

# 第7章

## 集成学习

### Ensemble Learning

孟 高 峰

[gfmeng@nlpr.ia.ac.cn](mailto:gfmeng@nlpr.ia.ac.cn)

<http://www.escience.cn/people/gfmeng/index.html>

时空数据分析与学习课题组 (STDAL)

中科院自动化研究所 模式识别国家重点实验室

助教: 方深 ([shen.fang@nlpr.ia.ac.cn](mailto:shen.fang@nlpr.ia.ac.cn))

# 内容提要

- 个体与集成
- Bagging
  - 特例：随机森林(Random Forest)
- AdaBoosting
- Gradient Boosting
  - 特例：提升树(Boosting Tree)
- 偏差-方差分解

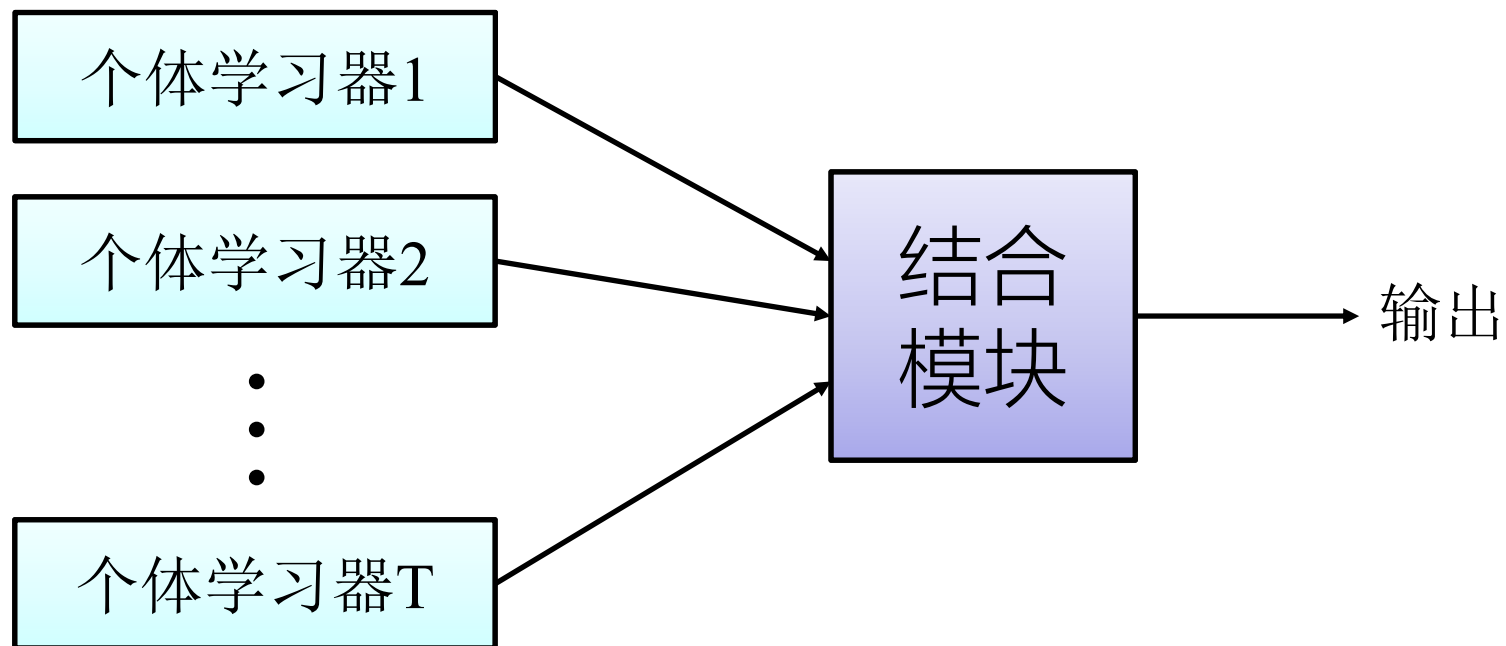
# 个体与集成

- 集成学习
  - “三个臭皮匠，顶个诸葛亮”
  - “不积跬步，无以至千里”
- 什么是集成学习
  - 通过构建并结合多个学习器来完成学习任务，又称多分类器系统（multi-classifier system）、基于委员会的学习(committee-based learning)等
- 将多个学习器进行结合，常可获得比单一学习器显著优越的泛化性能

## 三个臭皮匠，顶个诸葛亮

“皮匠”实际上是“裨将”的谐音，“裨将”在古代是指“副将”，原意是指三个副将的智慧合起来能顶一个诸葛亮。流传中，人们将“裨将”说成了“皮匠”。

# 集成学习示意图



# 一个简单例子

(1)

	测试例1	测试例2	测试例3
$h_1$	✓	✓	✗
$h_2$	✗	✓	✓
$h_3$	✓	✗	✓
集成	✓	✓	✓

集成提升性能

(2) 集成个体应“好而不同”  
准确性 vs 多样性

(3)

	测试例1	测试例2	测试例3
$h_1$	✓	✗	✗
$h_2$	✗	✓	✗
$h_3$	✗	✗	✓
集成	✗	✗	✗

集成起负作用

# 例子分析

- 考虑二分类问题

$$y \in \{-1, +1\} \quad \text{真实函数 } f$$

- 假定基分类器错误率为  $\varepsilon$ ，即：

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

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

$$H(x) = \text{sign} \left( \sum_{i=1}^T h_i(x) \right)$$

# 例子分析

- 假设基分类器的错误率相互独立，则集成的错误率可计算如下：

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor \frac{T}{2} \rfloor} \binom{T}{k} (1-\varepsilon)^k \varepsilon^{T-k}$$
$$\leq \exp\left(-\frac{1}{2}T(1-2\varepsilon)^2\right)$$

由Hoeffding不等式可得

## Remarks:

- 随着集成的个体分类器数目T的增大，集成错误率呈指数级下降，最终趋于零；
- 基学习器的误差独立性假设在实际中不满足！
- 基学习器的“准确性”与“多样性”是一对矛盾！

# 集成学习方法分类

- 根据个体学习器的生成方式，集成学习方法分为两大类

Bagging

个体学习器之间不存在强依赖关系、可同时生成的并行化方法

Boosting

个体学习器之间存在强依赖关系、必须串行生成的序列化方法



# Bagging算法

# Bagging

- 为获得好的集成，要求：
  - 多样性要求：个体学习器应尽可能独立；
    - 实际中无法做到独立，但可设法使基学习器尽可能具有较大差异。
  - 准确性要求：个体学习器不能太差；
    - 可使用相互有交叠的采样子集
- Bagging是并行式集成学习方法最著名的代表
- Bagging是Bootstrap aggregating的缩写，采用自助采样法（bootstrap sampling）来构造基学习器

