



软件开发环境国家重点实验室  
State Key Laboratory of Software Development Environment

# 机器学习

刘祥龙

北京航空航天大学计算机学院  
软件开发环境国家重点实验室

2018年11月20日

# 朴素贝叶斯分类器

- 估计后验概率  $P(c | \mathbf{x})$  主要困难：类条件概率  $P(\mathbf{x} | c)$  是所有属性上的联合概率难以从有限的训练样本估计获得。
- 朴素贝叶斯分类器(Naïve Bayes Classifier)采用了“属性条件独立性假设”(attribute conditional independence assumption)：每个属性独立地对分类结果发生影响。
- 基于属性条件独立性假设，基于生成式模型的贝叶斯决策可重写为

$$P(c | \mathbf{x}) = \frac{P(c)P(\mathbf{x} | c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^d P(x_i | c)$$

- 其中  $d$  为属性数目， $x_i$  为  $\mathbf{x}$  在第  $i$  个属性上的取值。

$$P(c \mid \mathbf{x}) = \frac{P(c)P(\mathbf{x} \mid c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^d P(x_i \mid c)$$

由于对所有类别来说  $P(\mathbf{x})$  是常数，因此贝叶斯判定准则可以简化为

$$h_{nb}(\mathbf{x}) = \operatorname{argmax}_{c \in y} P(c) \prod_{i=1}^d P(x_i \mid c)$$

## • 模型训练

- 基于训练集  $D$  估计类先验概率  $P(c)$

令  $D_c$  表示训练集  $D$  中第  $c$  类样本组合的集合，若有充足的独立同分布样本，则可容易地估计出类先验概率

$$P(c) = \frac{|D_c|}{|D|}$$

- 对每个属性估计条件概率  $P(x_i | c)$

对离散属性而言，令  $D_{c,x_i}$  表示  $D_c$  中第  $x_i$  个属性取值  $x_i$  的样本组合的集合，则条件概率可估计

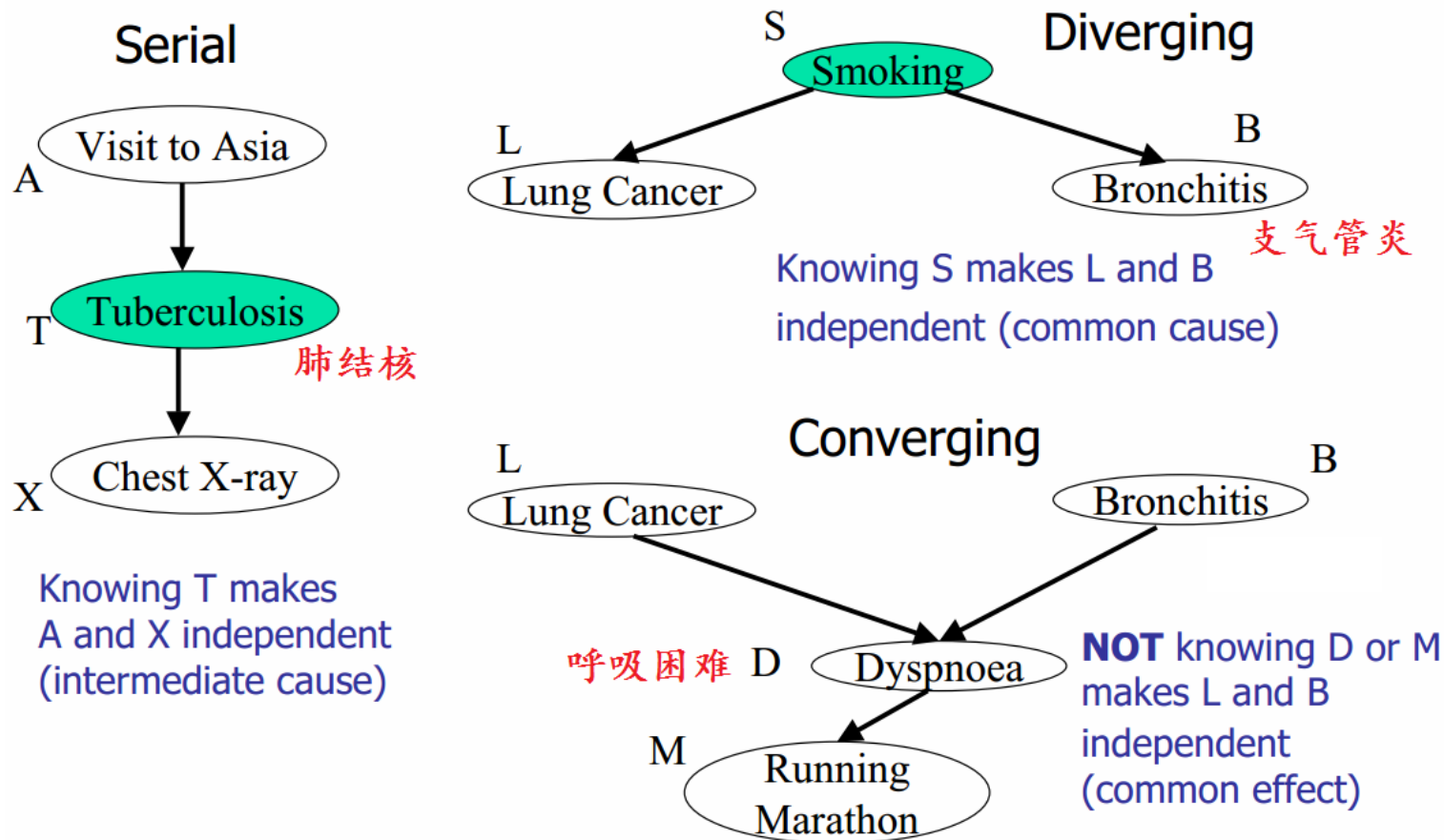
$$P(x_i | c) = \frac{|D_{c,x_i}|}{|D_c|}$$

对连续属性而言可考虑概率密度函数，假定  $p(x_i | c) \sim N(\mu_{c,i}, \sigma_{c,i}^2)$ ， $\mu_{c,i}$  与  $\sigma_{c,i}^2$  是第  $c$  类样本在第  $i$  个属性  $x_i$  上取值的均值和方差，则有

$$P(x_i | c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

# 贝叶斯网：结构

- 分析有向图中变量间的条件独立性，可使用“有向分离”（D-separation）



# K-means

- 问题：给定数据集  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ ，发现  $K$  个聚类
- 模型：每个数据分配给距离最近的类

$$k = \arg \min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2$$

- 优化目标：寻找聚类（中心  $\boldsymbol{\mu}_k$ ）使得所有数据到其聚类中心距离和最小

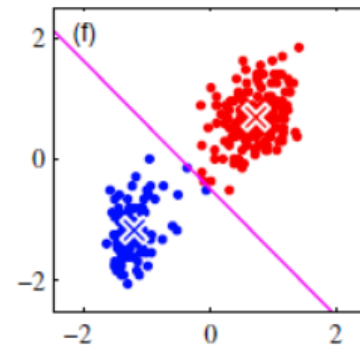
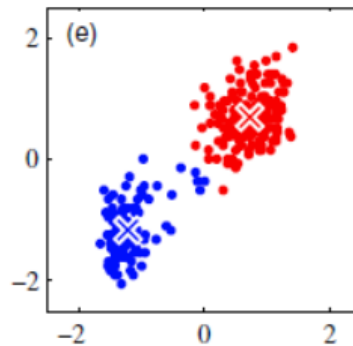
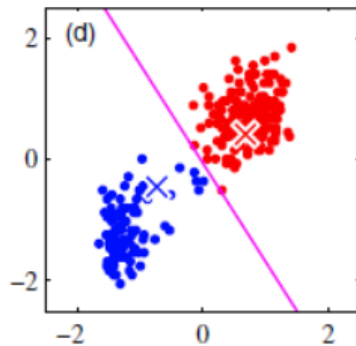
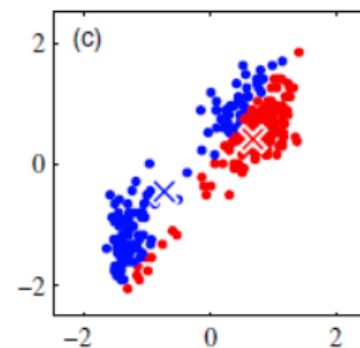
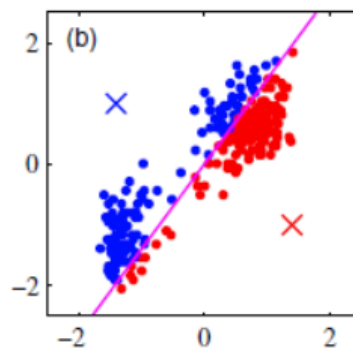
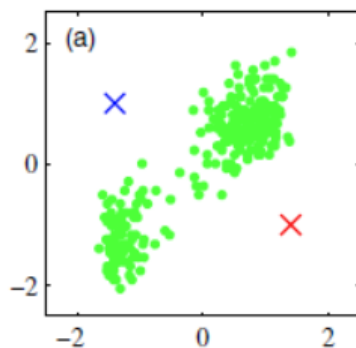
$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

# K-means

- 迭代优化:

- 计算每个数据到当前聚类中心的距离
- 对每个数据分配到其最近的聚类
- 重新计算聚类中心



- 停止条件

- 目标函数不变
- 最大迭代次数



## 更一般的问题：含有隐变量的参数估计



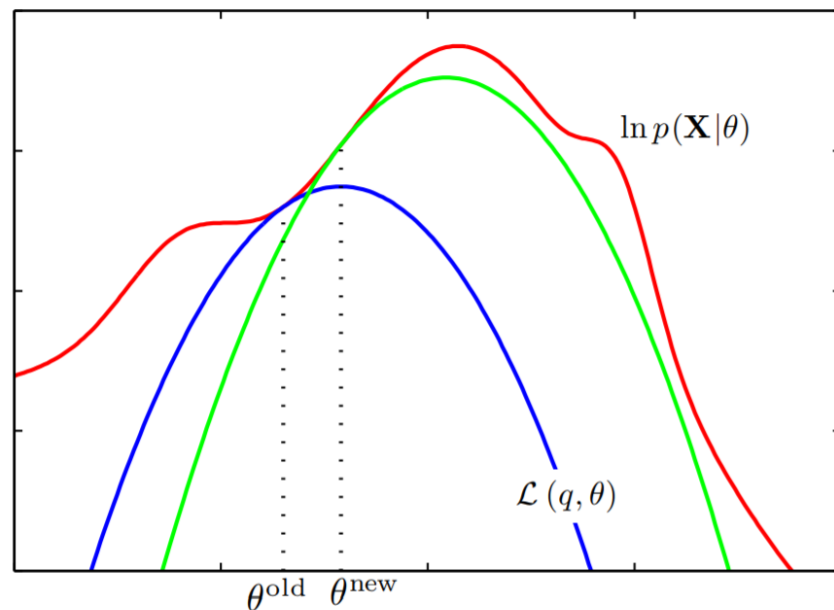
- “不完整”的样本：未观测的变量称为“隐变量” (latent variable)，比如聚类中每个样例属于哪个类。
- 未观测的变量称为“隐变量” (latent variable)。令  $\mathbf{X}$  表示已观测变量集， $\mathbf{Z}$  表示隐变量集，若预对模型参数  $\Theta$  做极大似然估计，则应最大化对数似然函数

$$LL(\Theta \mid \mathbf{X}, \mathbf{Z}) = \ln P(\mathbf{X}, \mathbf{Z} \mid \Theta)$$

- 由于  $\mathbf{Z}$  是隐变量，上式无法直接求解。此时我们可以通过对  $\mathbf{Z}$  计算期望，来最大化已观测数据的对数“边际似然” (marginal likelihood)

$$LL(\Theta \mid \mathbf{X}) = \ln P(\mathbf{X} \mid \Theta) = \ln \sum_{\mathbf{Z}} P(\mathbf{X}, \mathbf{Z} \mid \Theta)$$





