5月21日12:00前，提交文献阅读相关素材
6月3日12:00前，提交实验报告及相关素材

信息检索与数据挖掘

补充：数据挖掘经典算法概述

4月29日，补充：概率图及主题模型
5月6日，补充：数据挖掘经典算法概述(1)
5月8日，补充：数据挖掘经典算法概述(2)
5月13日，第12章 Web搜索
5月15日，第13章 多媒体信息检索
5月20日，复习
5月22日，同学们文献阅读报告
5月27日，同学们文献阅读报告
6月3日，期末考试【暂定】

名词演化

- **数据挖掘（data mining）**
- **数据库的知识发现（KDD, Knowledge Discovery in Database）**
- **模式识别（pattern recognition）**
- **人工智能（Artificial intelligence）**
- **机器学习（machine learning）**
- **统计机器学习（statistical learning）**

Top 10 algorithms in data mining, 2007

This paper presents the top 10 data mining algorithms identified by the IEEE International Conference on Data Mining (ICDM) in December 2006: **C4.5, *k*-Means, SVM, Apriori, EM, PageRank, AdaBoost, *k*NN, Naive Bayes,** and **CART**. These top 10 algorithms are among the most influential data mining algorithms in the research community……

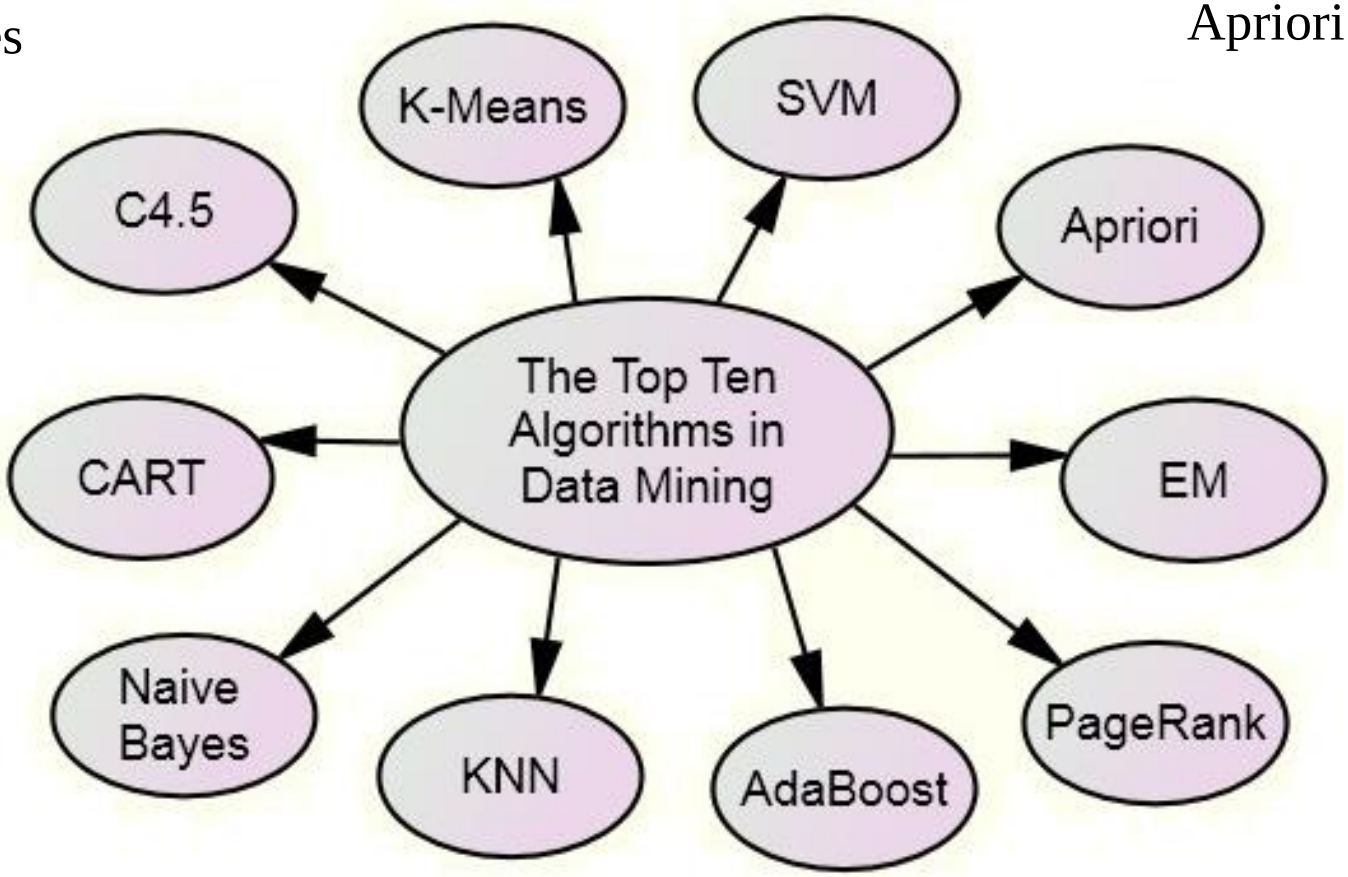
These 10 algorithms cover classification, clustering, statistical learning, association analysis, and link mining, which are all among the most important topics in data mining research and development.