# 自助采样法(bootstrap sampling)

- 自助采样法是一种可重复采样方法
- 从给定的数据集中通过有放回式抽样，产生一个或多个不同新数据集的方法；
- 给定包含 $m$ 个样本的数据集 $D$ ，对其采样产生数据集 $D_{bs}$ ：

1. 每次随机从 $D$ 中挑选一个样本，将其拷贝放入 $D_{bs}$ 中，使得该样本在下次采样时仍有可能被采到；
2. 重复上述过程 $m$ 次，可得到包含 $m$ 个样本的数据集 $D_{bs}$ ；

$$\lim_{m \rightarrow \infty} \left(1 - \frac{1}{m}\right)^m \mapsto \frac{1}{e} \approx 0.368$$

样本不出现在  
 $D_{bs}$ 的概率

# Bagging算法

- 基本流程：
  - 采样出 $T$ 个含 $m$ 个训练样本的采样集；
  - 基于每个采样集训练一个基学习器；
  - 通过简单投票法(平均法)将这些基学习器进行结合；

输入：训练集  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$

基学习算法  $\mathcal{L}$

训练轮数  $T$

过程：

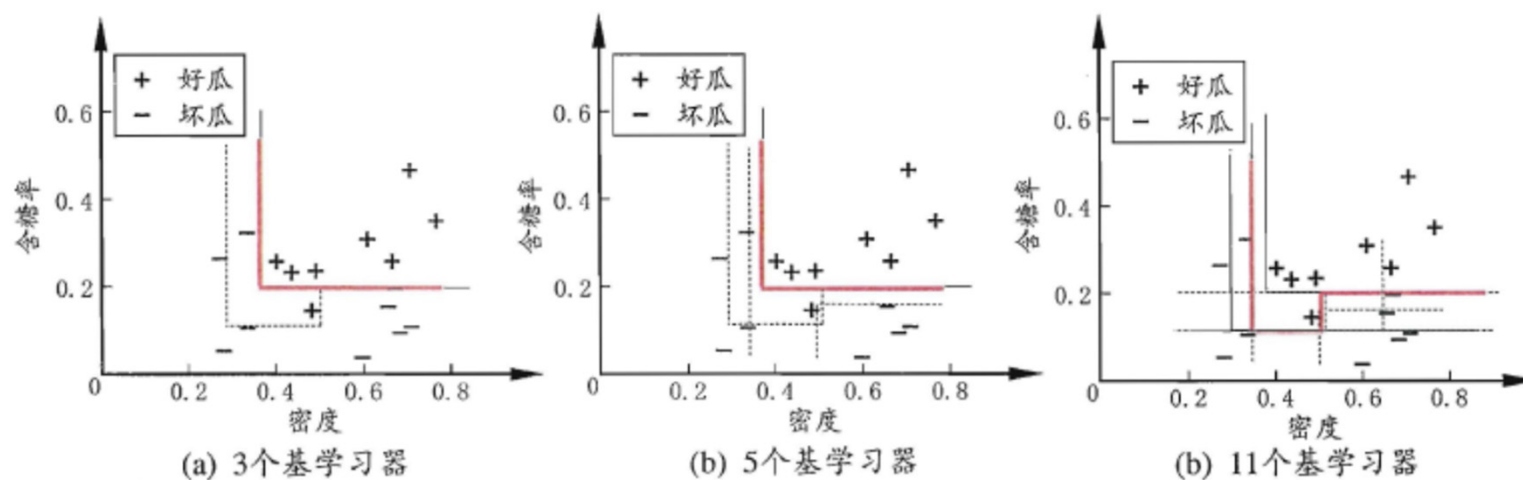
1. **for**  $t = 1, 2, \dots, T$  **do**

2.  $h_t = \mathcal{L}(D, D_{bs})$

3. **end for**

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

# 一个例子



基学习器：采用基于信息增益的决策树  
红色折线：分类边界  
虚线：基分类器

# Bagging算法分析

- 计算复杂度与训练基学习器的复杂度同阶；
- Bagging可直接用于多分类、回归等任务；
- 自助采样剩余的样本可用作验证集进行“包外估计”；
  - 决策树剪枝；
  - 神经网络early stop；
- Bagging要求基学习器具有“不稳定”性，即：数据集上小的扰动能够使分类结果产生显著变动；
  - 从而得到的基分类器的之间的差异性较大

# 随机森林 (Random Forest)

- Bagging算法的一个扩展变体
- 被誉为“代表集成学习技术水平的方法”
- 以决策树为基学习器构建Bagging集成
- “样本集扰动” + “属性集扰动”

---

## 传统决策树

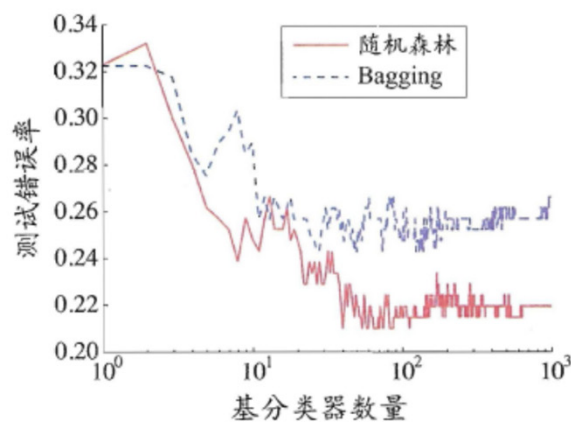
在选择划分属性时在**当前节点的属性集合**中选择一个最优属性

## 随机森林

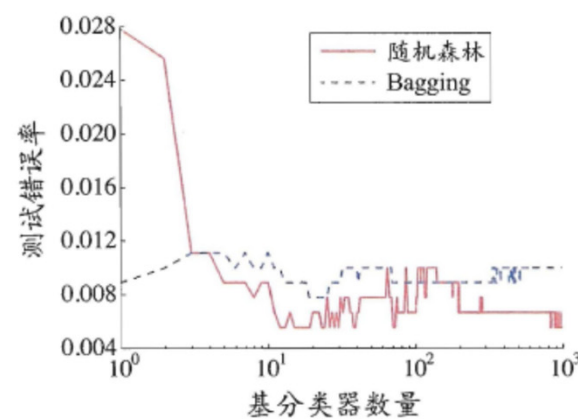
对基决策树的每个节点，先从该节点的属性集合中**随机选择一个包含k个属性的子集**，然后从该子集中选择一个最优属性用于划分。