**E步骤:** 根据参数初始值或上一次迭代的模型参数来计算出隐性变量的后验概率:

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)}; \theta).$$

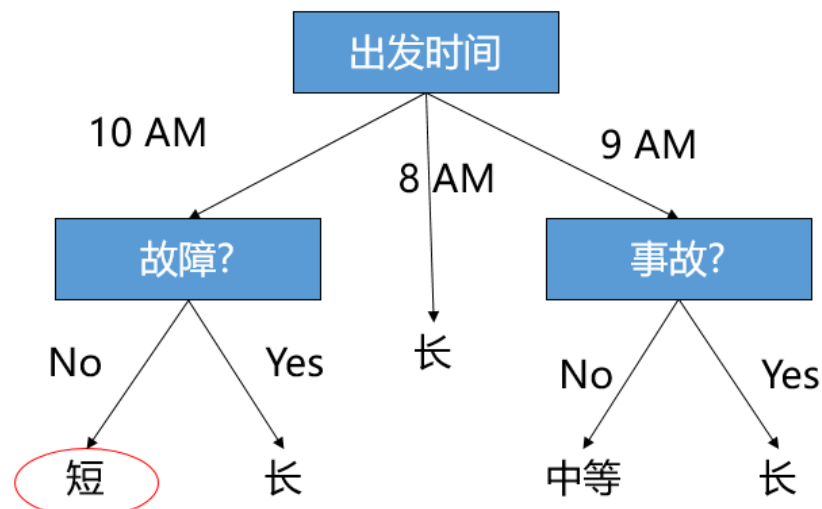
**M步骤:** 将似然函数最大化以获得新的参数值:

$$\theta := \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

# 决策树生成

## • 决策树创建的基本步骤

- ❑ 选择当前最佳属性作为决策节点
- ❑ 把剩余的样例划分到子节点
- ❑ 递归对子节点重复上述步骤
- ❑ 停止条件
  - 所有样例具有相同的目标属性值
  - 没有更多的属性可选择
  - 没有更多的样例



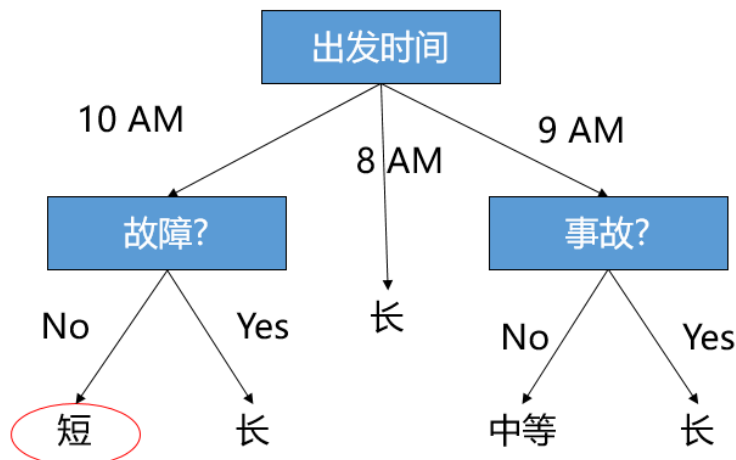
## • 熵的特点

- 当所有目标属性取值相同时，熵最小：如果通勤时间总是“短”，则熵为0
- 当目标属性各种取值均等时，熵最大（即结果是随机的）：如果通勤时间为“短”、“中等”、“长”的样例均为3个

## • 如何基于熵选择属性

- 出发时间？故障？
- 基本思想：

**选择信息量减少最快的属性**



以信息熵为度量，用于决策树节点的属性选择，每次优先选取信息量最多的属性，亦即使熵值变为最小的属性，以构造一颗熵值下降最快的决策树，到叶子节点处的熵值为0。此时，每个叶子节点对应的实例集中的实例属于同一类。

- 设训练样本集为 $D$ ,  $|D|$ 表示样本容量。样本目标属性共有 $K$ 个类 $C_k$ ,  $k=1, 2, \dots, K$ ,  $\sum_{k=1}^K |C_k| = |D|$
- 设属性 $A$ 有 $n$ 个不同的取值 $\{a_1, a_2, \dots, a_n\}$ , 根据属性 $A$ 的取值将 $D$ 划分为 $n$ 个子集 $D_1, D_2, \dots, D_n$ , 记子集 $D_i$ 中属于类 $C_k$ 的样本的集合为 $D_{ik}$ ,  $D_{ik} = D_i \cap C_k$

- (1) 计算数据集 $D$ 的经验熵

$$H(D) = - \sum_{k=1}^K \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

- (2) 计算每个属性 $A$ 对数据集 $D$ 的经验条件熵

$$H(D|A) = - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) = - \sum_{i=1}^n \frac{|D_i|}{|D|} \sum_{k=1}^K \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

- (3) 计算每个属性 $A$ 的信息增益

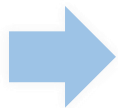
$$g(D, A) = H(D) - H(D|A)$$

# 决策树的问题



决策树

容易过拟合



随机森林

利用多棵树对样本进行训练并预测  
属性可随机抽取，更好的泛化能力



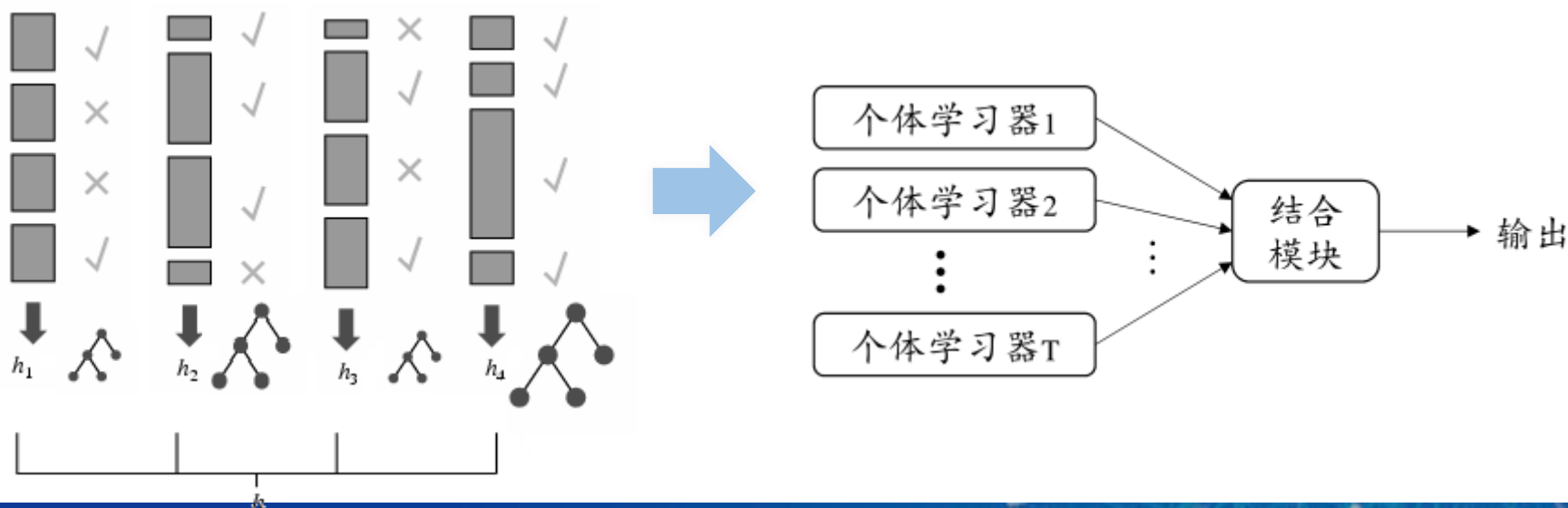
软件开发环境国家重点实验室  
State Key Laboratory of Software Development Environment

# 集成学习



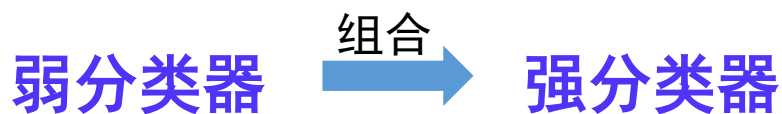
# 个体与集成

- No Free Lunch: 没有单一的算法可以保证永远最好
- 集成学习(ensemble learning)通过构建并结合多个学习器来提升性能
- 不同的学习器采用不同的算法、参数、表征（属性、特征、模态等）、训练数据、子问题等



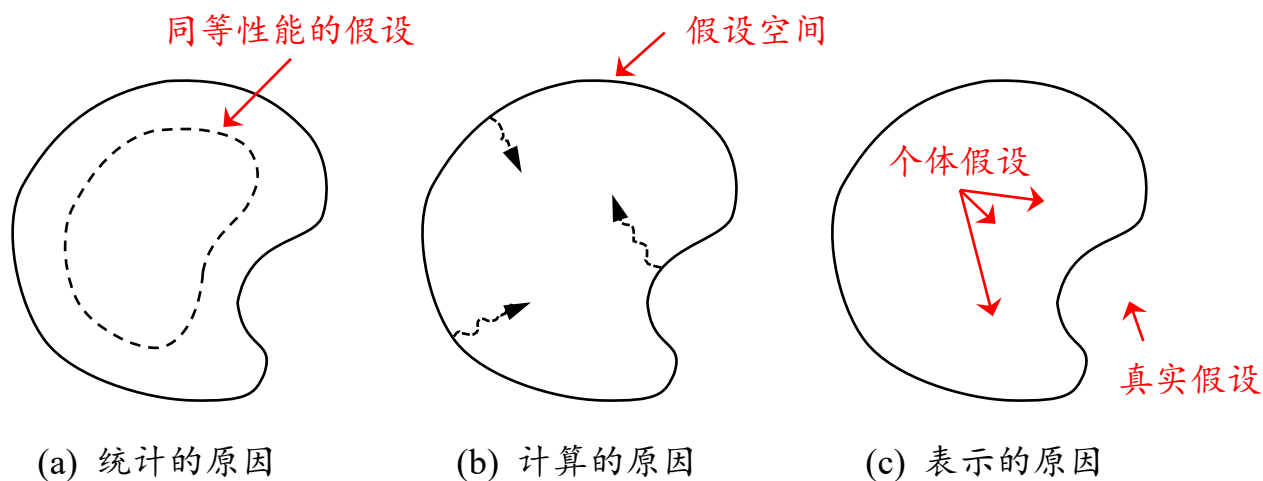


- 将多个分类器的判断进行适当的综合所得出的判断，比单独一个分类器判断的结果更好。
- 弱分类器——识别错误率小于 $1/2$ ，也即准确率仅比随机猜测略高的分类器。
- 强分类器——识别准确率很高，并能在多项式时间内完成的分类器。