Xindong Wu, Vipin Kumar, J. Ross Quinlan, et al. **Top 10 algorithms in data mining.** Knowledge and Information Systems, 14(1):1–37, 2008. Published online: 4 December 2007.

数据挖掘十大经典算法

C4.5
CART
AdaBoost
Apriori

Naive Bayes
kNN
k-Means
SVM
EM



Pagerank ⊂ 第12章Web搜索

[1] Xindong Wu, Vipin Kumar, J. Ross Quinlan, et al. Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1):1–37, 4 December 2007.

今日内容：数据挖掘经典算法概述

- **教材中有的**
  - Naive Bayes、EM、K-means、SVM、kNN
- **决策树**
  - ID3、C4.5、CART
- **把若干个分类器整合为一个分类器**
  - Bagging
  - Boosting
  - AdaBoost，1995
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、 K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

《Machine Learning》, Tom M.Mitchell, 1997, 第3章例子

示例：训练集、测试集

训练集

outlook	temperature	humidity	windy	play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

统计了14天的气象数据(指标包括outlook，temperature，humidity，windy)，并已知这些天气是否打球(play)。如果给出新一天的气象指标数据:sunny,cool,high,TRUE，判断一下会不会去打球。

这是个二分类问题

测试集

outlook	sunny
temperature	cool
humidity	high
windy	FALSE

朴素贝叶斯分类求解

回顾：Naive Bayes text classification

- 在文本分类中，我们的目标是找出文档最可能属于的类别。对于NB 分类来说，最可能的类是具有MAP估计值的结果c_{map}：

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c)$$

- 如何估计参数P^(c)及P^(t_k |c)？

$$\hat{P}(c) = \frac{N_c}{N} \qquad \hat{P}(t \mid c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- 零概率问题→平滑

$$\hat{P}(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'}) + B}$$

朴素贝叶斯分类器→概率图

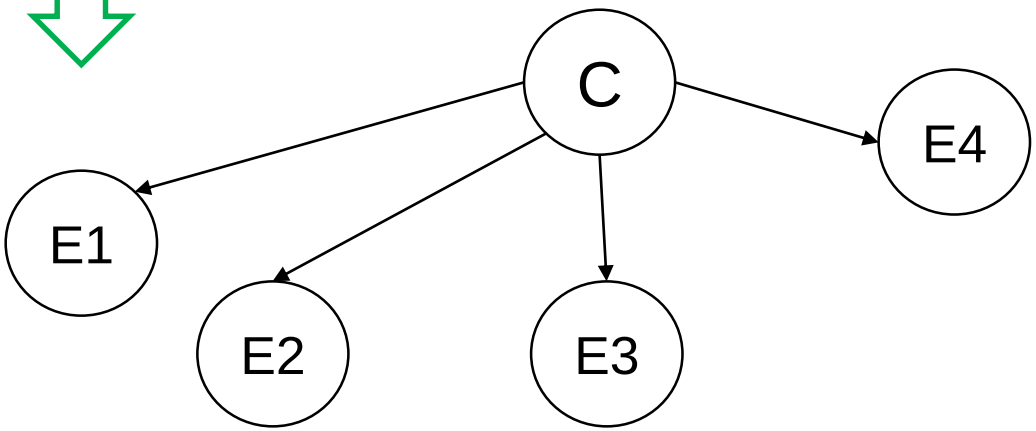
E1(outlook): sunny, overcast, rainy
E2(temperature): hot, mild, cool
E3(humidity): high, normal
E4(windy): FALSE, TRUE
C(play): no, yes