# 随机森林 (Random Forest)

- 随机森林的收敛性与Bagging类似
- 随着个体学习器数目的增加，随机森林通常会收敛到更低的泛化误差
- 随机森林的训练效率常优于Bagging（Bagging在选择划分属性时需要考察结点的所有属性，而随机森林只需随机地考察一个属性子集）



(a) glass 数据集



(b) auto-mpg 数据集



# Boosting算法

# 强可学习与弱可学习

- **强可学习**(strongly learnable) vs **弱可学习**(weakly learnable)
  - 在概率近似正确 (probably approximately correct, PAC) 学习框架中, 一个概念, 若存在一个多项式的学习算法可以学习它, 并且正确率很高, 则称这个概念是**强可学习的**;
  - 一个概念, 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**弱可学习的**;
- **强可学习与弱可学习是等价的**, 即: 在PAC学习框架下, 一个概念是强可学习的充要条件是它是弱可学习的

# AdaBoost

- 问题：
  - 在学习中，如果已经发现了“弱学习算法”，那么能否将他提升(boost)为“强学习算法”？
- Boosting是一族**可将弱学习器提升为强学习器**的算法，其中最具代表性的是AdaBoost算法
- AdaBoost的基本流程：
  - 先从初始训练集训练出一个基学习器；
  - 再根据基学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续收到更多关注
  - 然后基于调整后的样本分布来训练下一个基学习器
  - 重复M轮得到M个基学习器，将其加权结合。

# 符号约定

- 假设给定一个二类分类的训练数据集

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

样本点：  $x_i \in \mathcal{X} \subseteq \mathbf{R}^n$

标 记：  $y_i \in \mathcal{Y} = \{-1, +1\}$

- 最终分类器：

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

组合系数

基分类器

# AdaBoost算法

- 输入:

- 训练数据集T
- 弱学习算法

- 输出:

- 最终分类器G(x)

- Step 1: 初始化训练数据的权值分布:

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

- Step 2: 对  $m = 1, 2, \dots, M$

- a) 使用具有权值分布  $D_m$  的训练数据集学习, 得到基本分类器:  $G_m(x): \mathcal{X} \rightarrow \{-1, +1\}$

# AdaBoost算法

- 2) 计算  $G_m(x)$  在训练数据集上的分类误差率:

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

- 3) 计算  $G_m(x)$  的系数:

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

**Why?  
Prove it!**

- 4) 更新训练数据集的权值分布:

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

$$w_{m+1,i} = \frac{1}{Z_m} w_{m,i} \exp(-\alpha_m y_i G_m(x_i)), i = 1, 2, \dots, N$$

