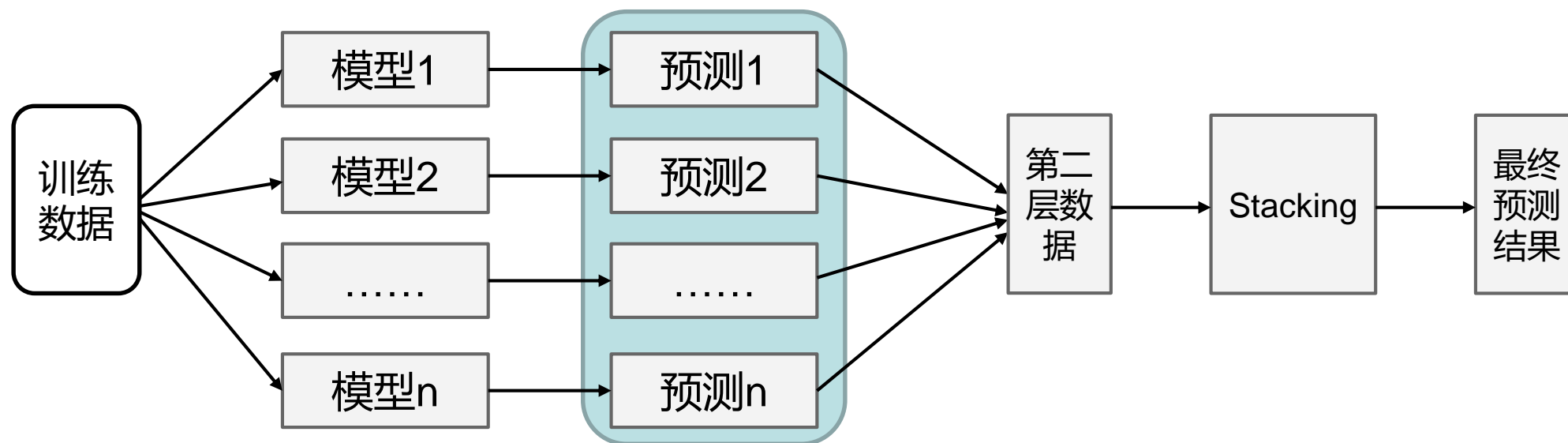


2.集成学习方法概述

18

• 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。



3.AdaBoost和GBDT算法

19

01 个体与集成

02 集成学习方法概述

03 AdaBoost和GBDT算法

04 XGBoost

05 LightGBM

3.AdaBoost和GBDT算法

20

• AdaBoost (Adaptive Boosting) 的起源

- 1984年，Kearns和Valiant提出强学习(strongly learnable)和弱学习(weakly learnable)：
 - 在概率近似正确 (probably approximately correct, PAC) 学习框架中，如果存在一个多项式的学习算法能够学习它，并且正确率很高，称这个概念是强学习的；
 - 如果存在一个多项式的学习算法能够学习它，学习的正确率仅比随机猜测略好，则称这个概念是弱学习的。

3.AdaBoost和GBDT算法

21

- **AdaBoost (Adaptive Boosting) 的起源**

- 问题的提出：

- 只要找到一个比随机猜测略好的弱学习算法就可以直接将其提升为强学习算法，而不必直接去找很难获得的强学习算法。

3.AdaBoost和GBDT算法

22

• 怎样实现弱学习转为强学习

- 例如：学习算法A在a情况下失效，学习算法B在b情况下失效，那么在a情况下可以用B算法，在b情况下可以用A算法解决。这说明通过某种合适的方式把各种算法组合起来，可以提高准确率。
- 为实现弱学习互补，面临两个问题：
 - 怎样获得不同的弱分类器？
 - 怎样组合弱分类器？

3.AdaBoost和GBDT算法

23

• 怎样获得不同的弱分类器

- 使用不同的弱学习算法得到不同的基学习器
 - 参数估计、非参数估计…
- 使用相同的弱学习算法，但用不同的参数
 - K-Mean不同的K，神经网络不同的隐含层…
- 使用不同的训练集
 - Bagging
 - Boosting

3.AdaBoost和GBDT算法

24

• 怎样组合弱分类器

➤ 多专家组合

– 一种并行结构，所有的弱分类器都给出各自的预测结果，通过“组合器”把这些预测结果转换为最终结果。例如，投票（voting）及其变种、混合专家模型。

➤ 多级组合

– K-Mean不同的K，神经网络不同的隐含层…一种串行结构，其中下一个分类器只在前一个分类器预测不够准的实例上进行训练或检测。例如，级联算法（cascading）。

3.AdaBoost和GBDT算法

25

- **AdaBoost的提出**

- 1990年，Schapire最先构造出一种多项式级的算法，即最初的Boost算法;
- 1993年，Drunker和Schapire第一次将神经网络作为弱学习器，应用Boosting算法解决OCR问题;
- 1995年，Freund和Schapire提出了Adaboost(Adaptive Boosting)算法，效率和原来Boosting算法一样，但是不需要任何关于弱学习器性能的先验知识，可以非常容易地应用到实际问题中。

3. AdaBoost和GBDT算法

26

- **AdaBoost**基本概念

AdaBoost

Adaptive Boosting

A learning algorithm

Building a strong classifier via a lot of weaker ones.

3. AdaBoost和GBDT算法

27

- AdaBoost基本概念

$$\left. \begin{array}{l} h_1(x) \in \{-1, +1\} \\ h_2(x) \in \{-1, +1\} \\ \vdots \\ h_T(x) \in \{-1, +1\} \end{array} \right\}$$

Weak classifiers

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

Strong classifier

3.AdaBoost和GBDT算法

28

• AdaBoost-两个问题

➤ 每一轮如何改变训练数据的权值或概率分布？

- AdaBoost：提高那些被前一轮弱分类器错误分类样本的权值，降低那些被正确分类样本的权值。

➤ 如何将弱分类器组合成一个强分类器？

- AdaBoost：加权多数表决，加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用，减小分类误差率大的弱分类器的权值，使其在表决中起较小的作用。