三个臭裨将（皮匠），胜过诸葛亮！

- 学习器的组合可以从三个方面带来好处



- 考虑二分类问题，假设基分类器的错误率为：

$$P(h_i(\mathbf{x}) \neq f(\mathbf{x})) = \epsilon$$

- 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

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

- 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：

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

如果每个分类器独立犯错，则在一定条件下，随着集成分类器数目的增加，集成可以提高性能，集成的错误率将指数级下降

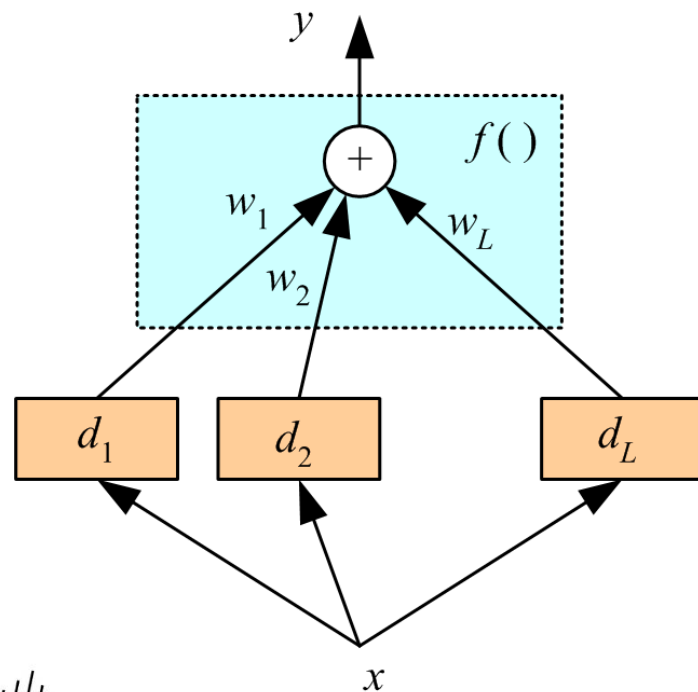
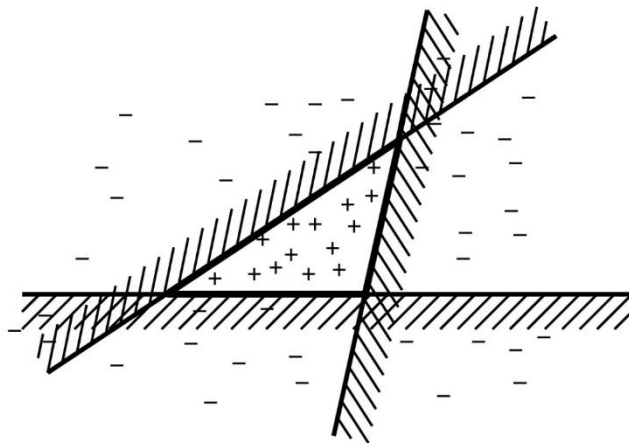
另外一种解释：不被单个分类器的随机误差影响，预测的误差将会降低

- 线性组合：平均/投票/加权

$$y = \sum_{j=1}^L w_j d_j$$

$$w_j \geq 0 \text{ and } \sum_{j=1}^L w_j = 1$$

- 分类为例



- 绝对多数投票法 (majority voting)

$$H(\mathbf{x}) = \begin{cases} c_j & \text{if } \sum_{i=1}^T h_i^j(\mathbf{x}) > \frac{1}{2} \sum_{k=1}^l \sum_{i=1}^T h_i^k(\mathbf{x}) \\ \text{rejection} & \text{otherwise.} \end{cases}$$

- ▣ 相对多数投票法 (plurality voting)

$$H(\mathbf{x}) = c_{\arg \max_j \sum_{i=1}^T h_i^j(\mathbf{x})}$$

- ▣ 加权投票法 (weighted voting)

$$H(\mathbf{x}) = c_{\arg \max_j \sum_{i=1}^T w_i h_i^j(\mathbf{x})}$$

# 几种典型的集成学习方法

- 有T个弱分类器  $y_m$  , 产生强分类器  $Y_M$

$$Y_M = \frac{1}{M} \sum_{m=1}^M y_m \quad \text{Bagging}$$

- 有T个弱分类器, 根据各处的权重, 产生强分类器

$$Y_M = \frac{1}{M} \sum_{m=1}^M \alpha_m y_m \quad \text{Boosting}$$

- 若进一步考虑弱分类器和样本进行自适应  
Adaptive Boosting Adaboost



# Bagging (Bootstrap aggregating)

## • 基本步骤

- 选取 $T$ 个bootstrap样例 (可重复选取)
- 在不同的bootstrap样例上训练得到 $T$ 个不同的分类器 (相互独立)
- 对于新的测试样例, 由 $T$ 个分类器分别预测, 并计算平均值 (或者多数投票)

---

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  $T$ .

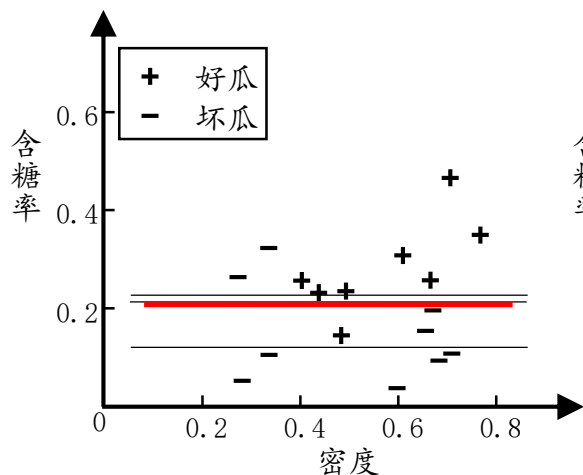
过程:

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

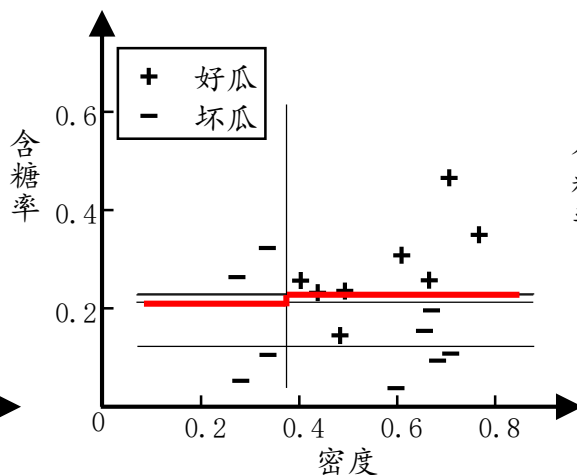
输出:  $H(\mathbf{x}) = \arg \max_{y \in \mathcal{Y}} \sum_{t=1}^T \mathbb{I}(h_t(\mathbf{x}) = y)$

---

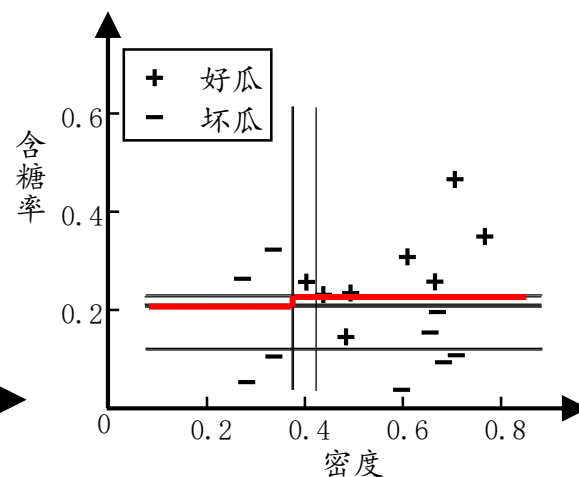
## Bagging与随机森林- Bagging实验



(a) 3个基学习器



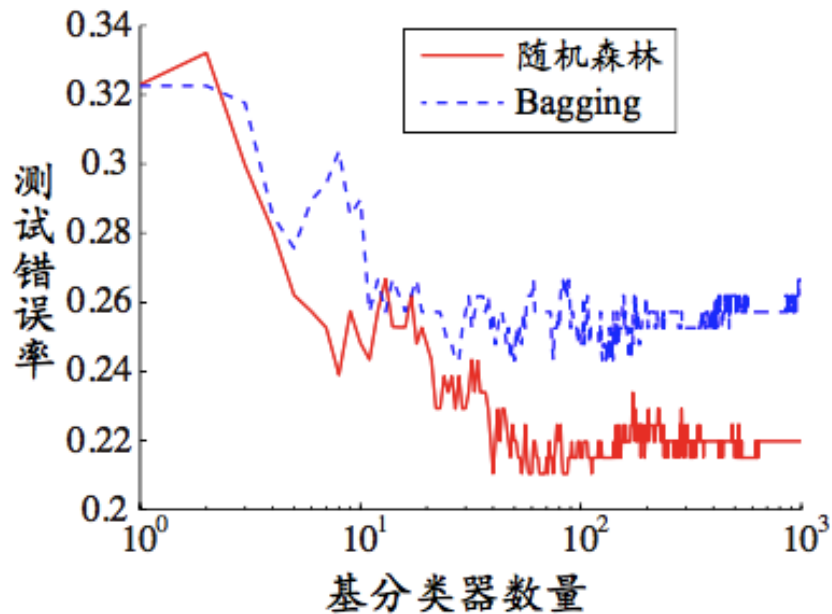
(b) 5个基学习器



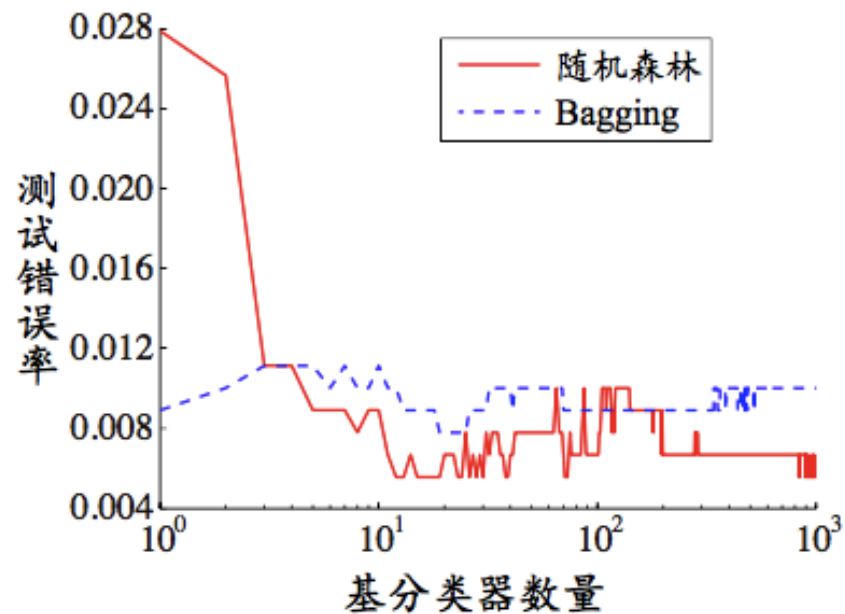
(c) 11个基学习器

从偏差-方差的角度：降低方差，在不剪枝的决策树、神经网络等易受样本影响的学习器上效果更好

# Bagging与随机森林实验对比



(a) glass 数据集

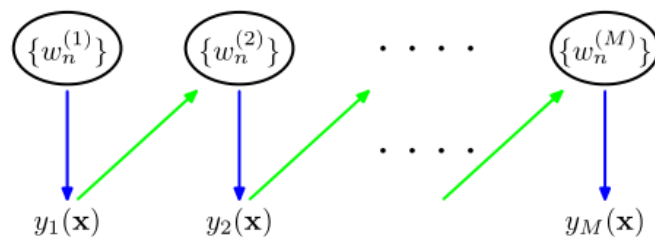


(b) auto-mpg 数据集

# Boosting

## • 基本步骤

- 顺序训练每个分类器
- 新的分类器主要集中在上一轮错误分类的样例上
- 组合所有得到的分类器预测结果



## • 特点

- 个体学习器存在强依赖关系
- 训练数据相同，但每次需要调整数据分布
- 每个分类器是“弱”的，但集成是“强”的

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

# Boosting基本思想

- 自适应对每个训练样例进行加权
- 被之前分类器错误分类的样例获得更高的权重
- 新的分类器在加权的数据上训练：新训练得到的分类器需要重点关注这些样例
- 对这些分类器进行加权形成最终的较强的单个分类器
  - 分类错误率低的分类器权重更大

# Boosting算法

**Input:** Sample distribution  $\mathcal{D}$ ;  
Base learning algorithm  $\mathcal{L}$ ;  
Number of learning rounds  $T$ .