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i)) \quad \text{规范化因子}$$

# AdaBoost算法

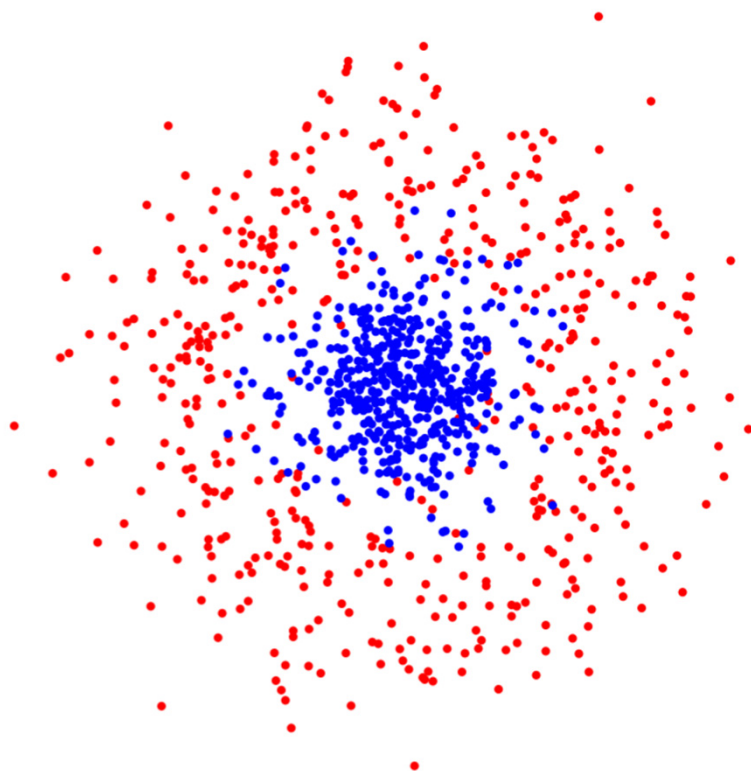
- **Step 3:** 构建基本分类器的线性组合

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

得到最终分类器：

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

# 一个例子



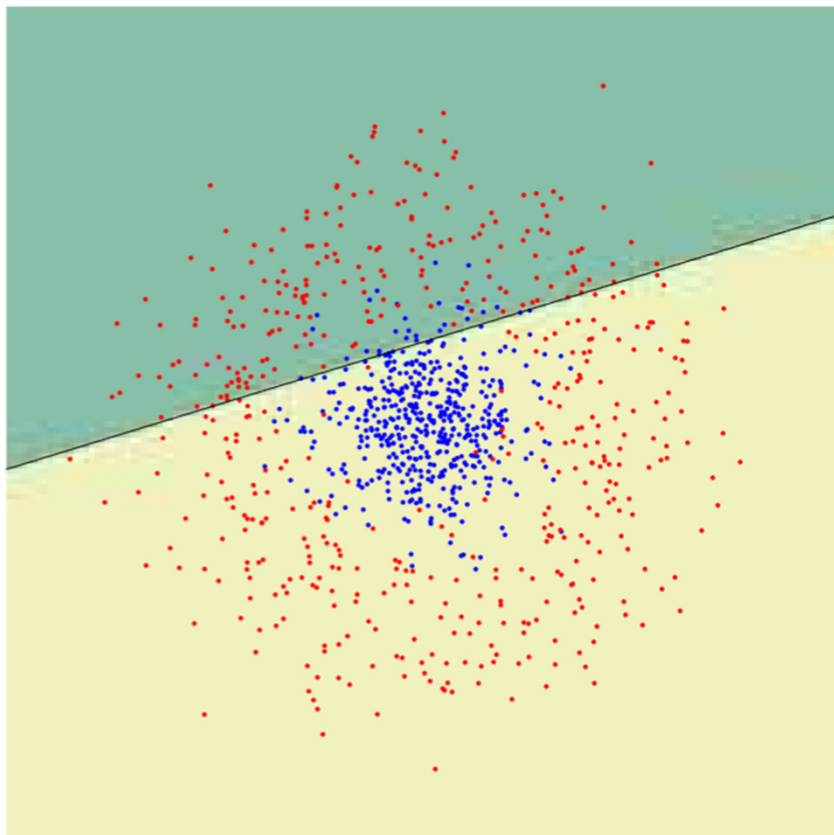
## 二分类问题

蓝点  $\sim N(0,1)$

红点  $\sim \frac{1}{r\sqrt{8\pi^3}} e^{-1/2(r-4)^2}$



# 一个例子



## 二类分类问题

蓝点  $\sim N(0,1)$

红点  $\sim \frac{1}{r\sqrt{8\pi^3}} e^{-1/2(r-4)^2}$

弱分类器采用感知器

# 一个例子

1. 初始化训练数据的权值分布

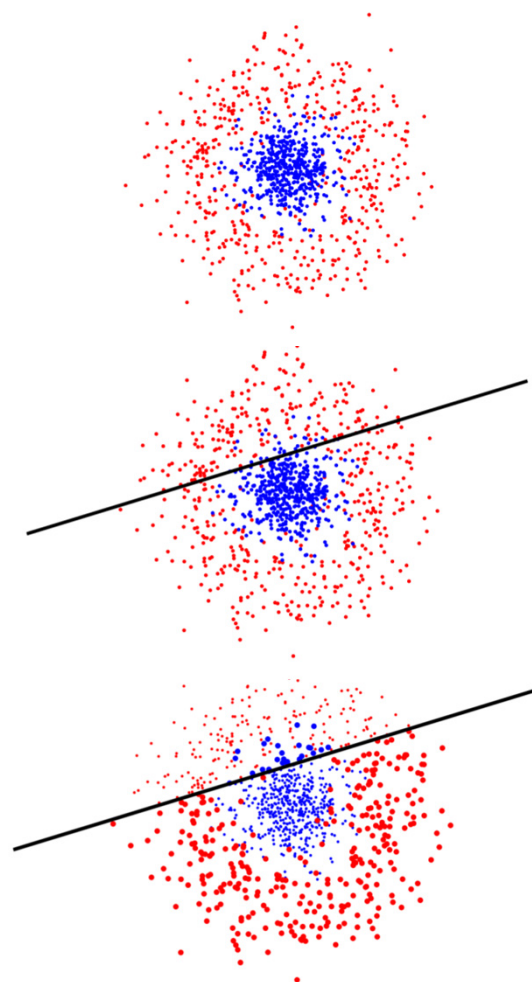
2. 对  $m = 1, 2, \dots, M$

a) 构造基本分类器

b) 计算该分类器的分类误差率

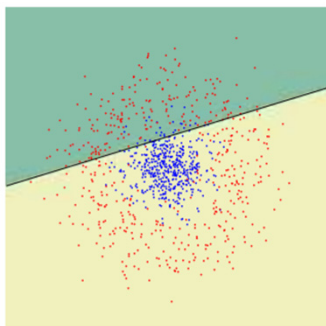
c) 计算该分类器的系数

d) 更新训练数据集的权值分布

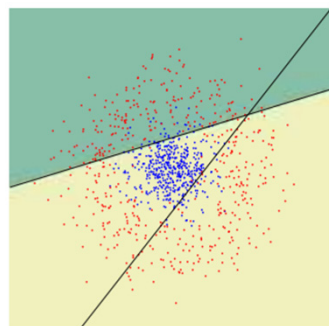


# 一个例子

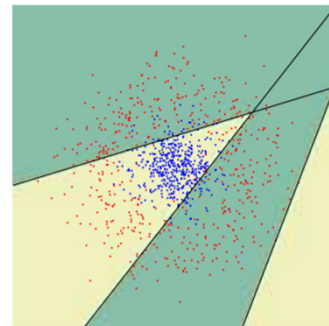
$m = 1$



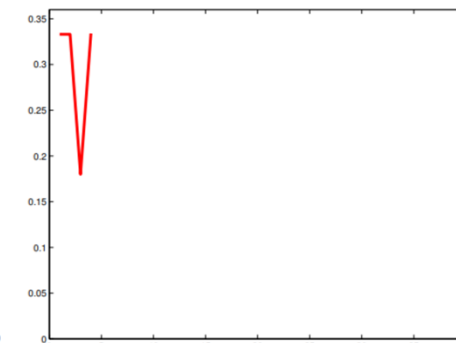
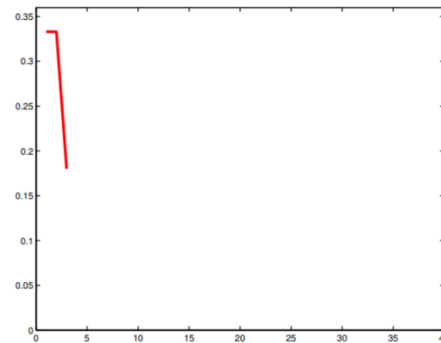
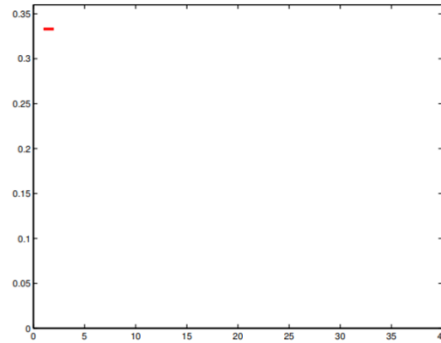
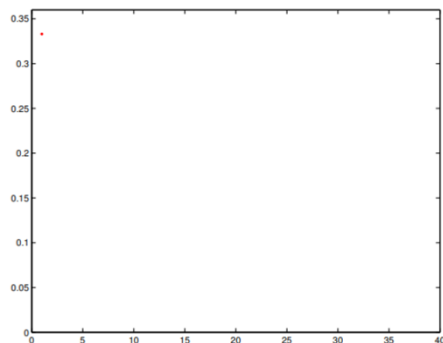
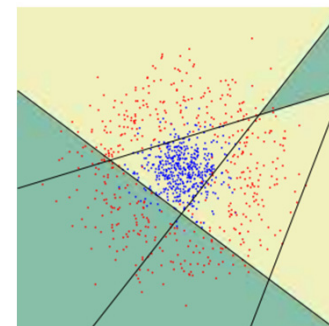
$m = 2$



$m = 3$

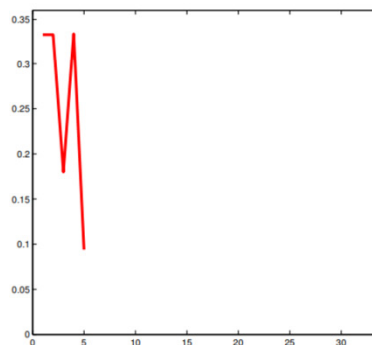
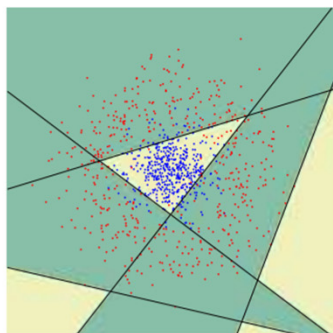