3.AdaBoost和GBDT算法

29

- **AdaBoost (Adaptive Boosting , 自适应增强)**

- 其自适应在于：前一个基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！

3.AdaBoost和GBDT算法

30

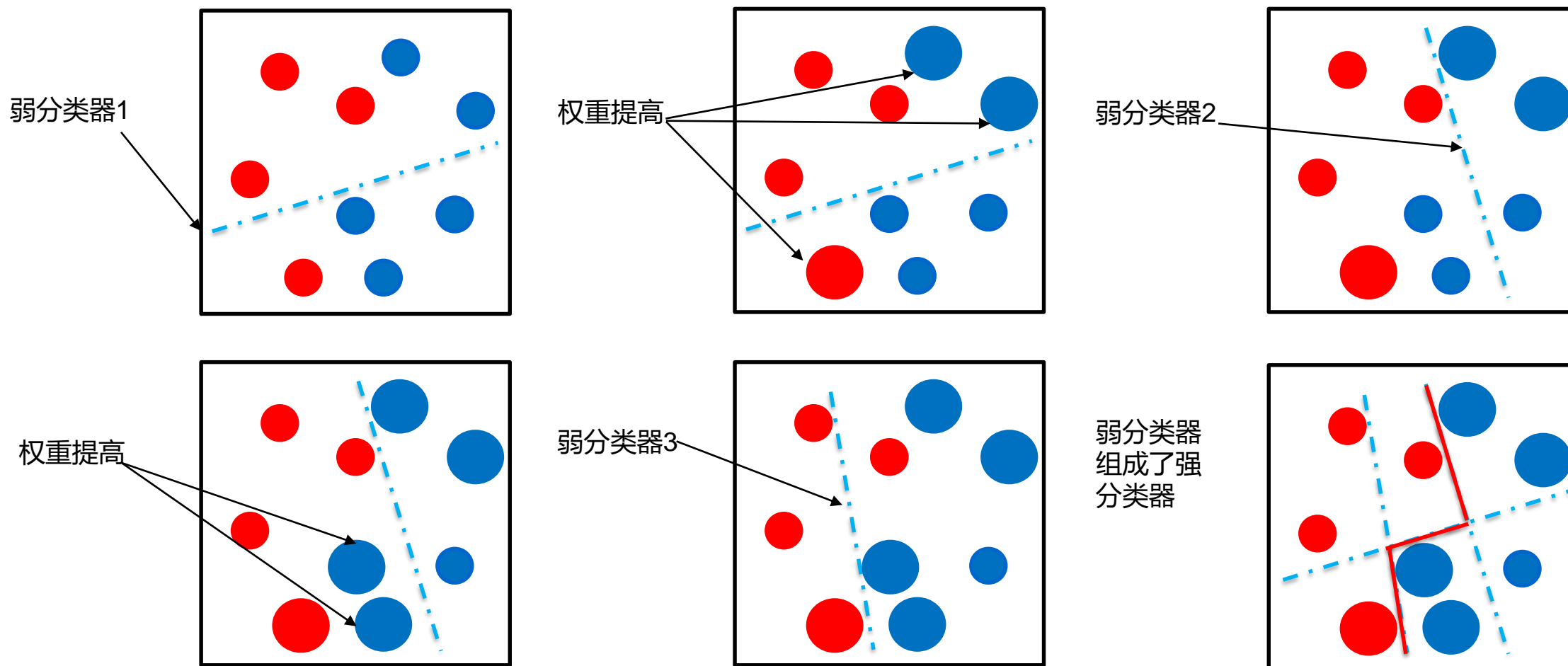
• AdaBoost算法思想

- 初始化训练样本的权值分布，使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时，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。

3.AdaBoost和GBDT算法

31

• AdaBoost算法示意图

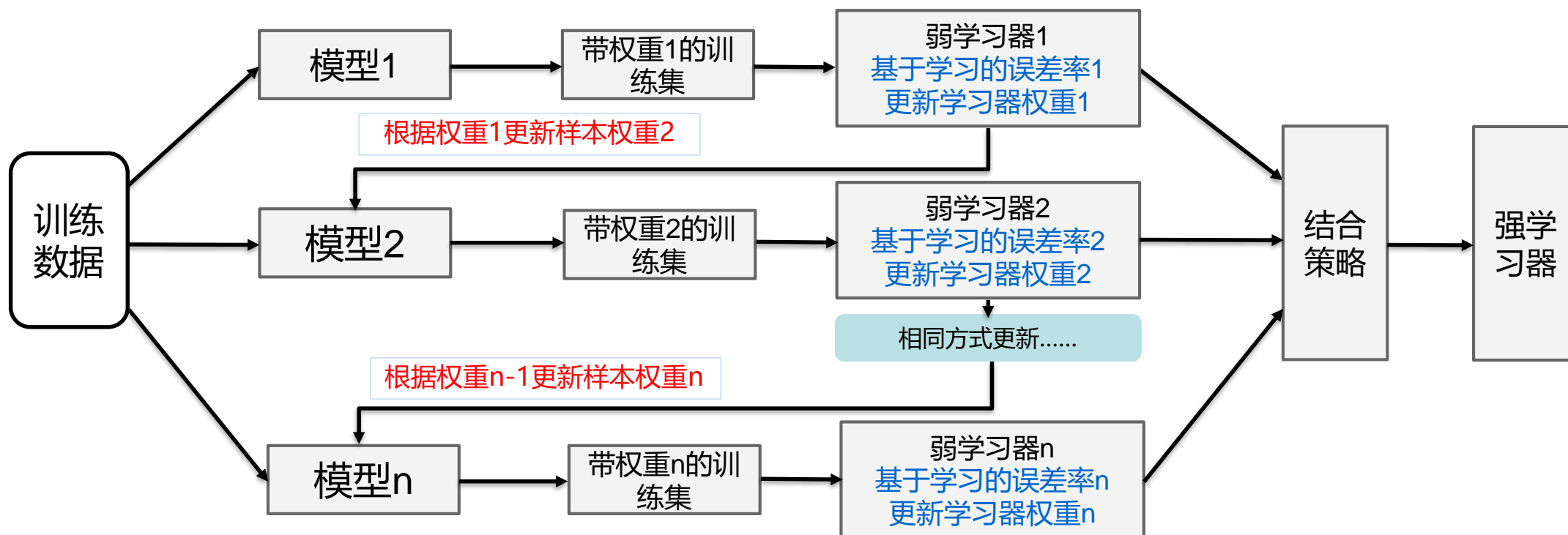


3.AdaBoost和GBDT算法

32

• AdaBoost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！



3.AdaBoost和GBDT算法

33

- **AdaBoost算法注意事项**

- 训练的每一轮都需要检查当前生成的基分类器是否比随机猜测好
- 对特定的数据分布进行学习
 - 重赋权法(**re-weighting**)
 - 重采样法(**re-sampling**)