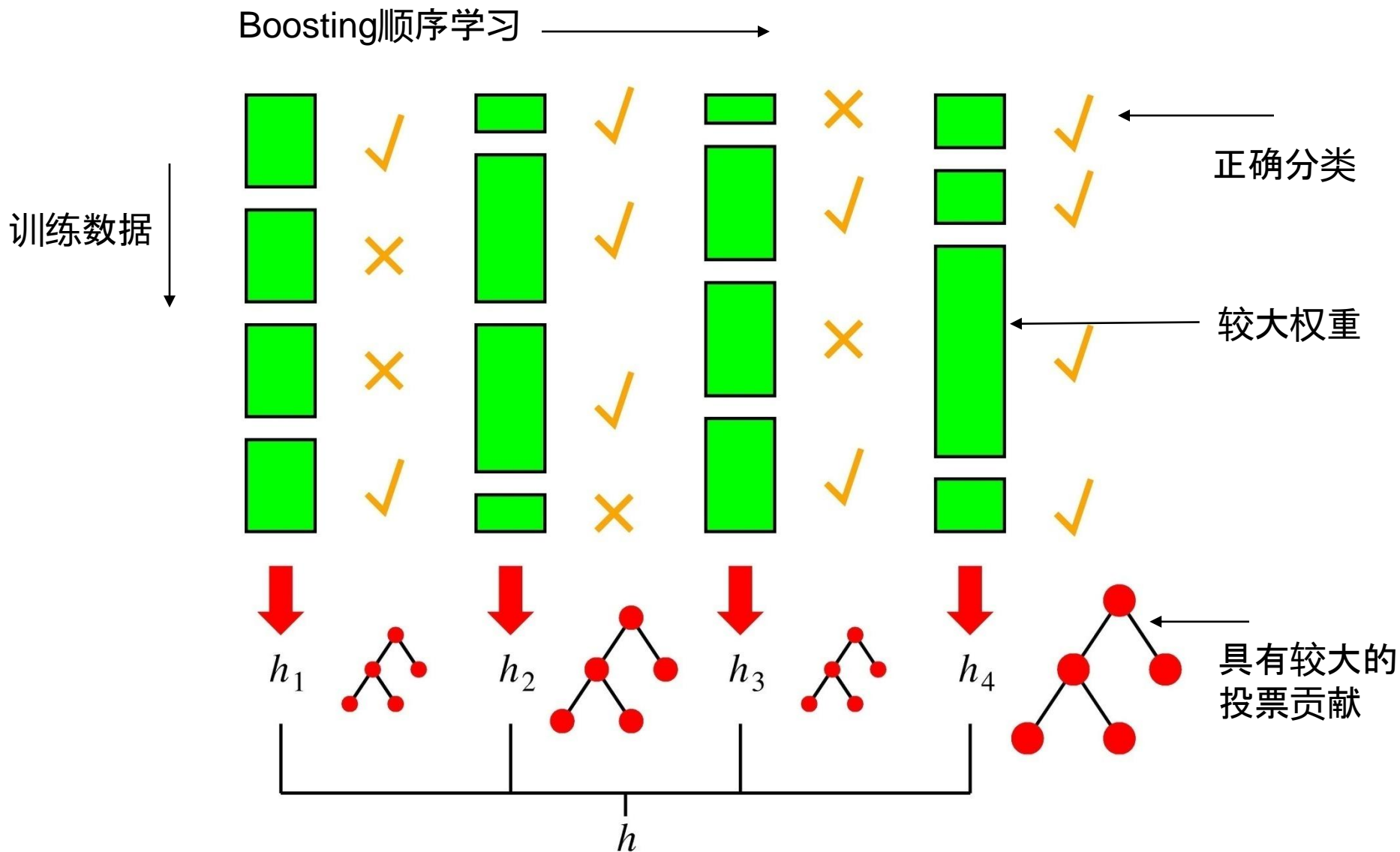
**Process:**

1.  $\mathcal{D}_1 = \mathcal{D}$ .     % Initialize distribution
2. **for**  $t = 1, \dots, T$ :
3.      $h_t = \mathcal{L}(\mathcal{D}_t)$ ;     % Train a weak learner from distribution  $\mathcal{D}_t$
4.      $\epsilon_t = P_{\mathbf{x} \sim \mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$ ;     % Evaluate the error of  $h_t$
5.      $\mathcal{D}_{t+1} = \text{Adjust\_Distribution}(\mathcal{D}_t, \epsilon_t)$
6. **end**

**Output:**  $H(\mathbf{x}) = \text{Combine\_Outputs}(\{h_1(\mathbf{x}), \dots, h_t(\mathbf{x})\})$

---

# Boosting实例





# Boosting – AdaBoost算法

Boosting算法中最著名的代表是AdaBoost

---

**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  $T$ .

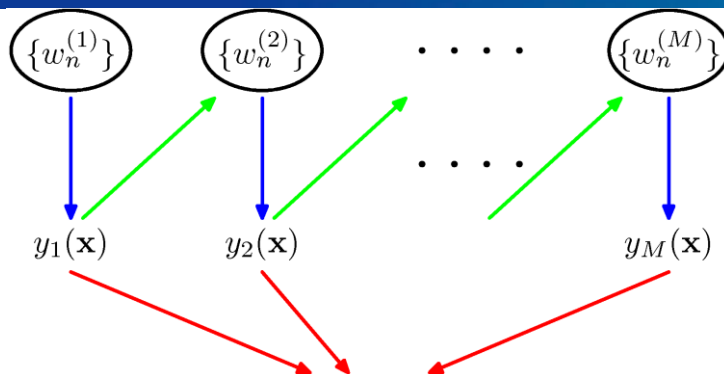
**过程:**

- 1:  $\mathcal{D}_1(\mathbf{x}) = 1/m$ .
- 2: **for**  $t = 1, 2, \dots, T$  **do**
- 3:    $h_t = \mathcal{L}(D, \mathcal{D}_t)$ ;
- 4:    $\epsilon_t = P_{\mathbf{x} \sim \mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$ ;
- 5:   **if**  $\epsilon_t > 0.5$  **then break**
- 6:    $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$ ;
- 7:   
$$\mathcal{D}_{t+1}(\mathbf{x}) = \frac{\mathcal{D}_t(\mathbf{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\mathbf{x}) = f(\mathbf{x}) \\ \exp(\alpha_t), & \text{if } h_t(\mathbf{x}) \neq f(\mathbf{x}) \end{cases}$$
$$= \frac{\mathcal{D}_t(\mathbf{x}) \exp(-\alpha_t f(\mathbf{x}) h_t(\mathbf{x}))}{Z_t}$$
- 8: **end for**

**输出:**  $H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$

---

# Boosting – AdaBoost推导



$$Y_M(\mathbf{x}) = \text{sign} \left( \sum_m^M \alpha_m y_m(\mathbf{x}) \right)$$

- 模型：基学习器的线性组合

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

- 损失函数：最小化指数损失函数

$$\ell_{\text{exp}}(H \mid \mathcal{D}) = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} [e^{-f(\mathbf{x})H(\mathbf{x})}]$$

# Boosting – AdaBoost推导

- 优化:

- 参数 $\alpha_t$ : 当基分类器 $h_t$ 基于分布 $D_t$ 产生后, 该基分类器的权重 $\alpha_t$ 应使得 $\alpha_t h_t$ 最小化指数损失函数

$$\begin{aligned}\ell_{\text{exp}}(\alpha_t h_t \mid \mathcal{D}_t) &= \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_t} \left[ e^{-f(\mathbf{x}) \alpha_t h_t(\mathbf{x})} \right] \\ &= \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_t} \left[ e^{-\alpha_t} \mathbb{I}(f(\mathbf{x}) = h_t(\mathbf{x})) + e^{\alpha_t} \mathbb{I}(f(\mathbf{x}) \neq h_t(\mathbf{x})) \right] \\ &= e^{-\alpha_t} P_{\mathbf{x} \sim \mathcal{D}_t}(f(\mathbf{x}) = h_t(\mathbf{x})) + e^{\alpha_t} P_{\mathbf{x} \sim \mathcal{D}_t}(f(\mathbf{x}) \neq h_t(\mathbf{x})) \\ &= e^{-\alpha_t} (1 - \epsilon_t) + e^{\alpha_t} \epsilon_t \quad \epsilon_t = P_{\mathbf{x} \sim \mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))\end{aligned}$$

- 对 $\alpha_t$ 求导为0

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

# Boosting – AdaBoost推导

## • 优化

- 在获得 $H_{t-1}$ 之后的样本分布进行调整, 使得下一轮的基学习器 $h_t$ 能纠正 $H_{t-1}$ 的一些错误, 理想的 $h_t$ 能纠正全部错误

$$h_t(\mathbf{x}) = \arg \min_h \ell_{\text{exp}}(H_{t-1} + h \mid \mathcal{D})$$

$$= \arg \min_h \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[ e^{-f(\mathbf{x})H_{t-1}(\mathbf{x})} \left( 1 - f(\mathbf{x})h(\mathbf{x}) \right) \right]$$

$$= \arg \max_h \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[ e^{-f(\mathbf{x})H_{t-1}(\mathbf{x})} f(\mathbf{x})h(\mathbf{x}) \right]$$

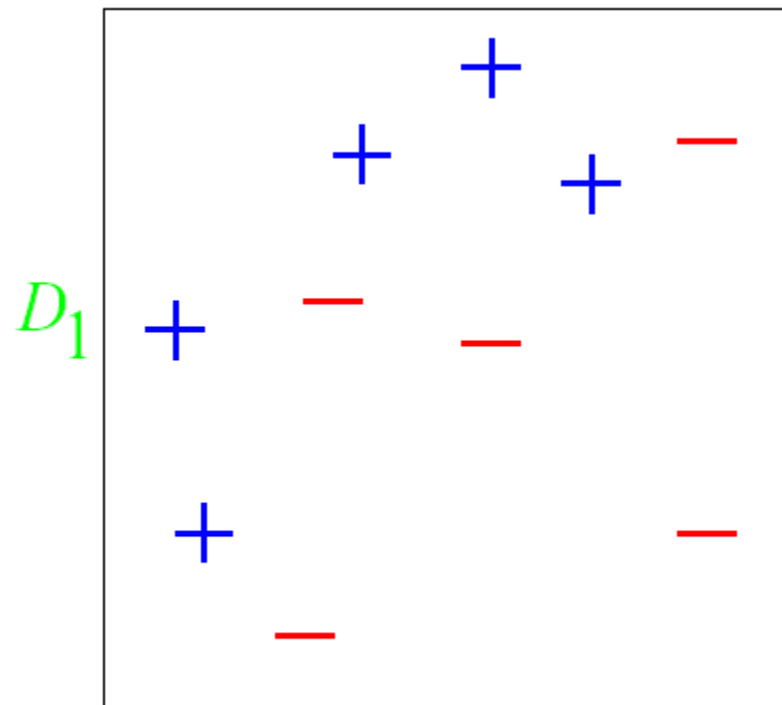
$$= \arg \max_h \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[ \frac{e^{-f(\mathbf{x})H_{t-1}(\mathbf{x})}}{\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[e^{-f(\mathbf{x})H_{t-1}(\mathbf{x})}]} f(\mathbf{x})h(\mathbf{x}) \right],$$

$$= \arg \max_h \mathbb{E}_{\mathbf{x} \sim \mathcal{D}_t} [f(\mathbf{x})h(\mathbf{x})] .$$

分类器

$$\begin{aligned} \mathcal{D}_{t+1}(\mathbf{x}) &= \frac{\mathcal{D}(\mathbf{x}) e^{-f(\mathbf{x})H_t(\mathbf{x})}}{\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[e^{-f(\mathbf{x})H_t(\mathbf{x})}]} \\ &= \frac{\mathcal{D}(\mathbf{x}) e^{-f(\mathbf{x})H_{t-1}(\mathbf{x})} e^{-f(\mathbf{x})\alpha_t h_t(\mathbf{x})}}{\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[e^{-f(\mathbf{x})H_t(\mathbf{x})}]} \\ &= \mathcal{D}_t(\mathbf{x}) \cdot e^{-f(\mathbf{x})\alpha_t h_t(\mathbf{x})} \frac{\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[e^{-f(\mathbf{x})H_{t-1}(\mathbf{x})}]}{\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[e^{-f(\mathbf{x})H_t(\mathbf{x})}]} \end{aligned}$$