$m = 4$

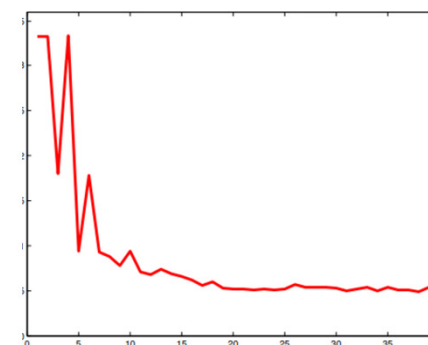
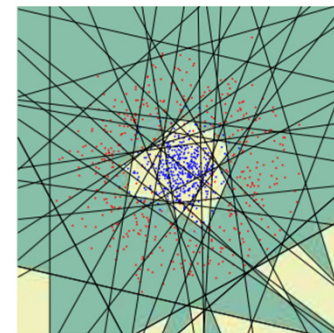


# 一个例子

$m = 5$



$m = 40$



训练样本的最终  
权重分布

# 算法的几点说明

- 在步骤2中，AdaBoost使用当前分布 $D_m$ 加权的训练数据集，来学习基本分类器 $G_m(x)$
- 基本分类器 $G_m(x)$ 在当前加权训练数据集上的分类误差率与样本权值有关：

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{G_m(x_i) \neq y_i} w_{mi}$$

这里 $w_{mi}$ 表示第 $m$ 轮中第 $i$ 个实例的权值， $\sum_{i=1}^N w_{mi} = 1$   
这表明 $G_m(x)$ 在加权的训练数据集上的分类误差率  
是被 $G_m(x)$ 误分类样本的权值之和。

# 算法的几点说明

- $G_m(x)$  的系数  $\alpha_m$  表示  $G_m(x)$  在最终分类器中的重要性。

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

可见：当  $e_m \leq \frac{1}{2}$  时， $\alpha_m \geq 0$ ，并且  $\alpha_m$  随着  $e_m$  的减小而增大，所以误差率越小的分类器在最终分类器中的作用越大；

# 算法的几点说明

- 样本点的权值更新可写成如下形式：

$$w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases}$$

可见，被基本分类器 $G_m(x)$ 误分类样本的权值将增大，而被正确分类样本的权值将缩小。因此，误分类样本在下一轮学习中将起更大的作用。

- 不改变所给的训练数据，而不断改变训练数据权值的分布，使得数据在基本分类器的学习中起不同作用是AdaBoost的一个特点；

# AdaBoost训练误差分析

- AdaBoost能在学习过程中不断减少训练误差，即在训练数据集上的分类误差

**定理1：** AdaBoost算法最终分类器训练误差界为

$$\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \frac{1}{N} \sum_i \exp(-y_i f(x_i)) = \prod_m Z_m$$

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

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

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



证明:

(前半部分) 当 $G(x_i) \neq y_i$ 时,  $y_i f(x_i) < 0$ , 因此

$$I(G(x_i) \neq y_i) = 1 \leq \exp(-y_i f(x_i))$$

(后半部分)

$$\begin{aligned} & \frac{1}{N} \sum_i \exp(-y_i f(x_i)) \\ &= \frac{1}{N} \sum_i \exp\left(-\sum_{m=1}^M \alpha_m y_i G_m(x_i)\right) \quad \left(w_{1i} = \frac{1}{N}\right) \\ &= \sum_i w_{1i} \prod_{m=1}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= Z_1 \sum_i \frac{w_{1i} \exp(-\alpha_1 y_i G_1(x_i))}{Z_1} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \end{aligned}$$

证明（续）：

$$\begin{aligned}
 &= Z_1 \sum_i \frac{w_{1i} \exp(-\alpha_1 y_i G_1(x_i))}{Z_1} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \\
 &= Z_1 \sum_i w_{2i} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \left( w_{m+1,i} = \frac{1}{Z_m} w_{m,i} \exp(-\alpha_m y_i G_m(x_i)) \right) \\
 &= Z_1 Z_2 \sum_i w_{3i} \prod_{m=3}^M \exp(-\alpha_m y_i G_m(x_i)) \\
 &= \dots \\
 &= Z_1 Z_2 \cdots Z_{M-1} \sum_i w_{Mi} \exp(-\alpha_M y_i G_M(x_i)) \\
 &= \prod_{m=1}^M Z_m \sum_i w_{M+1,i} \left( \sum_i w_{M+1,i} = 1 \right) \\
 &= \prod_{m=1}^M Z_m
 \end{aligned}$$

# Remarks

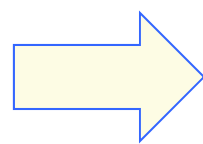
$$\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \frac{1}{N} \sum_i \exp(-y_i f(x_i)) = \prod_m Z_m$$

技巧1. 优化损失函数的上界

$$L_0 = \frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \prod_m Z_m = L_1$$

技巧2. 分步优化

$$\min L_1 = \min \prod_m Z_m$$


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

# Remarks

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

$$\frac{\partial L}{\partial \alpha_m} = -\sum_{i=1}^N w_{mi} y_i G_m(x_i) \exp(-\alpha_m y_i G_m(x_i)) = 0$$

$$\Rightarrow \sum_{i: G(x_i)=y_i} w_{mi} e^{-\alpha_m} + \sum_{i: G(x_i) \neq y_i} -w_{mi} e^{\alpha_m} = 0$$

$$\Rightarrow \exp(-\alpha_m)(1 - e_m) - \exp(\alpha_m)e_m = 0$$

$$\Rightarrow \alpha_m = \frac{1}{2} \log \left( \frac{1 - e_m}{e_m} \right)$$

# AdaBoost训练误差分析

**定理2：** 二类分类问题AdaBoost的训练误差界

$$\prod_{m=1}^M Z_m = \prod_{m=1}^M \left[ 2\sqrt{e_m(1-e_m)} \right] = \prod_{m=1}^M \sqrt{1-4\gamma_m^2} \leq \exp\left(-2\sum_{m=1}^M \gamma_m^2\right)$$