独立性的假设

$$P(C,E1,E2,E3,E4)=P(C)P(E1|C)P(E2|C)P(E3|C)P(E4|C)$$

$$c_{map} = \arg\max_{c\in\mathbb{C}} \hat{P}(c|d) = \arg\max_{c\in\mathbb{C}} \hat{P}(c) \prod_{1\le k\le n_d} \hat{P}(t_k|c)$$

Graphical Model



朴素贝叶斯分类求解

E1:outlook			E2: temperature			E3:humidity			E4:windy			C:play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								

Bayes公式：P(c|e) → P(c)P(e|c)

e = {e1=sunny, e2=cool, e3=high, e4=false}

测试集

outlook	sunny
temperature	cool
humidity	high
windy	FALSE

$P(c=yes|e) \propto P(c=yes)P(E1|c=yes)P(E2|c=yes)\ P(E3|c=yes)\ P(E4|c=yes)$

$P(c=no|e) \propto P(c=no)P(E1|c=no)P(E2|c=no)\ P(E3|c=no)\ P(E4|c=no)$

$P(c=yes|e)*P(E)=9/14\times 2/9\times 3/9\times 3/9\times 3/9=0.0053$

$P(c=no|e)*P(E)=5/14\times 3/5\times 1/5\times 4/5\times 3/5=0.0206$

⇨ c=no

今日内容：数据挖掘经典算法概述

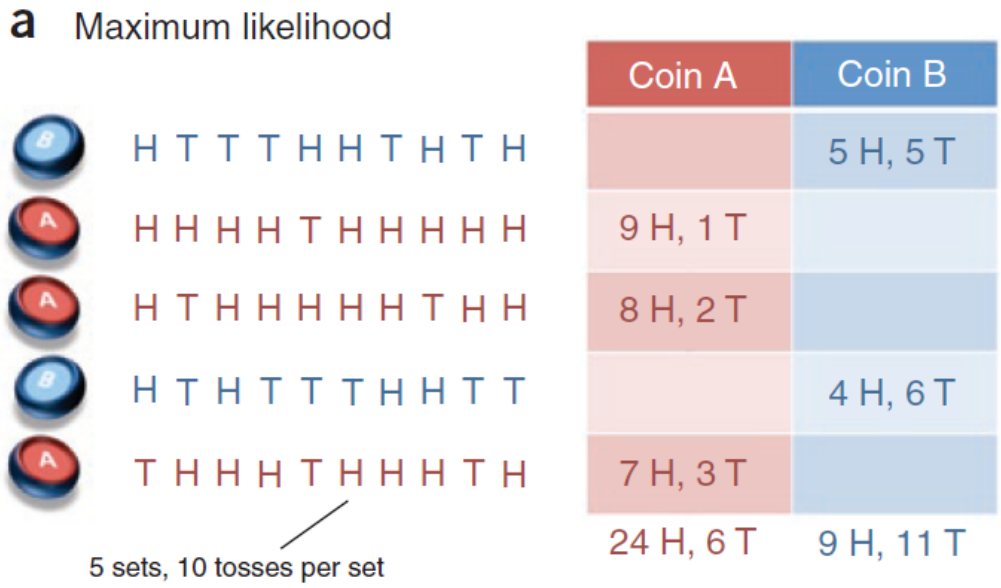
- 教材中有的
  - Naive Bayes、 EM、K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

EM算法示例：求两个硬币随机抛出后正反面的概率

完全信息下的MLE估计

$x = (x_1, x_2, \ldots, x_5)$, $x_i \in \{0,1,\ldots,10\}$ 第i次抛硬币试验中正面(head)朝上的次数

$z = (z_1, z_2, \ldots, z_5)$, $z_i \in \{A,B\}$ 第i次抛硬币试验中被抛掷的硬币是A还是B



x=(5,9,8,4,7)

z=(B,A,A,B,A)

$\hat{\theta}_A = \frac{24}{24+6} = 0.80$

$\hat{\theta}_B = \frac{9}{9+11} = 0.45$

MLE：Find parameters $\hat{\theta} = (\hat{\theta}_A, \hat{\theta}_B)$ that maximize logP(x,z;θ).

找到使得logP(x,z;θ)最大的参数 θ，logP(x,z ; θ) 分号左边是随机变量，右边是模型参数

$$\hat{\theta}_A = \frac{\text{硬币}A\text{正面朝上的次数}}{\text{硬币}A\text{抛的总次数}}, \quad \hat{\theta}_B = \frac{\text{硬币B正面朝上的次数}}{\text{硬币B抛的总次数}}$$

C. B. Do and S. Batzoglou, "What is the expectation maximization algorithm?," *Nature Biotechnology,* vol. 26, p. 897, 08/01/online 2008.