# AdaBoost实例



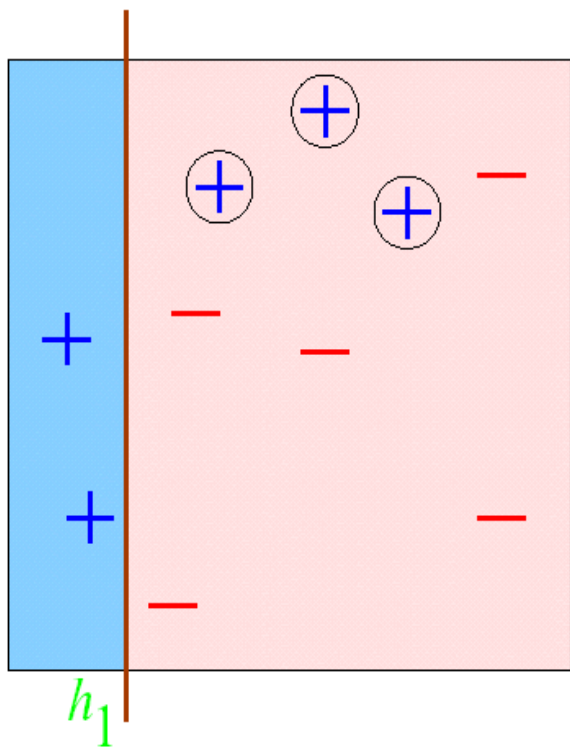
原始训练数据：所有样例等权重

# AdaBoost实例

$\varepsilon$  = 分类器错误率

$\alpha$  = 分类器权重

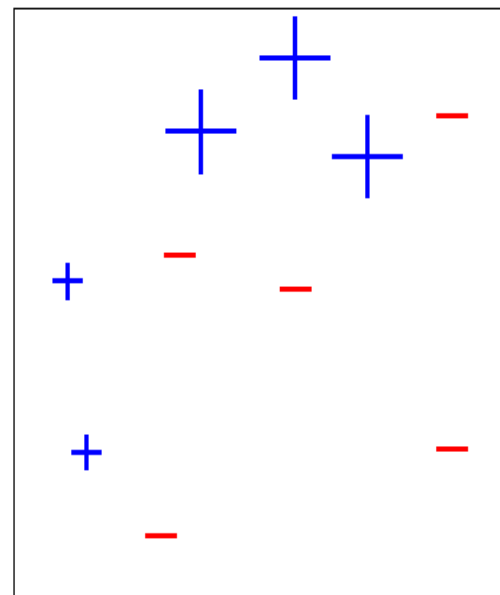
ROUND 1



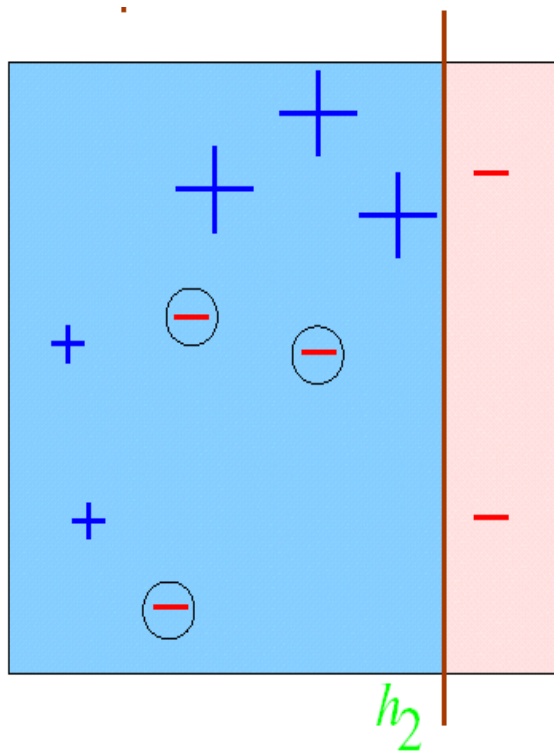
$$\varepsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$

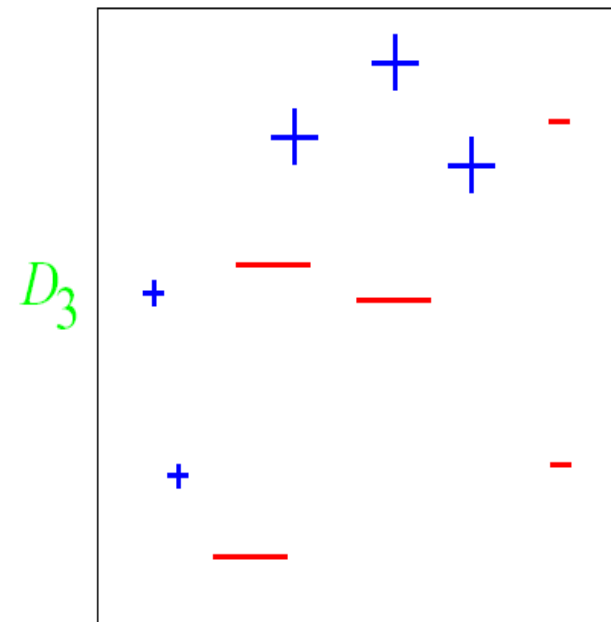
$D_2$



ROUND 2

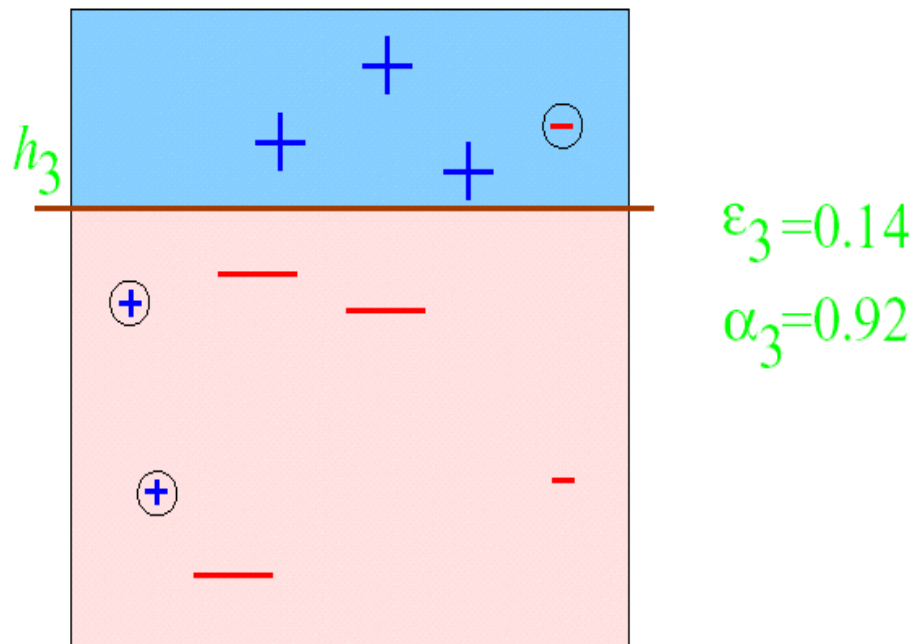


$$\epsilon_2 = 0.21$$
$$\alpha_2 = 0.65$$





## ROUND 3

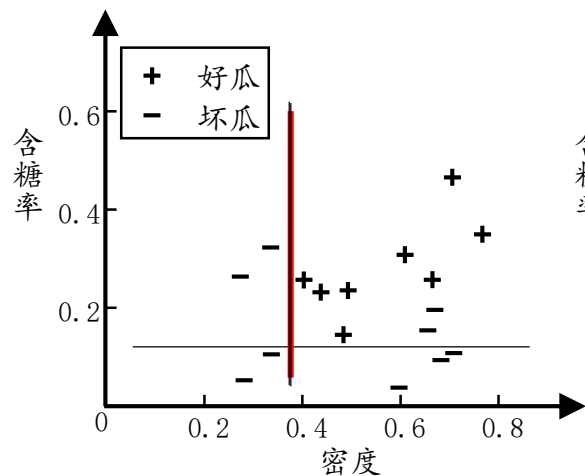


- 最终分类器

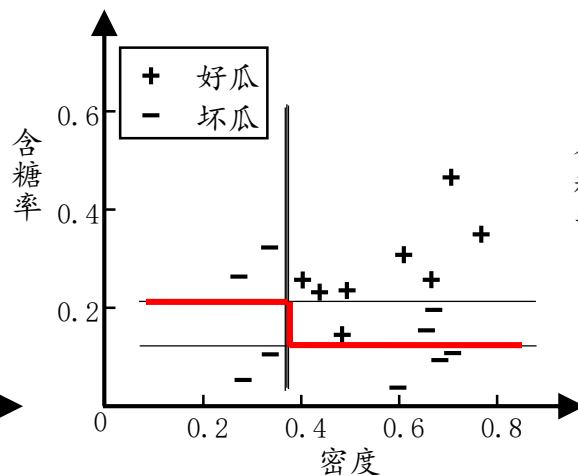
$$H_{\text{final}} = \text{sign} \left( 0.42 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \end{array} + 0.65 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \end{array} + 0.92 \begin{array}{|c|} \hline \text{blue} \\ \hline \text{red} \end{array} \right)$$

The diagram illustrates the final AdaBoost classifier  $H_{\text{final}}$ . It is represented as the sign of the weighted sum of three weak classifiers. Each weak classifier is shown as a square divided into two regions (blue and red) by a vertical line. The weights for each classifier are 0.42, 0.65, and 0.92. The final classifier is the sign of the weighted sum.

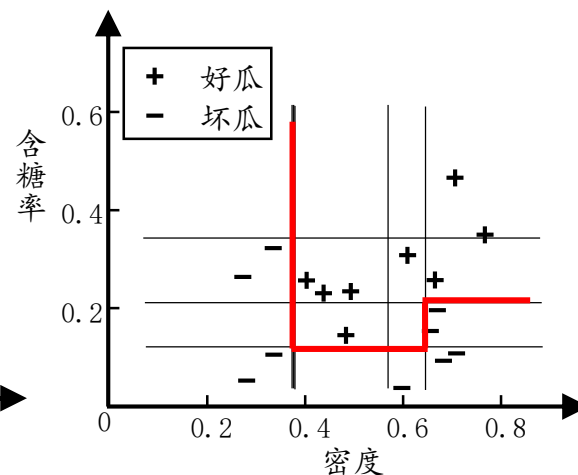
# AdaBoost性能



(a) 3个基学习器



(b) 5个基学习器



(c) 11个基学习器

- 从偏差-方差的角度：降低偏差，可对泛化性能相当弱的学习器构造出很强的集成

# Stacking方法

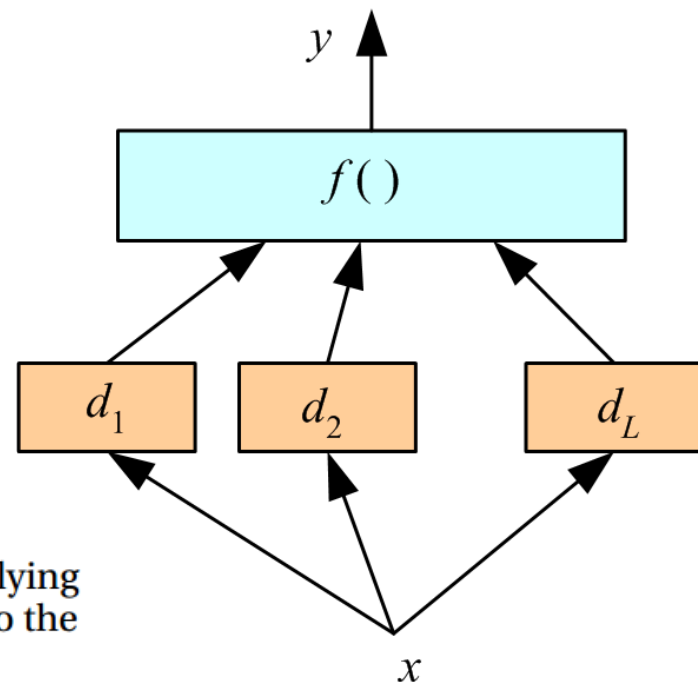
- 组合策略也是一个学习器 (Wolpert, 1992)

**Input:** Data set  $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ ;  
First-level learning algorithms  $\mathcal{L}_1, \dots, \mathcal{L}_T$ ;  
Second-level learning algorithm  $\mathcal{L}$ .