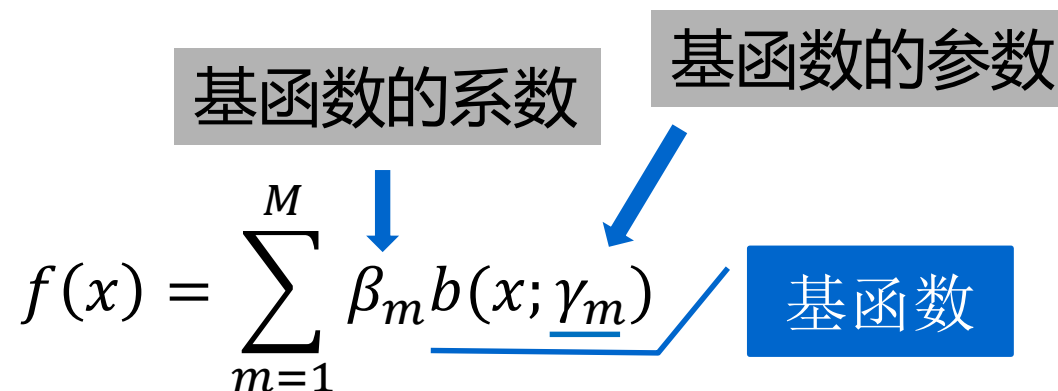
3.AdaBoost和GBDT算法

34

• 前向分步算法

➤ AdaBoost算法是前向分步算法的特例，是由基分类器组成的加法模型。

➤ 加法模型 (Additive Model)


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

➤ 给定训练数据和损失函数 $L(y, f(x))$, 加法模型 $f(x)$ 的优化问题转化为经验风险极小化，即损失函数极小化问题：

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

3.AdaBoost和GBDT算法

35

• 前向分步算法

从前向后，每一步只学习一个基函数及其系数，逐步逼近优化目标函数式。

- 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 $L(y, f(x))$ ，基函数集合 $b(x; \gamma)$
- 输出：加法模型 $f(x)$

1. 初始化 $f_0(x) = 0$

2. 对 $m = 1, 2, \dots, M$

极小化损失函数 $(\beta_m, \gamma_m) = \operatorname{argmin} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x; \gamma))$ ，得到参数 β_m, γ_m

更新 $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

3. 得到加法模型 $f(x) = f_M(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m)$

3.AdaBoost和GBDT算法

36

• 提升树 (Boosting Tree)

- 提升树是以分类树或回归树为基本分类器的提升方法；提升树被认为是统计学习中性能最好的方法之一。
- 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**，以**决策树**为基函数。

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

决策树的个数

决策树的参数

决策树

3.AdaBoost和GBDT算法

37

• 提升树算法

➤ 采用前向分步算法

1. 确定初始提升树 $f_0(x) = 0$;

2. 第 m 步模型： $f_m(x) = f_{m-1}(x) + T(x; \theta_m)$ ，其中， $f_{m-1}(x)$ 为当前模型，通过经验风险极小化确定下一棵决策树的参数 $\hat{\theta}_m = \operatorname{argmin} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x; \theta_m))$ 。

3.AdaBoost和GBDT算法

38

• 提升树算法

- 针对不同问题的提升树学习算法，使用的损失函数不同：
 - 用平方误差损失函数的回归问题，
 - 用指数损失函数的分类问题，
 - 用一般损失函数的一般决策问题（自定义）。
- 对二类分类问题：提升树算法只需将AdaBoost算法中的基本分类器限制为二类分类树即可。

3.AdaBoost和GBDT算法

39

• 梯度提升 (Gradient Boosting)

- 提升树利用加法模型与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，每一步优化是很简单的，但对一般损失函数而言，往往每一步优化并不容易。
- 针对这一问题，Freidmao提出了梯度提升算法。这是利用最速下降法的近似方法，其关键是利用损失函数的负梯度在当前模型的值作为回归问题提升树算法中的残差的近似值，拟合一个回归树，即：

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

3.AdaBoost和GBDT算法

40

- **GBDT (Gradient Boosting Decision Tree)**

- GBDT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)。

3.AdaBoost和GBDT算法

41

• GBDT算法

1. 初始化弱学习器 $f_0(x) = \operatorname{argmin} \sum_{i=1}^N L(y_i, c)$

2. 对 $m = 1, 2, \dots, M$

a. 对 $i = 1, 2, \dots, N$, 计算 $r_{mi} = - \left[\frac{\partial L(y, f(x_i))}{\partial f(x_i)} \right]_{f(x) = f_{m-1}(x)}$

b. 对 r_{mi} 拟合一个CART回归树 , 得到第 m 个弱学习器 , 对应的叶节点区域为 R_{mj}

c. 对 $j = 1, 2, \dots, J$, 计算 $c_{mj} = \operatorname{argmin} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)$

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

3. 得到强学习器 $\hat{f}(x) = f_M(x) = \sum_{m=1}^M \sum_{j=1}^J c_{mj} I(x \in R_{mj})$

3.AdaBoost和GBDT算法

42

• GBDT总结

优点：

- 适用范围广。一方面，对于各种类型的数据可以灵活处理（离散连续都可以）。另一方面，分类问题和回归问题都适用。
- 鲁棒性强。
- 适用于低维数据和非线性数据。
- 在相对较少的调参时间下，预测的准确度较好。

缺点：

- 弱学习器之间的依赖性强，需要一个一个训练，较为耗时。

4.XGBoost

43

01 个体与集成

02 集成学习方法概述

03 AdaBoost和GBDT算法

04 XGBoost

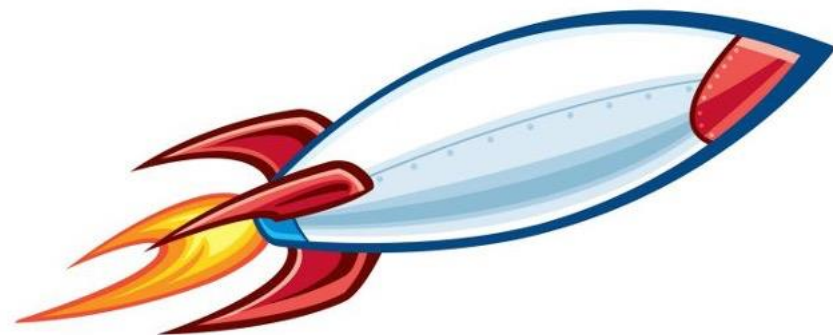
05 LightGBM

4.XGBoost

44

- **XGBoost (eXtreme Gradient Boosting)**

- 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。



4.XGBoost

45

- **XGBoost算法原理**

- 和GBDT一样，XGBoost是一个加法模型，在每一步迭代中只优化当前步中的子模型。在第 m 步中：

$$F_m(x_i) = F_{m-1}(x_i) + f_m(x_i)$$

- 其中， $f_m(x_i)$ 为当前步的子模型， $F_{m-1}(x_i)$ 为训练完已经固定的前 $m - 1$ 个子模型。

4.XGBoost

46

• XGBoost算法原理

➤ 目标函数：经验风险 + 正则项

$$\begin{aligned} Obj(t) &= \sum_{i=1}^N L(F_m(x_i), y_i) + \sum_{j=1}^m \Omega(f_j) \\ &= \sum_{i=1}^N L(F_{m-1}(x_i) + f(x_i), y_i) + \sum_{j=1}^m \Omega(f_j) \end{aligned}$$