这里,  $\gamma_m = \frac{1}{2} - e_m$ 。

**证明：**

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

$$= \sum_{y_i=G(x_i)} w_{mi} e^{-\alpha_m} + \sum_{y_i \neq G(x_i)} w_{mi} e^{\alpha_m}$$

$$= (1-e_m) e^{-\alpha_m} + e_m e^{\alpha_m}$$

$$\alpha_m = \frac{1}{2} \log\left(\frac{1-e_m}{e_m}\right)$$

$$= 2\sqrt{e_m(1-e_m)} = \sqrt{1-4\gamma_m^2}$$

# AdaBoost训练误差分析

## 证明（续）

利用 $e^x$ 和 $\sqrt{1-x}$ 在点 $x=0$ 的泰勒展开式，  
可推出不等式

$$\sqrt{1-4\gamma_m^2} \leq \exp\left(-2\sum_{m=1}^M \gamma_m^2\right)$$

**推论：**如果存在 $\gamma > 0$ ，对所有 $m$ 有 $\gamma_m \geq \gamma$ ，则有

$$\frac{1}{N} \sum_{m=1}^M I(G(x_i) \neq y_i) \leq \exp(-2M\gamma^2)$$

# AdaBoost算法的解释

- AdaBoost算法
  - 模型为加法模型
  - 损失函数为指数函数
  - 学习算法为前向分步算法
  - 二类分类学习方法

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

$$L(y, f(x)) = \exp[-yf(x)]$$

# 前向分步算法

- 考虑加法模型

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

- 其中

$b(x; \gamma_m)$  为基函数；

$\gamma_m$  为基函数的参数；

$\beta_m$  为基函数的系数；



# 前向分步算法

- 给定训练数据和损失函数的条件下，学习加法模型  $f(x)$  成为经验风险极小化，也即损失函数极小化问题：

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

- 这通常是一个复杂的优化问题
- 前向分步算法利用**贪婪法求解**这一优化问题：
  - 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，也即每步求解：

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

# 前向分步算法步骤

输入：训练数据集；损失函数；基函数集

输出：加法模型  $f(x)$

1) 初始化  $f_0(x) = 0$

2) 对  $m = 1, 2, \dots, M$

a) 极小化损失函数得到参数  $\beta_m, \gamma_m$

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

b) 更新

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

3) 得到加法模型

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

# 前向分步算法与AdaBoost

- 定理：AdaBoost算法是前向分步算法的特例。其中，模型是由基本分类器组成的加法模型，损失函数是指数函数。

- 最终分类器：
$$f(x) = \sum_{m=1}^M \alpha_m G_m(x)$$
- 指数损失函数： $L(y, f(x)) = \exp[-yf(x)]$
- 假设经过前  $m-1$  轮迭代前向分步算法已经得到

$$\begin{aligned} f_{m-1}(x) &= f_{m-2}(x) + \alpha_{m-1} G_{m-1}(x) \\ &= \alpha_1 G_1(x) + \cdots + \alpha_{m-1} G_{m-1}(x) \end{aligned}$$

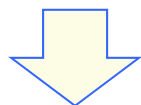
# 前向分步算法与AdaBoost

- 在第  $m$  轮迭代得到  $\alpha_m, G_m(x)$  和  $f_m(x)$

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

- 为此，前向分步算法优化  $\alpha_m, G_m(x)$  使得  $f_m(x)$  在训练数据集上的指数损失函数最小，即：

$$(\alpha_m, G_m(x)) = \arg \min_{\alpha, G} \sum_{i=1}^N \exp[-y_i (f_{m-1}(x_i) + \alpha G(x_i))]$$



$$(\alpha_m, G_m(x)) = \arg \min_{\alpha, G} \sum_{i=1}^N \bar{w}_{mi} \exp[-y_i \alpha G(x_i)]$$

$$\text{其中, } \bar{w}_{mi} = \exp[-y_i f_{m-1}(x_i)]$$

# 前向分步算法与AdaBoost

- 可以证明，使得上式达到最小的  $\alpha_m^*, G_m^*(x)$  就是 AdaBoost 算法所得到的  $\alpha_m, G_m(x)$ 。
- 求解过程可分为两步：
  - 求  $G_m^*(x)$ 。对任意的  $\alpha > 0$ ,  $G_m^*(x)$  可通过优化下式得到

$$G_m^*(x) = \arg \min_G \sum_{i=1}^N \bar{w}_{mi} I(y_i \neq G(x_i))$$

$$\text{其中, } \bar{w}_{mi} = \exp[-y_i f_{m-1}(x_i)]$$

- 求  $\alpha_m^*$  (过程略)。
- 样本权值的更新（迭代公式）：

$$\begin{aligned} \bar{w}_{m+1,i} &= \exp[-y_i f_m(x_i)] = \exp[-y_i (f_{m-1}(x_i) + \alpha_m G_m(x_i))] \\ &= \bar{w}_{mi} \exp[-y_i \alpha_m G_m(x_i)] \end{aligned}$$

# Gradient Boosting算法

- AdaBoost算法针对指数型损失函数，对于一般的损失函数不适用；
- Gradient Boosting 算法本质上是一个 “Iterative Functional Gradient Decent Algorithms”
  - 通过在函数空间，沿负梯度方向，迭代选取弱学习器来优化损失函数；
  - 可看作AdaBoost算法的推广；
  - 可适用于任意可微分的损失函数；

$$\min_{F(x)} L(y, F(x)) \implies F_{m+1}(x) \leftarrow F_m(x) - \gamma \nabla_{F_{m-1}} L(y, F_m(x))$$

函数空间的梯度下降

# Gradient Boosting基本思想

- 举例：
  - 考虑最小二乘回归问题
  - 通过最小化均方误差 $(\hat{y} - y)^2$ 来学习一个模型  $\hat{y} = F(x)$
  - 采用迭代方式来学习：假设在第 $m$ 轮已经得到一个模型估计  $F_m(x)$ ，在第 $m+1$ 轮对其添加一个  $h(x)$  得到一个更好的模型估计：

$$F_{m+1}(x) = F_m(x) + h(x)$$

- Gradient Boosting算法试图求解最优的 $h(x)$ ，即

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

或  $h(x) = y - F_m(x)$ ，也即 $h(x)$ 可通过拟合残差得到。

注意该残差恰好是均方误差关于模型的梯度

# Gradient Boosting基本思想

- 可将上述思想推广到一般的损失函数情形
- 给定一组训练样本，监督学习的目标是

$$\hat{F} = \arg \min_F \mathbb{E}_{x,y} [L(y, F(x))]$$

- Gradient Boosting采用如下形式的加法模型

$$F(x) = \sum_{i=1}^M \gamma_i h_i(x) + \text{const.}$$

- 求解时采用贪婪法来逐步优化损失函数：

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

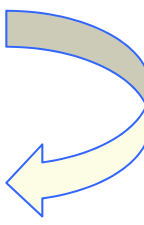
$$F_m(x) = F_{m-1}(x) + \arg \min_{h_m \in \mathcal{H}} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h_m(x_i))$$



# Gradient Boosting基本思想

$$F_m(x) = F_{m-1}(x) + \arg \min_{h_m \in \mathcal{H}} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + h_m(x_i))$$

- 对于任意形式的损失函数，求解上述优化问题非常困难。
- Gradient Boosting采用梯度下降来替代该优化：

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^n \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))$$
$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i)))$$


- 注意：上述做法只是一个启发式方法，不能得到精确解，而只是一个好的近似。

# Gradient Boosting算法流程

Input: training set  $\{(x_i, y_i)\}_{i=1}^n$ , a differentiable loss function  $L(y, F(x))$ , number of iterations  $M$ .