**Process:**

```
1. for  $t = 1, \dots, T$ :    % Train a first-level learner by applying the
2.    $h_t = \mathcal{L}_t(D)$ ;    % first-level learning algorithm  $\mathcal{L}_t$ 
3. end
4.  $D' = \emptyset$ ;          % Generate a new data set
5. for  $i = 1, \dots, m$ :
6.   for  $t = 1, \dots, T$ :
7.     $z_{it} = h_t(\mathbf{x}_i)$ ;
8.   end
9.    $D' = D' \cup ((z_{i1}, \dots, z_{iT}), y_i)$ ;
10. end
11.  $h' = \mathcal{L}(D')$ ;      % Train the second-level learner  $h'$  by applying
                        % the second-level learning algorithm  $\mathcal{L}$  to the
                        % new data set  $D'$ .
```

**Output:**  $H(\mathbf{x}) = h'(h_1(\mathbf{x}), \dots, h_T(\mathbf{x}))$



- 降维
- 矩阵分解
  - 主成分分析
  - 潜在语义分析
  - 谱分解
- 推荐算法
  - 协同过滤



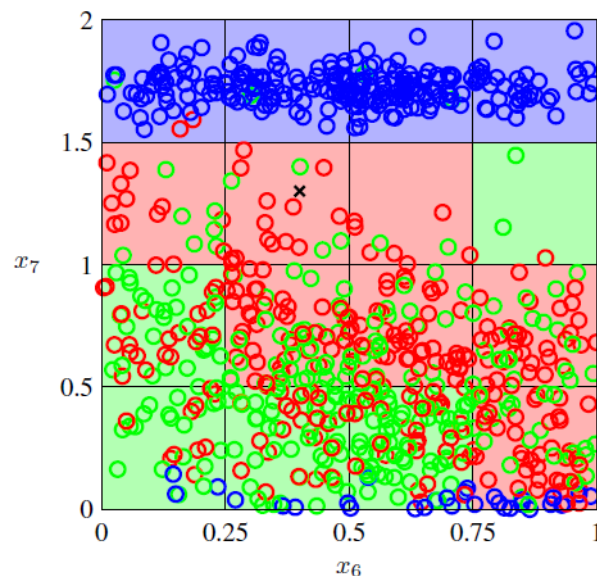
软件开发环境国家重点实验室  
State Key Laboratory of Software Development Environment

降维

# 维度灾难

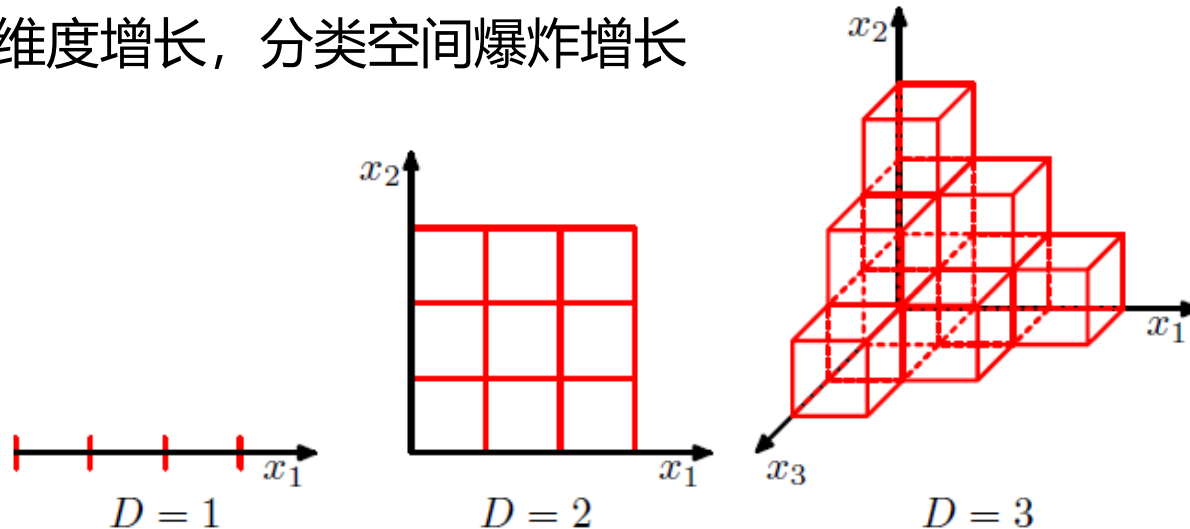
- 分类为例

- 最近邻分类方法
  - 以最近的格子：投票分类



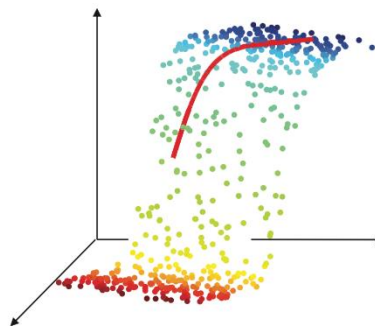
- 问题

- 当数据维度增长，分类空间爆炸增长

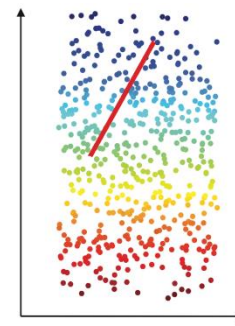




- 缓解维数灾难的一个重要途径是降维(dimension reduction)
  - 即通过某种数学变换，将原始高维属性空间转变为一个低维“子空间”(subspace)，在这个子空间中样本密度大幅度提高，距离计算也变得更为容易。
- 为什么能进行降维？
  - 数据样本虽然是高维的，但与学习任务密切相关的也许仅是某个低维分布，即高维空间中的一个低维“嵌入”(embedding)，因而可以对数据进行有效的降维。



(a) 三维空间中观察到的样本点



(b) 二维空间中的曲面



# 线性降维方法

- 对原始高维空间进行线性变换。给定 $d$ 维空间中的样本  $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m) \in \mathbb{R}^{d \times m}$ ，变换之后得到  $d' \leq d$  维空间中的样本

$$\mathbf{Z} = \mathbf{W}^T \mathbf{X},$$

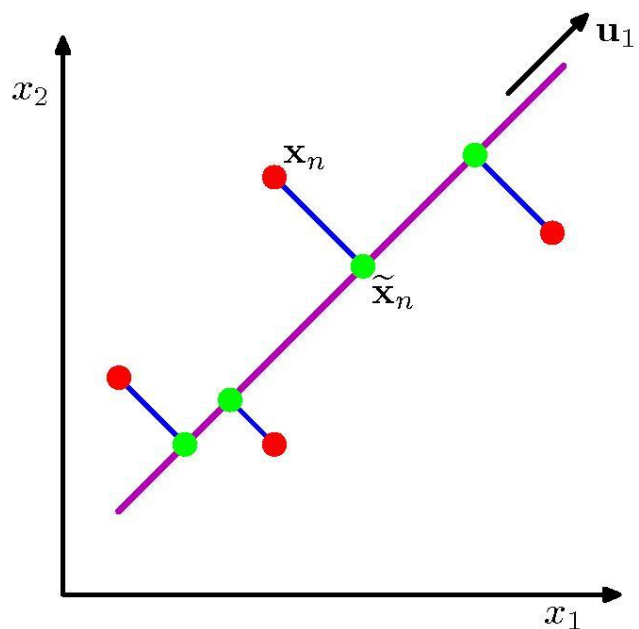
其中  $\mathbf{W} \in \mathbb{R}^{d \times d'}$  是变换矩阵,  $\mathbf{Z} \in \mathbb{R}^{d' \times m}$  是样本在新空间中的表达。

- 变换矩阵  $\mathbf{W}$  可视为  $d'$  个  $d$  维属性向量。换言之,  $z_i$  是原属性向量  $x_i$  在新坐标系  $\{w_1, w_2, \dots, w_{d'}\}$  中的坐标向量。若  $w_i$  与  $w_j (i \neq j)$  正交, 则新坐标系是一个正交坐标系, 此时  $\mathbf{W}$  为正交变换。
- 显然, 新空间中的属性是原空间中的属性的线性组合。

- 降维：对于正交属性空间中的样本点，如何用一个超平面对所有样本进行恰当的表达？
- 容易想到，若存在这样的超平面，那么它大概应具有这样的性质：
  - 使得降维后最大程度保持原始的数据特性：方差最大
  - 使得降维后数据的误差尽可能小：均方误差最小
- 能分别得到主成分分析的两种等价推导。

# 主成分分析-概述

- K. Pearson(1901年论文) 针对非随机变量
- H. Hotelling(1933年论文) 推广到随机向量
- 主成分分析(Principal Component Analysis, PCA), 将原有众多具有一定相关性的指标重新组合成一组少量互相无关的综合指标。



使得降维后样本的**方差尽可能大**

使得降维后数据的**均方误差尽可能小**

# 最大化方差

- 基本思想：使用较少的数据维度保留数据特性（方差）
- 将D维数据集  $\{\mathbf{x}_n\}, n = 1, 2, \dots, N$  降为  $M < D$ ，不失一般性，先考虑  $M = 1$ ，投影为  $\mathbf{u}_1$ ， $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- 模型：

- 每个数据点  $\mathbf{x}_n$  在新空间中表示为标量  $\mathbf{u}_1^T \mathbf{x}_n$
- 样本均值在新空间中表示为  $\mathbf{u}_1^T \bar{\mathbf{x}}$ ，其中  $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$

- 投影后样本方差表示为

$$\frac{1}{N} \sum_{n=1}^N \{\mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}}\}^2 = \boxed{\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1} \text{ 最大}$$

- 其中原样本方差  $\mathbf{S} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$

# 最大化方差

- 基本思想：使用较少的数据维度保留原数据特性

- 优化目标： $\max \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1, \text{ s.t. } \mathbf{u}_1^T \mathbf{u}_1 = 1$

- 求解：利用拉格朗日乘子法  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T \mathbf{u}_1)$

- 对  $\mathbf{u}_1$  求导置零得到

$$\mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \quad \mathbf{u}_1 \text{ 是 } \mathbf{S} \text{ 的特征向量}$$

- 进一步得到


$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1$$

$\mathbf{u}_1$  是  $\mathbf{S}$  最大特征值对应的特征向量时  
方差取到极大值，称  $\mathbf{u}_1$  为第一主成分

- 考虑更一般性的情况(  $M > 1$  )：新空间中数据方差最大的最佳投影方向由协方差矩阵S的M个特征向量定义： $\mathbf{u}_1, \dots, \mathbf{u}_M$  , 其分别对应M个最大的特征值  $\lambda_1, \dots, \lambda_M$
- 算法：
  - 首先获得方差最大的1维，生成该维的补空间；
  - 继续在补空间中获得方差最大的1维，生成新的补空间；
  - 依次循环下去得到M维的空间。