➤ 其中，正则项 $\Omega(f)$ 表示子模型 f 的复杂度， $F_{m-1}(x_i)$ 为训练完已经固定的前 $m - 1$ 个子模型。

4.XGBoost

47

• XGBoost算法原理

➤ 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

$$Obj(t) = \sum_{i=1}^N L \left(F_{m-1}(x_i) + \frac{\partial L}{\partial F_{m-1}(x_i)} f_m(x_i) + \frac{1}{2} \frac{\partial^2 L}{\partial^2 F_{m-1}(x_i)} f_m^2(x_i) \right) + \sum_{j=1}^m \Omega(f_j)$$

➤ 当前 $m - 1$ 个子模型固定时, 除 $f_m(x_i)$ 外, 其余部分均为常数, 目标函数转化为:

$$Obj(t) = \sum_{i=1}^N L \left(\frac{\partial L}{\partial F_{m-1}(x_i)} f_m(x_i) + \frac{1}{2} \frac{\partial^2 L}{\partial^2 F_{m-1}(x_i)} f_m^2(x_i) \right) + \sum_{j=1}^m \Omega(f_j)$$

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。

4.XGBoost

48

• XGBoost算法原理

- XGBoost支持的基分类器包括决策树和线性模型，以基于树的情况为例。为防止过拟合，XGBoost设置了基于树的复杂度作为正则项：

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|\omega\|^2$$

- 其中， T 为树 f 的叶节点个数， ω 为所有叶节点输出回归值构成的向量， $\|\omega\|^2$ 为该向量的 L_2 范数， γ ， λ 为超参数，即：

$$Obj(t) = \sum_{i=1}^N L \left(\frac{\partial L}{\partial F_{m-1}(x_i)} f_m(x_i) + \frac{1}{2} \frac{\partial^2 L}{\partial^2 F_{m-1}(x_i)} f_m^2(x_i) \right) + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T \omega_j^2$$

4.XGBoost

49

• XGBoost算法原理

- 定义节点 j 上的样本集为 $I(j) = \{x_i | q(x_i) = j\}$ ，其中 $q(x_i)$ 为将样本映射到叶节点上的索引函数，叶节点 j 上的回归值为 $\omega_j = f_m(x_i)$ ， $i \in I(j)$ ：

$$Obj(t) = \sum_{i=1}^T \left(\left(\sum_{i \in I(j)} \frac{\partial L}{\partial F_{m-1}(x_i)} f_m(x_i) \right) \omega_j \right) + \frac{1}{2} \left(\frac{\partial^2 L}{\partial^2 F_{m-1}(x_i)} f_m^2(x_i) + \lambda \right) + \omega_j^2 + \gamma T$$

4.XGBoost

50

• XGBoost算法原理

➤ 令： $g_i = \frac{\partial L}{\partial F_{m-1}(x_i)}$ ， $h_i = \frac{\partial^2 L}{\partial^2 F_{m-1}(x_i)}$ ， $G_j = \sum_{i \in I(j)} g_i$ ， $H_j = \sum_{i \in I(j)} h_i$ ，则上式可表示为：

$$Obj = \sum_{i=1}^T (G_j \omega_j + \frac{1}{2} (H_j + \lambda) \omega_j^2) + \gamma T$$

➤ 此时，若树的结构已经确定，则各个节点内的样本 (x_i, y_i, g_i, h_i) 是确定的，每个叶节点输出的回归值应该使得上式最小，由二次函数极值点，可得：

$$\omega_j^* = -\frac{G_j}{H_j + \lambda}$$

4.XGBoost

51

• XGBoost算法原理






➤ 按此规则输出回归值 ω_j^* 后，目标函数值，即树的评分如下：

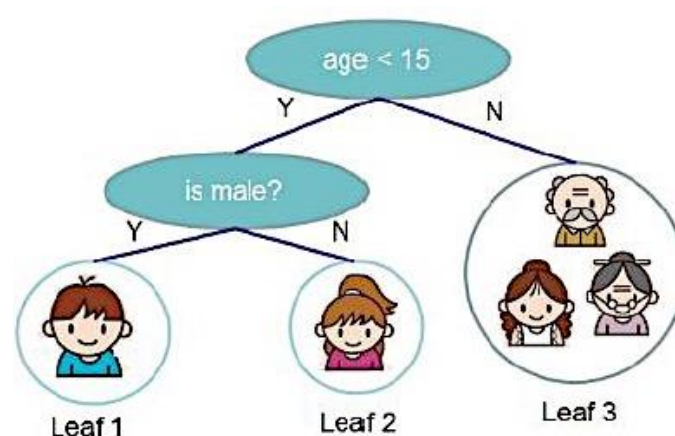
$$Obj^* = \sum_{i=1}^T \left(-\frac{G_j^2}{2(H_j + \lambda)} + \gamma \right)$$