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma).$$

2. For  $m = 1$  to  $M$ :

1. Compute so-called *pseudo-residuals*:

$$r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

2. Fit a base learner (e.g. tree)  $h_m(x)$  to pseudo-residuals, i.e. train it using the training set  $\{(x_i, r_{im})\}_{i=1}^n$ .

3. Compute multiplier  $\gamma_m$  by solving the following **one-dimensional optimization** problem:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output  $F_M(x)$ .

# 提升树

- 提升树是以分类树或回归树为基本分类器的提升方法；
- 该方法被认为是统计学习中性能最好的方法之一
- 针对不同的问题有不同的提升树学习方法，其主要区别在于采用的损失函数不同
  - 分类问题：指数损失函数
  - 回归问题：平方误差函数
  - 一般决策问题：一般的损失函数

# 针对回归问题的提升树

- 训练数据集

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}, x_i \in \mathcal{X} \subseteq \mathbf{R}^n, y_i \in \mathcal{Y} \subseteq \mathbf{R}$$

- 如果将输入空间  $\mathcal{X}$  划分成  $J$  个互不相交的区域，且每个区域上确定输出的常量，则树可表示成：

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

- 其中：

- 参数  $\Theta = \{(R_1, c_1), (R_2, c_2), \dots, (R_J, c_J)\}$  表示树的区域划分和各区域上的常量；

# 针对回归问题的提升树

- 回归问题提升树采用如下的前向分步算法：

$$f_0(x) = 0$$

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

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

- 在前向分步算法的第 $m$ 步，给定当前模型  $f_{m-1}(x)$ ，需求解

$$\hat{\Theta}_m = \arg \min_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

得到  $\hat{\Theta}_m$ ，即第 $m$ 棵树的参数

# 针对回归问题的提升树

- 若采用平方误差损失函数：

$$L(y, f(x)) = (y - f(x))^2$$

带入模型，其损失变为

$$\begin{aligned} L(y, f_{m-1}(x) + T(x; \Theta_m)) \\ &= [y - f_{m-1}(x) - T(x; \Theta_m)]^2 \\ &= [r - T(x; \Theta_m)]^2 \end{aligned}$$

因此，对回归问题的提升树算法，只需简单的拟合当前模型的残差即可。

当前模型  
拟合数据  
的残差

# 回归问题提升树算法

输入：  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}, x_i \in \mathcal{X} \subseteq \mathbf{R}^n, y_i \in \mathcal{Y} \subseteq \mathbf{R}$

输出： 提升树  $f_M(x)$

1) 初始化  $f_0(x) = 0$

2) 对  $m = 1, 2, \dots, M$

(a) 计算数据的拟合残差  $r_{mi} = y_i - f_{m-1}(x_i), i = 1, 2, \dots, N$

(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) 得到回归问题提升树

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

# 梯度提升

- 当损失函数是平方损失和指数损失时，上述每一步的优化都很简单；
- 对于一般的损失函数，可以采用梯度提升算法（Gradient Boosting）；
- 其中的关键步骤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，来拟合一个回归树。即：

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

- (算法步骤略)



# 集成学习算法分析

# 集成学习

- 集成学习通过构建并结合多个学习器来完成学习任务；
- 代表性的两类方法
  - Bagging：个体学习器之间不存在强依赖关系、可同时生成的并行化方法
  - Boosting：个体学习器之间存在强依赖关系、必须串行生成的序列化方法
- 为得到好的集成，要求集成个体应“好而不同”
- 准确性 vs 多样性

# 多样性增强

- 在集成学习中需要有效地生成多样性大的个体学习器，那么如何增强多样性？
- 思路是在学习过程中引入随机性
  - 数据样本扰动：对“不稳定基学习器”有效
  - 输入属性扰动：随机森林、随机子空间算法等
  - 输出表示扰动：对输出表示进行操纵来增强多样性。如“翻转法”随机改变一些训练样本的标记等；
  - 算法参数扰动：随机设置基学习算法的参数来产生差别较大的个体学习器。如Dropout等

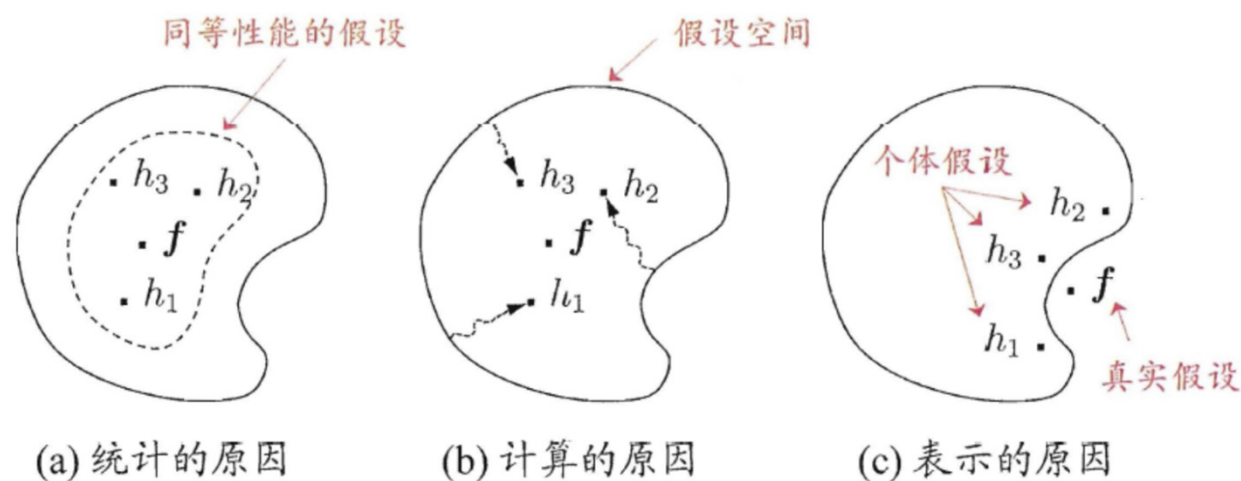
# 集成个体的结合策略

- 学习器结合可能从三个方面带来好处：
  - 1) 统计方面：
    - 学习任务的假设空间往往很大，可能有多个假设在训练集上达到同等性能，若使用单学习器可能因误选导致泛化性能不佳，而结合多个学习器则会减少这一风险；
  - 2) 计算方面：
    - 学习算法往往会陷入局部极小，有的局部极小点所对应的泛化性能可能很糟糕，而通过多次运行之后进行结合，则可降低陷入糟糕局部极小点的风险；