# 主成分分析

- 定义一组正交的D维基向量  $\{\mathbf{u}_i\}, i = 1, \dots, D$  ,  
满足  $\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$
- 由于基是完全的, 每个数据点可以表示为基向量的线性组合
- 相当于进行了坐标变换  $\mathbf{x}_n = \sum_{i=1}^D \alpha_{ni} \mathbf{u}_i$

$$\{\mathbf{x}_{n1}, \dots, \mathbf{x}_{nD}\} \xrightarrow{\{\mathbf{u}_i\}} \{\alpha_{n1}, \dots, \alpha_{nD}\}$$


$$\alpha_{nj} = \mathbf{x}_n^T \mathbf{u}_j$$

# 最小化误差

- 基本思想：使原数据与降维后的数据(在原空间中的重建)的误差最小

- 在M维变量(  $M < D$  )生成的空间中对其进行表示

$$\begin{aligned}\mathbf{x}_n &= \sum_{i=1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i \\ \tilde{\mathbf{x}}_n &= \sum_{i=1}^M z_{ni} \mathbf{u}_i + \sum_{i=M+1}^D b_i \mathbf{u}_i\end{aligned}\quad \begin{aligned}z_{nj} &= \mathbf{x}_n^T \mathbf{u}_j, j = 1, \dots, M \\ b_j &= \bar{\mathbf{x}}^T \mathbf{u}_j, j = M + 1, \dots, D\end{aligned}$$

- 优化目标：最小化失真度

$$J = \frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2$$



# 最小化误差

- 优化目标：最小化失真度

$$\mathbf{x}_n - \tilde{\mathbf{x}}_n = \sum_{i=M+1}^D \{(\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i\} \mathbf{u}_i$$

$$J = \frac{1}{N} \sum_{n=1}^N \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i - \bar{\mathbf{x}}^T \mathbf{u}_i)^2 = \sum_{i=M+1}^D \mathbf{u}_i^T \mathbf{S} \mathbf{u}_i$$

- 优化：拉格朗日乘子法

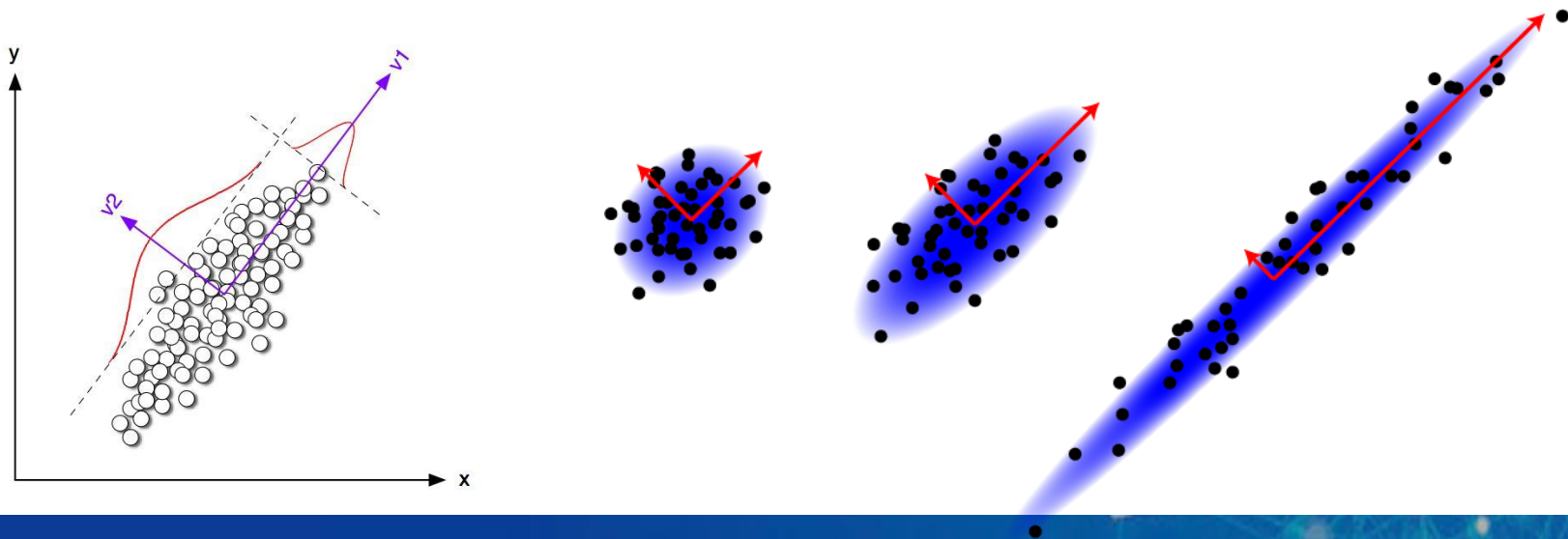
$$\tilde{J} = \sum_{i=M+1}^D \mathbf{u}_i^T \mathbf{S} \mathbf{u}_i + \sum_{i=M+1}^D \lambda_i (1 - \mathbf{u}_i^T \mathbf{u}_i)$$

□ 求导得到  $\mathbf{S} \mathbf{u}_i = \lambda_i \mathbf{u}_i$   **$J$ 最小时取 $D-M$ 个最小的特征值**

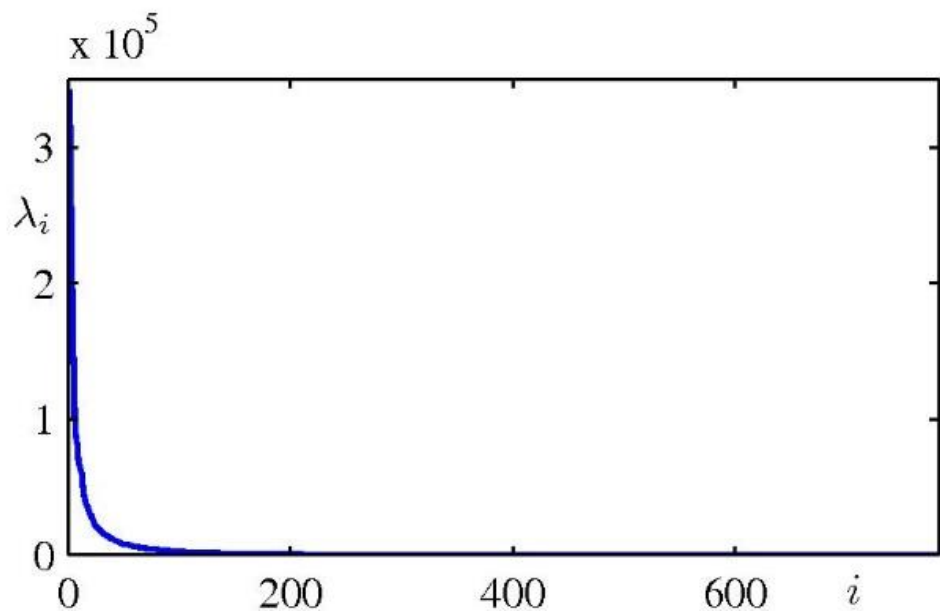
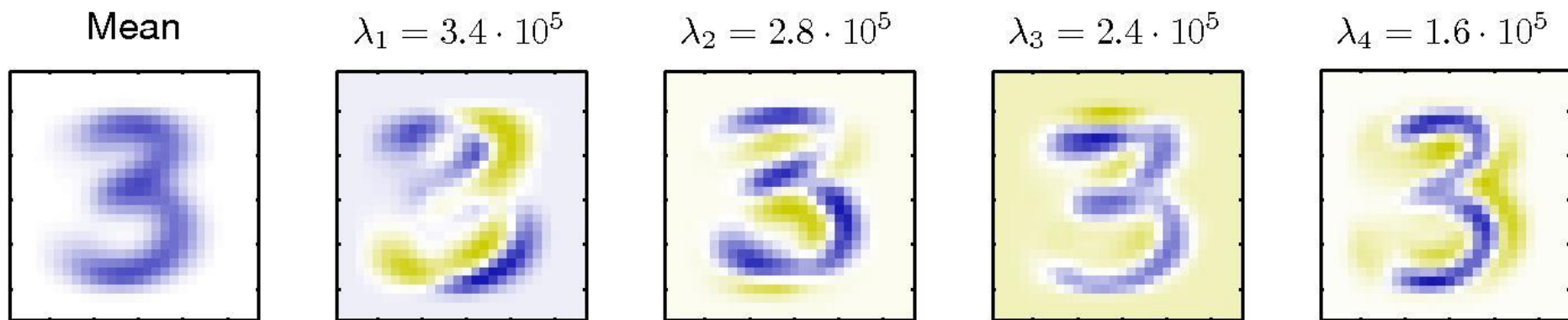
□ 对应失真度为  $J = \sum_{i=M+1}^D \lambda_i$  **主子空间对应 $M$ 个最大特征值**

# 主成分分析-算法

- 计算步骤
- ①计算给定样本  $\{\mathbf{x}_n\}, n = 1, 2, \dots, N$  的均值  $\bar{\mathbf{x}}$  和协方差矩阵  $S$ ;
- ②计算  $S$  的特征向量与特征值,  $X = U\Lambda U^T$ ;
- ③将特征值从大到小排列, 前  $M$  个特征值  $\lambda_1, \dots, \lambda_M$  所对应的特征向量  $\mathbf{u}_1, \dots, \mathbf{u}_M$  构成投影矩阵。



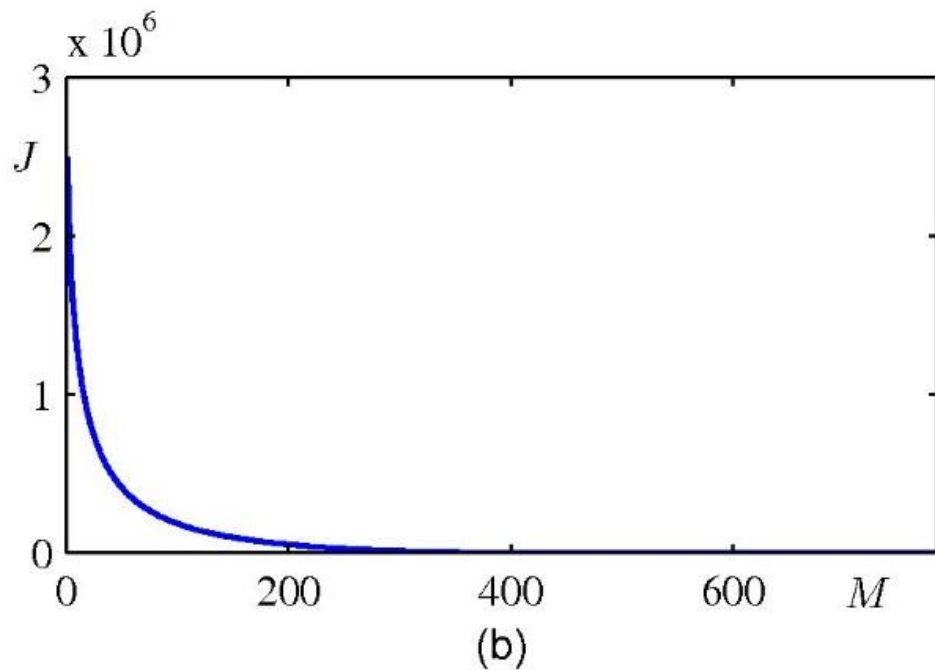
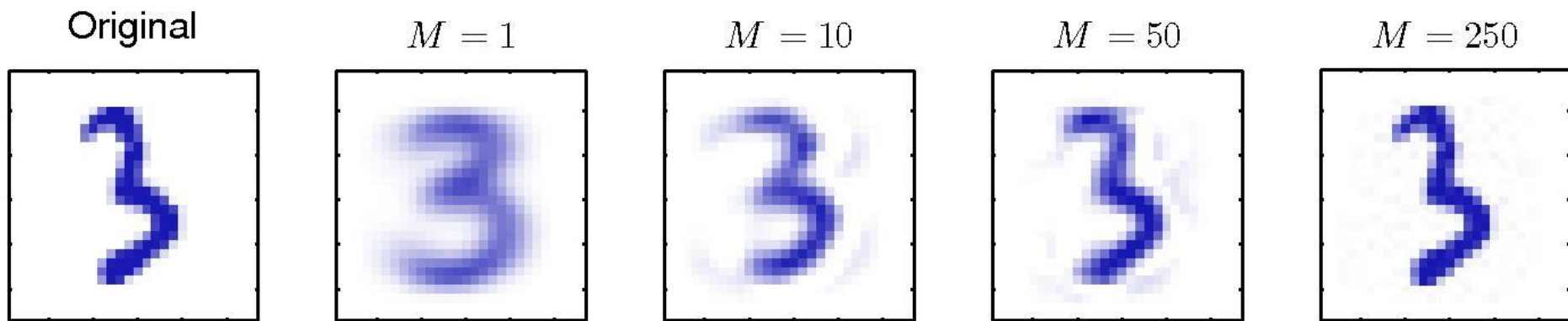
# 主成分分析-应用