➤ 观察上式，树的评分也可以理解成所有叶节点的评分之和，值越小代表树的结构越好。

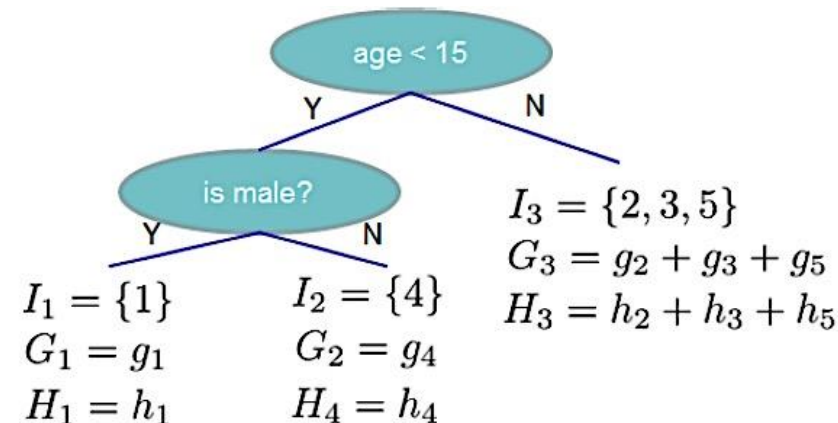
4.XGBoost

52

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

4.XGBoost

53

• XGBoost的分裂准则

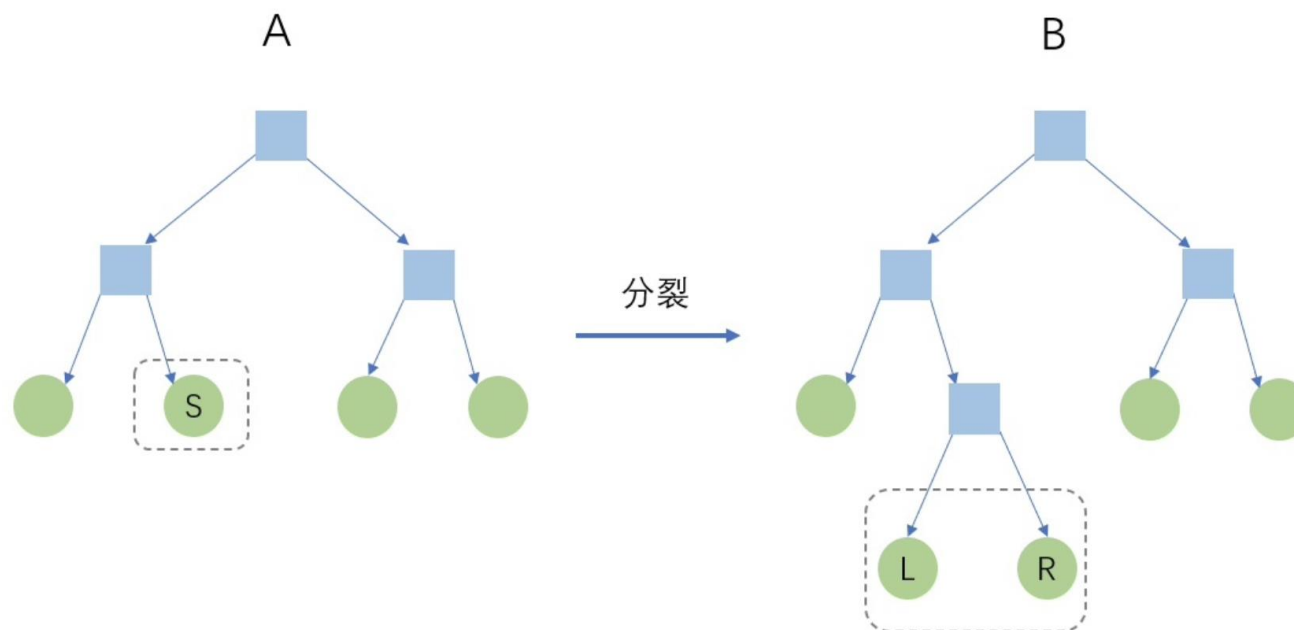
- XGBoost的子模型树和决策树模型一样，要依赖节点递归分裂的贪心准则来实现树的生成。除此外，XGBoost还支持近似算法，解决数据量过大超过内存、或有并行计算需求的情况。
- 使用贪心准则，选增益（*gain*）最大的分裂方式。
- 贪心准则，众多*gain*中找到最大值做为最优分割节点（split point），因此模型会将所有样本按照（一阶梯度）从小到大排序，通过遍历，查看每个节点是否需要分割，计算复杂度是：决策树叶子节点数 - 1。

4.XGBoost

54

• 贪心准则

- 基本思路和CART一样，对特征值排序后遍历划分点，将其中最优的分裂收益作为该特征的分裂收益，选取具有最优分裂收益的特征作为当前节点的划分特征，按其最优划分点进行二叉划分，得到左右子树。



4.XGBoost

55

• 贪心准则

- 分裂收益：树A的评分减去树B的评分。虚线框外的叶节点，即非分裂节点的评分均被抵消，只留下分裂后的L、R节点和分裂前的S节点进行比较，因此分裂收益的表达式为：

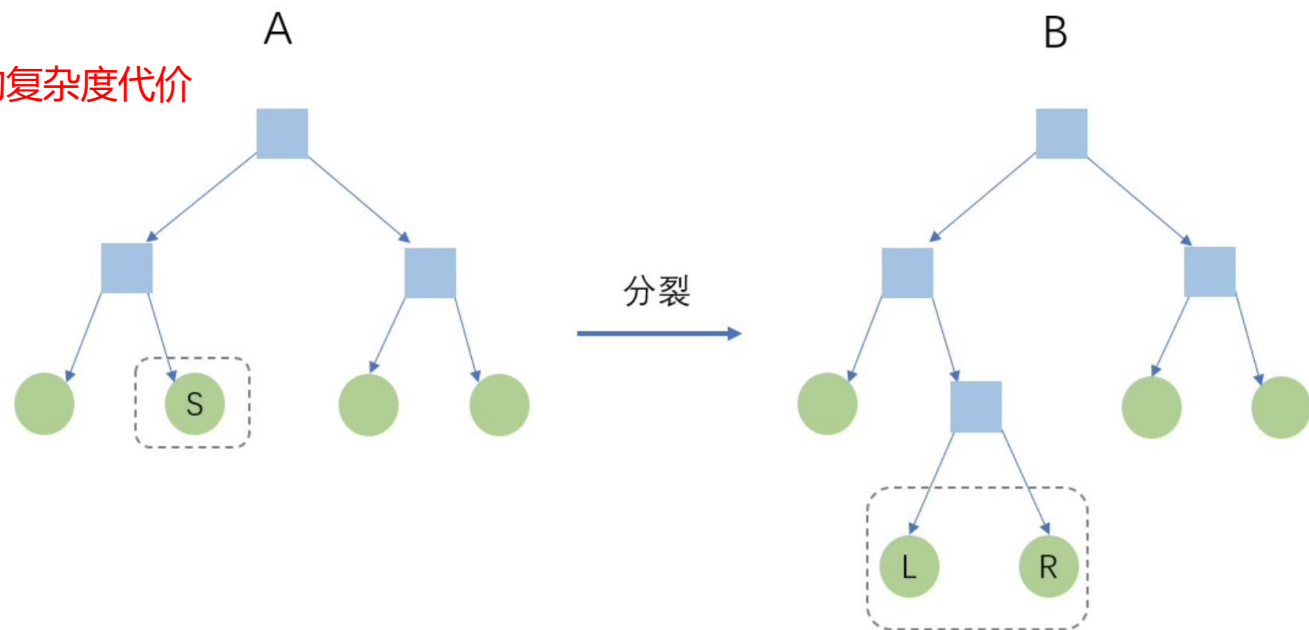
$$Gain = Gain(before) - Gain(after)$$
$$= \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$$

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

加入新叶子节点引入的复杂度代价



4.XGBoost

56

- **列采样和学习率**

- XGBoost还引入了两项特性：列采样和学习率。
- 列采样，即随机森林中的做法，每次节点分裂的待选特征集合不是剩余全部特征，而是剩余特征的一个子集。可以更好地对抗过拟合，还能减少计算开销。
- 学习率，或者叫步长，是在每个子模型前（即在每个叶节点的回归值上）乘上该系数，削弱每颗树的影响，XGBoost默认设定为0.3。