# 集成个体的结合策略

## — 3) 表示方面:

- 某些学习任务的真实假设可能不在当前学习算法所考虑的假设空间中，此时若使用单学习器肯定无效，而通过结合多个学习器，由于相应的假设空间有所扩大，有可能学到更好的近似；



# 集成个体的结合策略

- 基学习器的结合策略
  - 平均法
    - 简单平均法
    - 加权平均法
  - 投票法
    - 绝对多数投票法
    - 相对多数投票法
    - 加权投票法
  - 学习法：
    - 即通过另一个学习器来进行结合，如stacking算法等

# 偏差方差分解

- 分析学习算法的一个有效工具
- 对于一个训练得到的模型  $\hat{f}(x)$ ，给定测试集中的一个点  $x$ ，可利用  $\hat{f}(x)$  得到对应的预测。预测的均方误差为：

$$E_{(x,y) \sim \text{testset}} \left| \hat{f}(x) - y \right|^2$$

- 如果均方误差高，则通常是由于：
  - 过拟合
  - 欠拟合
  - 数据中的噪声

# 偏差方差分解

- 假设训练数据来自如下模型：

$$y_i = f(x_i) + \varepsilon_i$$

其中,  $E\varepsilon_i = 0$ ,  $Var(\varepsilon_i) = \sigma^2$

- 说明：
  - 对于给定的  $x_j$ ,  $f(x_j)$  是固定值, 而  $\varepsilon_j$  是随机变量
  - 模型的预测值  $\hat{f}(x_j)$  是随机的, 因为它依赖于训练数据中的  $\varepsilon_i$ , 因此偏差  $E(\hat{f}(x) - f(x))$  和  $\hat{f}(x)$  的方差才有定义



# 偏差方差分解

- 测试集上的均方误差可分解如下:

$$\begin{aligned} \text{Test MSE} &= E((y - \hat{f}(x))^2) \\ &= E((\varepsilon + f(x) - \hat{f}(x))^2) \\ &= E(\varepsilon^2) + E((f(x) - \hat{f}(x))^2) \\ &= \sigma^2 + [E(f(x) - \hat{f}(x))]^2 + \text{Var}(f(x) - \hat{f}(x)) \\ &= \sigma^2 + [\text{Bias}(\hat{f}(x))]^2 + \text{Var}(\hat{f}(x)) \end{aligned}$$

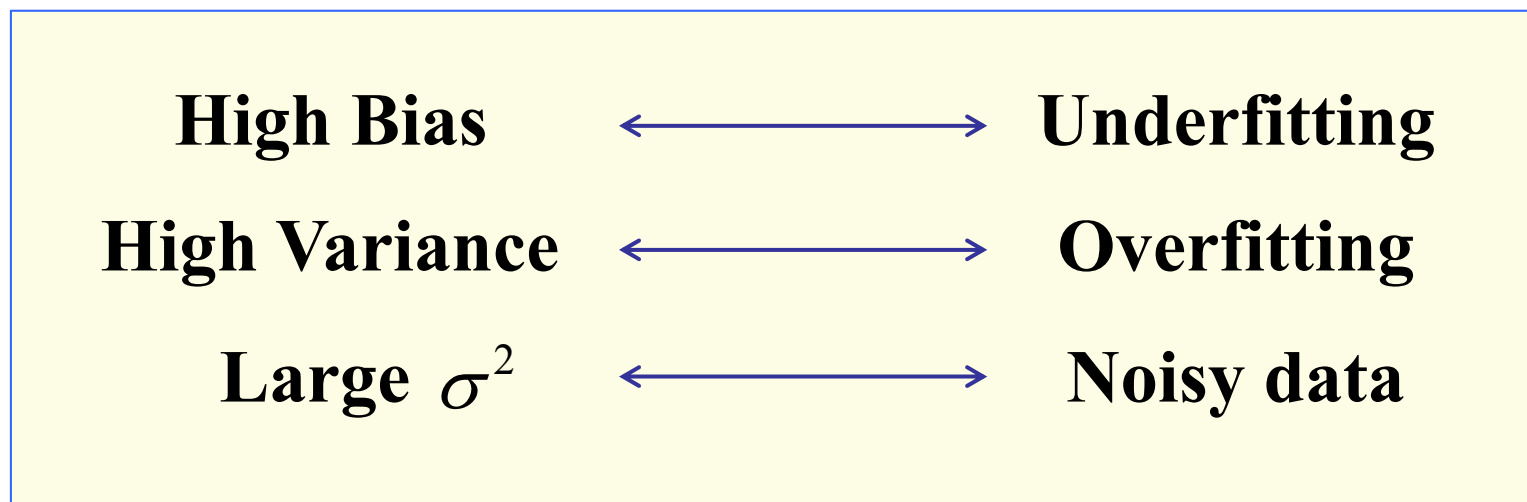
数据噪声

模型偏差

模型方差

# 偏差方差分解

- 说明：
  - $\sigma^2$  是数据自身的噪声
  - 偏差项是由于模型的欠拟合
  - 方差项与数据的过拟合有关



# Bagging vs Boosting

- 集成学习的两个代表性方法
- Bagging通过数据扰动构造基学习器，基学习器可并行同时生成
- Boosting通过数据加权构造基学习器，基学习器必须串行生成
- 从偏差方差分解角度来看
  - Bagging试图降低模型的方差
  - Boosting试图降低模型的偏差
- Boosting可看作函数空间上的梯度下降

# 参考文献

- 周志华, 《机器学习》
- 李航, 《统计学习方法》
- [https://en.wikipedia.org/wiki/Gradient\\_boosting](https://en.wikipedia.org/wiki/Gradient_boosting)

Thank All of You!  
(Questions?)

孟高峰

[gfmeng@nlpr.ia.ac.cn](mailto:gfmeng@nlpr.ia.ac.cn)

<http://www.escience.cn/people/gfmeng>

时空数据分析与学习课题组 (STDAL)

中科院自动化研究所· 模式识别国家重点实验室