(a)

特征值分布谱  
特征值由大到小排列

# 主成分分析-应用

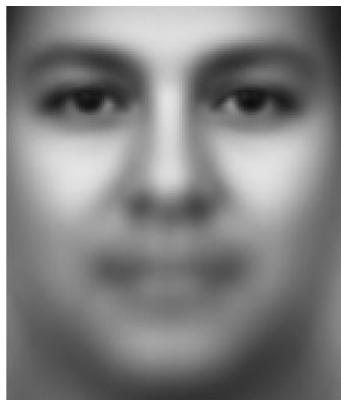


失真度分布谱  
随 $M$ 取值由小到大排列



特征脸(Eigenfaces)#1~#8

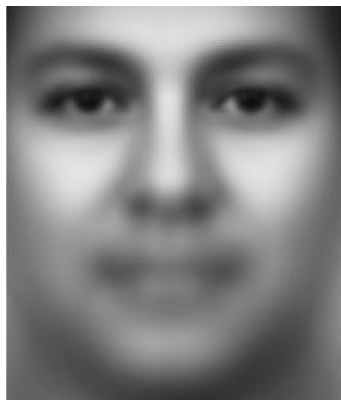




特征脸(Eigenfaces)#101~#108







特征脸(Eigenfaces)#501~#508



# 核主成分分析 ( Kernel PCA )

- 将主成分分析的线性假设一般化使之适应非线性数据
- 传统PCA: D维样本  $\{\mathbf{x}_n\}, n = 1, 2, \dots, N$  ,  $\sum_n \mathbf{x}_n = \mathbf{0}$

$$\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i \quad \mathbf{S} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \quad \mathbf{u}_i^T \mathbf{u}_i = 1$$

- 核PCA: 非线性映射  $\phi(\mathbf{x})$  ,  $\mathbf{x}_n \mapsto \phi(\mathbf{x}_n)$  ,  $\sum_n \phi(\mathbf{x}_n) = \mathbf{0}$

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \mathbf{C} = \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T$$

$$\longrightarrow \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \{ \phi(\mathbf{x}_n)^T \mathbf{v}_i \} = \lambda_i \mathbf{v}_i$$



# 核主成分分析

• 新的数据空间下  $\mathbf{v}_i = \sum_{n=1}^N a_{in} \phi(\mathbf{x}_n)$

$$\frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T \sum_{m=1}^N a_{im} \phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^N a_{in} \phi(\mathbf{x}_n)$$

$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$$

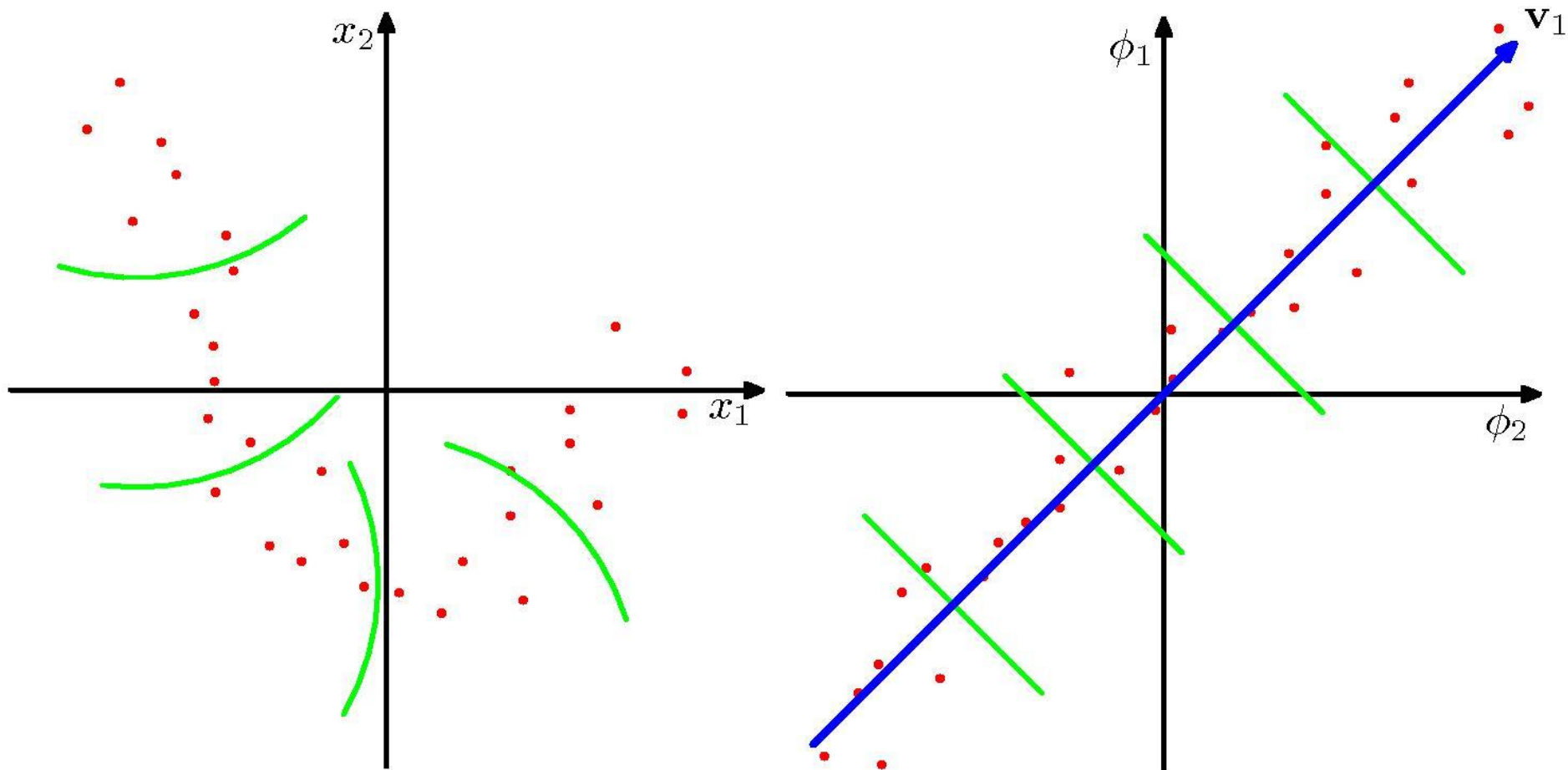
$$\longrightarrow \frac{1}{N} \sum_{n=1}^N k(\mathbf{x}_l, \mathbf{x}_n) \sum_{m=1}^N a_{im} k(\mathbf{x}_n, \mathbf{x}_m) = \lambda_i \sum_{n=1}^N a_{in} k(\mathbf{x}_l, \mathbf{x}_n)$$

$$\longrightarrow \mathbf{K}^2 \mathbf{a}_i = \lambda_i N \mathbf{K} \mathbf{a}_i$$

$$\longrightarrow \mathbf{K} \mathbf{a}_i = \lambda_i N \mathbf{a}_i$$

# 核主成分分析

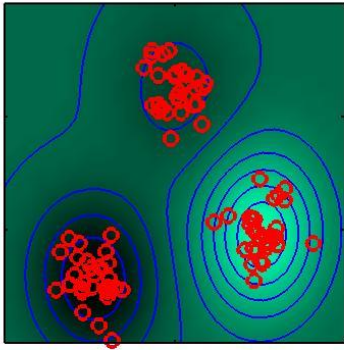
- Kernel PCA



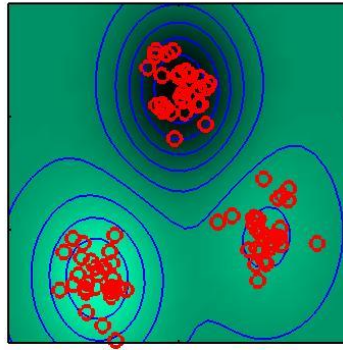
# 核主成分分析

- Kernel PCA

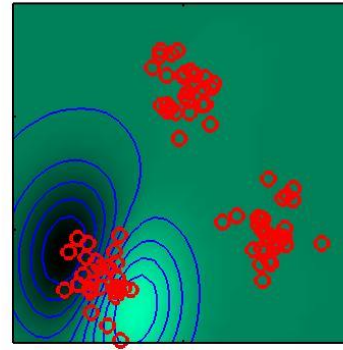
Eigenvalue=21.72



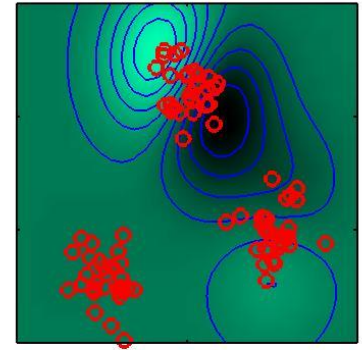
Eigenvalue=21.65



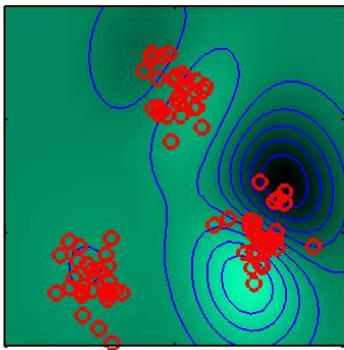
Eigenvalue=4.11



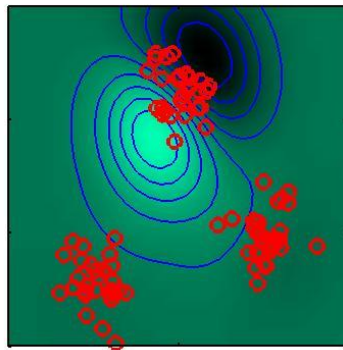
Eigenvalue=3.93



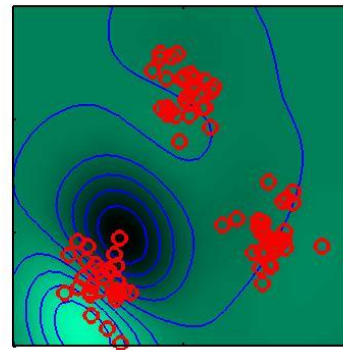
Eigenvalue=3.66



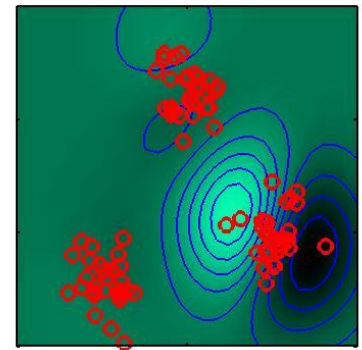
Eigenvalue=3.09



Eigenvalue=2.60



Eigenvalue=2.53





软件开发环境国家重点实验室  
State Key Laboratory of Software Development Environment

本节课结束