4.XGBoost

57

- **稀疏感知**

- 缺失值或是特征稀疏问题，如部分特征中出现大量的0或是one-hot向量这种情况。XGBoost用稀疏感知策略来同时处理这两个问题：概括地说，将缺失值和稀疏0值等同视作缺失值，再将这些缺失值“绑定”在一起，分裂节点的遍历会跳过缺失值的整体。这样大大提高了运算效率。

5.LightGBM

58

01 个体与集成

02 集成学习方法概述

03 AdaBoost和GBDT算法

04 XGBoost

05 LightGBM

5.LightGBM

59

• LightGBM的起源

- 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) 更少的样本
 - 2) 更快的训练速度
 - 3) 更低的内存消耗
 - 4) 分布式支持，可快速处理海量数据

5.LightGBM

60

- **LightGBM的主要改进**

LightGBM与XGBoost相比，主要有以下几个改进：

- 直方图算法（Histogram）；
- 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）；
- 基于最大深度的 Leaf-wise 的垂直生长算法；

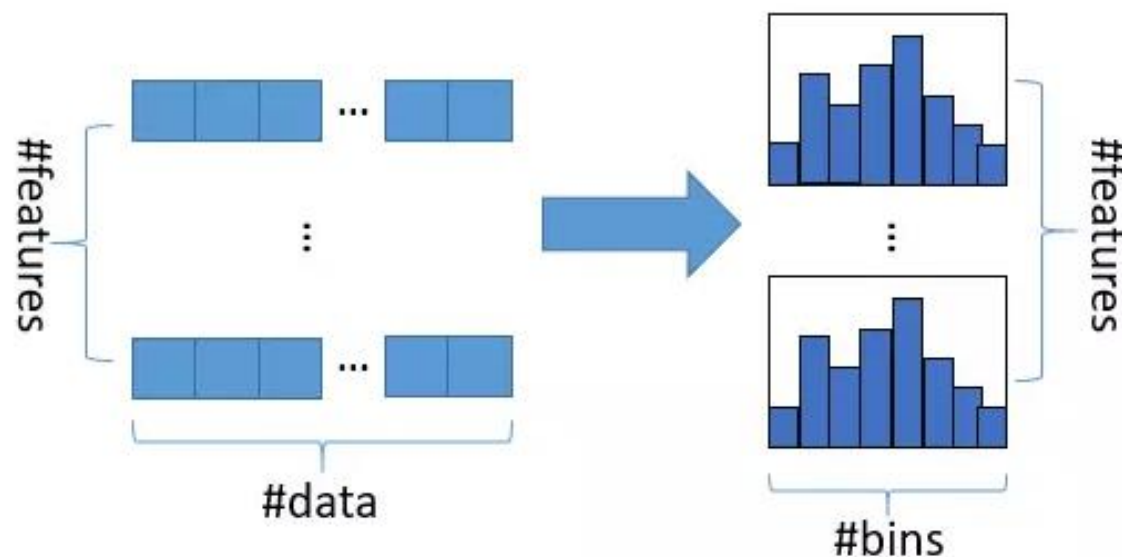
LightGBM = XGBoost + Histogram + GOSS + EFB

5.LightGBM

61

• 直方图算法 (Histogram)

- 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。



5.LightGBM

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

5.LightGBM

63

• 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。



5.LightGBM

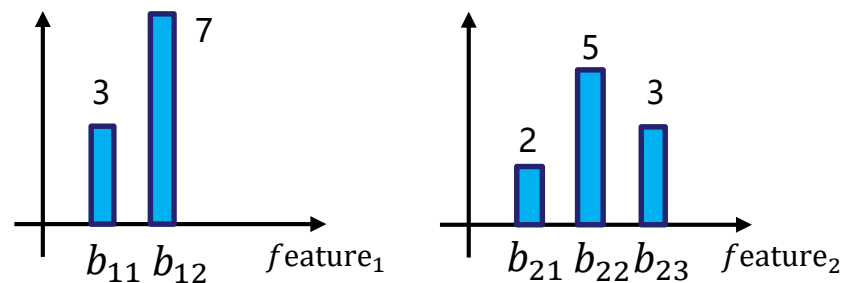
64

• 直方图算法加速

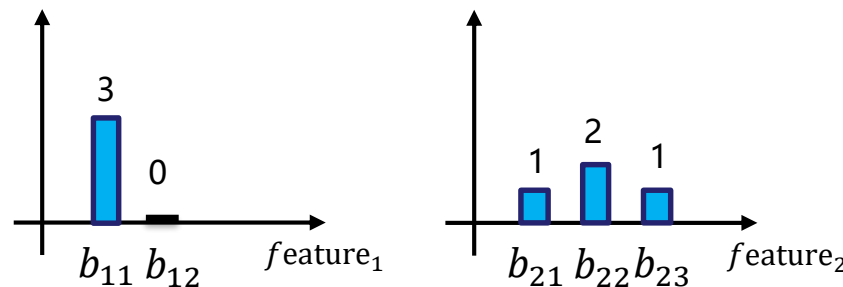
- 直方图算法还可以进一步加速：一个叶子节点的直方图可以由它的父亲节点的直方图与其兄弟的直方图做差得到。

父节点10个样本，2个特征。

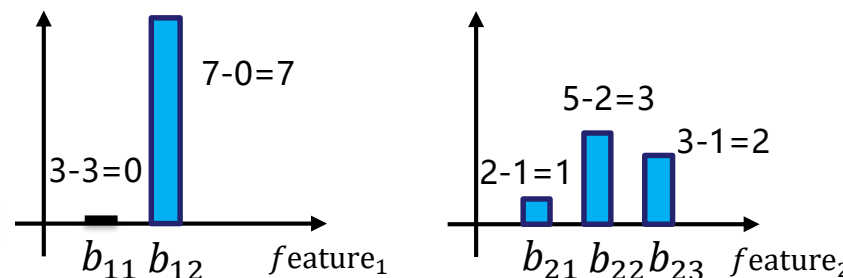
①父节点特征的直方图



②左子节点特征的直方图



③右子节点特征的直方图



5.LightGBM

65

• 直方图算法 (Histogram)

优点

1. 内存消耗降低。直方图算法仅需要存储特征的 bin 值(离散化后的数值)，不需要原始的特征值，也不用排序，而bin值用8位整型存储就足够了。
2. 算法时间复杂度大大降低。

缺点

1. 特征被离散化后，找到的并不是很精确的分割点，可能会对结果产生影响。

5.LightGBM

66

- **基于梯度的单边采样算法 (Gradient-based One-Side Sampling, GOSS)**
 - 单边梯度采样是一种在减少数据量和保证精度上平衡的算法。在GBDT中，对损失函数的负梯度进行拟合，样本误差越大，梯度的绝对值越大，这说明模型对该样本的学习还不足够，相反如果越小则表示模型对该样本的学习已经很充分。
 - 因此，可以这样推论：**梯度的绝对值越大，样本重要性越高**。单边梯度采样正是以样本的梯度作为样本的权重进行采样。

5.LightGBM

67

- **基于梯度的单边采样算法 (Gradient-based One-Side Sampling, GOSS)**
 - 主要思想是通过**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)。

5.LightGBM

68

• 基于梯度的单边采样算法

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 ，损失函数和弱学习器的类型（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。

5.LightGBM

69

- **互斥特征捆绑算法 (Exclusive Feature Bundling, EFB)**
 - 高维的数据通常是稀疏的，这种特征空间的稀疏性给我们提供了一种设计一种接近无损地降维的可能性。
 - 特别的，在稀疏特征空间中，许多特征是互斥的，换句话说，大部分特征不会同时取非0值，例如One-hot之后的类别特征他们从不同时为非0值。我们可以合并互斥的特征为单一特征，这个过程称为Exclusive Feature Bundling

5.LightGBM

70

- **互斥特征捆绑算法 (Exclusive Feature Bundling, EFB)**
 - 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。
 - 那么接下来有两个问题需要处理：
 - 需要合并哪些特征
 - 如何合并这些特征

5.LightGBM

71

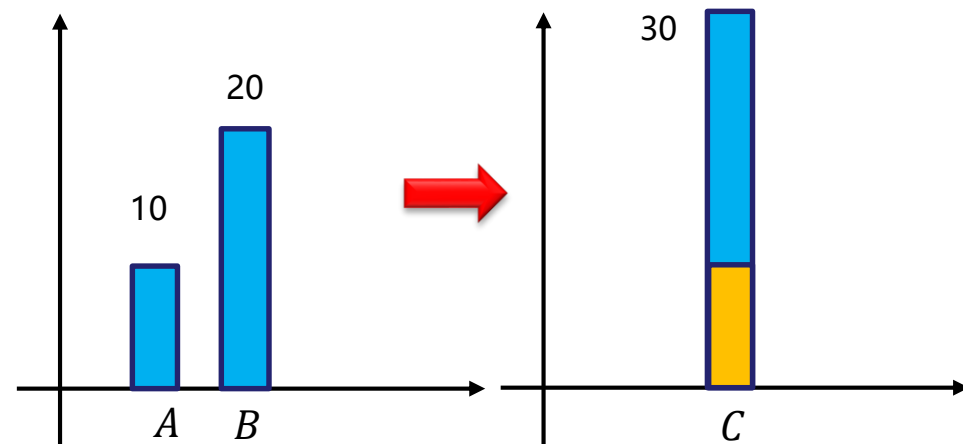
- **互斥特征捆绑算法 (Exclusive Feature Bundling, EFB)**
 - 需要合并哪些特征-Greedy Bundling找出最优的 bundle 组合数是一个 NP 问题，LightGBM通过将原问题转化为“图着色”问题进行贪心近似解决。
 - 首先创建一个图 $G(V, E)$ ，其中 V 就是特征，为图中的节点， E 为 G 中的边，将不是相互独立的特征用一条边连接起来，边的权重就是两个相连接的特征的总冲突值，这样需要绑定的特征就是在图着色问题中要涂上同一种颜色的那些点，即特征。

5.LightGBM

72

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

- 如何合并这些特征-Merge Exclusive Features
- 该过程主要是关键在于原始特征值可以从bundle中区分出来，即绑定几个特征在同一个bundle里需要保证绑定前的原始特征的值在bundle中能够被识别，这可以通过在特征值中加一个偏置常量来解决。

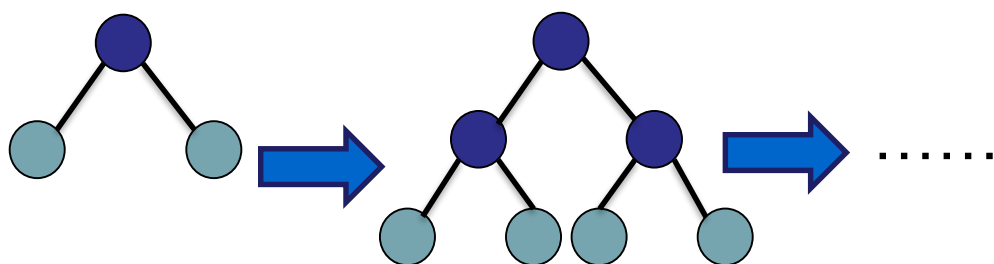


假设 Bundle中有两个特征值，A取值为 $[0,10]$ ，B取值为 $[0,20]$ ，为了保证特征A、B的互斥性，我们可以给特征B添加一个偏移量转换为 $C[10,30]$ ，Bundle后的特征其取值为 $[0,30]$ ，这样便实现了特征合并。

5.LightGBM

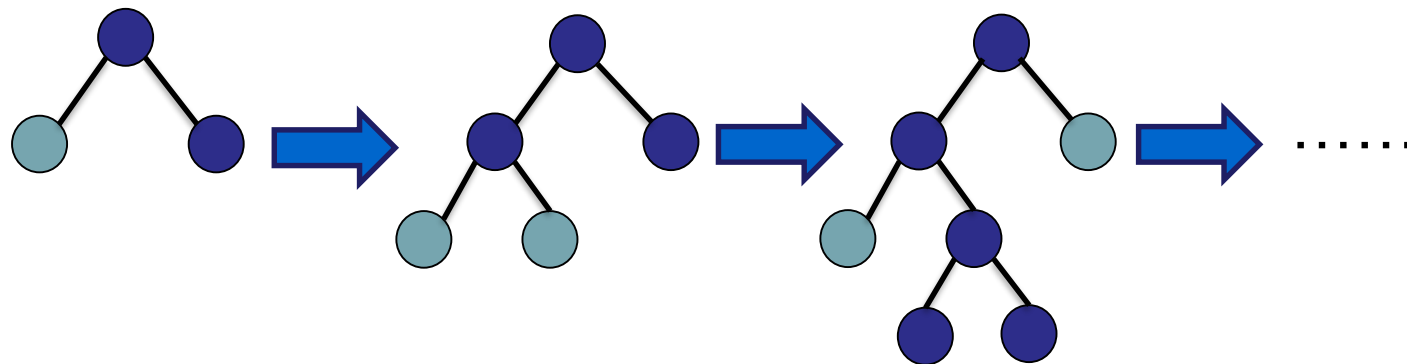
73

• 建树过程的两种方法：Leaf-wise VS Level-wise



Level-wise tree growth

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。



Leaf-wise tree growth

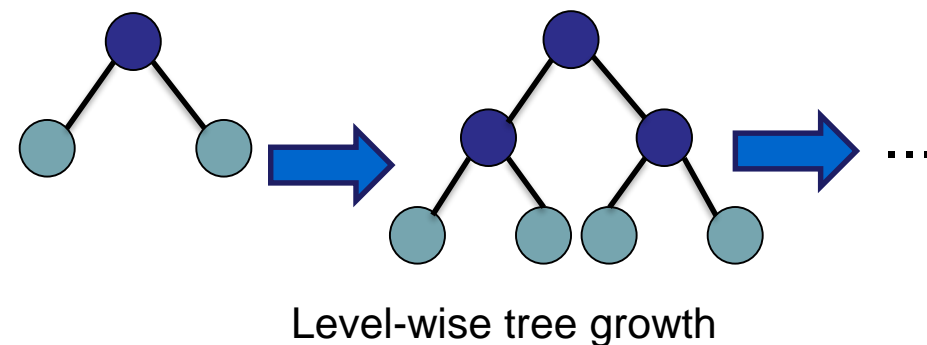
LightGBM 通过 leaf-wise (best-first)策略来生长树。它将选取具有最大 delta loss 的叶节点来生长。

5.LightGBM

74

• Level-wise

- 大多数GBDT框架使用的按层生长 (level-wise) 的决策树生长策略，Level-wise遍历一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度，不容易过拟合。
- 但实际上Level-wise是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多不必要的开销，因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

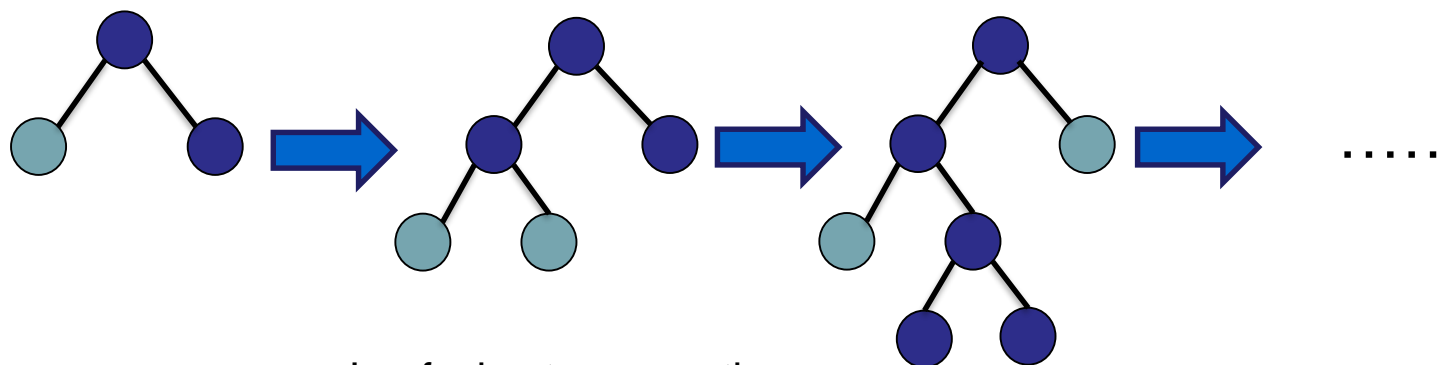


5.LightGBM

75

• Leaf-wise

- Leaf-wise则是一种更为高效的策略，每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。同Level-wise相比，在分裂次数相同的情况下，Leaf-wise可以降低更多的误差，得到更好的精度。Leaf-wise的缺点是可能会长出比较深的决策树，产生过拟合。因此LightGBM在Leaf-wise之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。



Leaf-wise tree growth

5.LightGBM

76

- **LightGBM并行计算**

- LightGBM具有支持高效并行的优点。
- LightGBM原生支持并行学习，目前支持：
 - 特征并行
 - 数据并行

5.LightGBM

77

• 特征并行

- 特征并行主要是并行化决策树中寻找最优划分点（Find Best Split）的过程。
- 传统特征并行做法如下：
 - 1.垂直划分数据（对特征划分），不同的worker（工作节点）有不同的特征集；
 - 2.每个workers找到局部最佳的切分点{feature, threshold}；
 - 3.workers使用点对点通信，找到全局最佳切分点；
 - 4.具有全局最佳切分点的worker进行节点分裂，然后广播切分后的结果（左右子树的instance indices）；
 - 5.其它worker根据收到的instance indices也进行划分。

5.LightGBM

78

- **数据并行**

- 数据并行的目标是并行化整个决策学习的过程。
- 传统数据并行做法如下：
 - 1.水平切分数据，不同的worker拥有部分数据；
 - 2.每个worker根据本地数据构建局部直方图；
 - 3.合并所有的局部直方图得到全部直方图；
 - 4.根据全局直方图找到最优切分点并进行分裂。

- [1] 李航. 统计学习方法[M]. 清华大学出版社,2019.
- [2] 周志华. 机器学习[M]. 清华大学出版社,2016.
- [3] Quinlan J R . Bagging, Boosting, and C4.5[C]// Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1. 1996.
- [4] BREIMAN L. Random forests[J]. Machine learning, 2001, 45(1): 5–32.
- [5] Ridgeway G . Special Invited Paper. Additive Logistic Regression: A Statistical View of Boosting: Discussion[J]. Annals of Statistics, 2000, 28(2):393-400.
- [6] FRIEDMAN J H . Stochastic gradient boosting[J]. Computational Statistics & Data Analysis, 2002, 38.

- [7] FRIEDMAN J H. Greedy function approximation: A gradient boosting machine[J]. Annals of statistics, 2001: 1189–1232.
- [8] MACQUEEN J, OTHERS. Some methods for classification and analysis of multivariate observations[C]//Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. Oakland, CA, USA, 1(14): 281–297.
- [9] CHEN T, GUESTRIN C. XGBoost:A Scalable Tree Boosting System[C]//Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM: 785–794.
- [10] NIELSEN Didrik. Tree Boosting With XGBoost - Why Does XGBoost Win “Every” Machine Learning Competition?[J], 2016.
- [11] Polyak B T . Newton's method and its use in optimization[J]. European Journal of Operational Research, 2007, 181(3):1086-1096.
- [12] KE G, MENG Q, FINLEY T, et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree[J]. 2017: 9.



谢谢！