EM算法示例：求两个硬币随机抛出后正反面的概率

不完全信息：不知道每次抛的是A还是B

如果是A硬币，抛出HHHHTHHHHH的概率是$\hat{\theta}_A{}^9(1-\hat{\theta}_A)^1=0.6^9\times0.4^1=0.010077696\times0.4\approx 0.004$

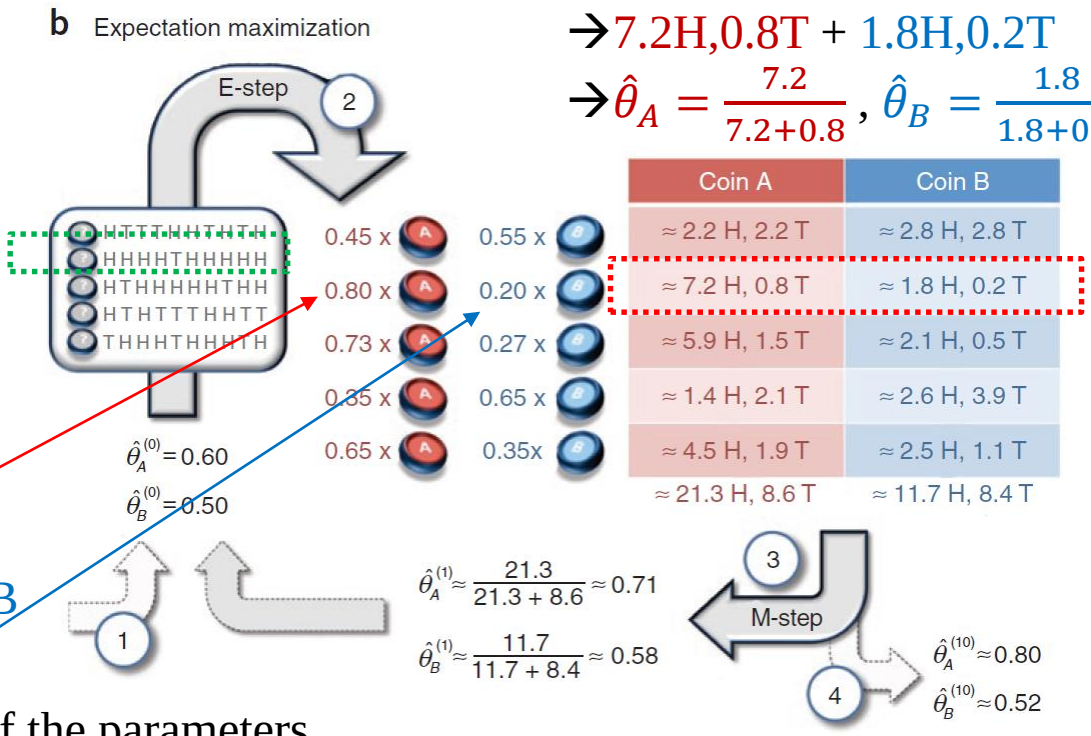
如果是B硬币，抛出HHHHTHHHHH的概率是$\hat{\theta}_B{}^9(1-\hat{\theta}_B)^1=0.5^{10}\approx 0.001$

故这次试验是硬币A的期望为0.004/(0.004+0.001)=0.80，是硬币B的期望为0.004/(0.004+0.001)=0.20

9H1T→0.80*9H1T + 0.20*9H1T

→7.2H,0.8T + 1.8H,0.2T

→$\hat{\theta}_A=\frac{7.2}{7.2+0.8}$, $\hat{\theta}_B=\frac{1.8}{1.8+0.2}$



1. EM starts with an initial guess of the parameters.
2. In the E-step, a probability distribution over possible completions is computed using the current parameters. The counts shown in the table are the expected numbers of heads and tails according to this distribution.
3. In the M-step, new parameters are determined using the current completions.
4. After several repetitions of the E-step and M-step, the algorithm converges.

C. B. Do and S. Batzoglou, "What is the expectation maximization algorithm?," *Nature Biotechnology,* vol. 26, p. 897, 08/01/online 2008.

EM算法示例：求两个硬币随机抛出后正反面的概率迭代过程分析

$x = (x_1, x_2, \ldots, x_5)$, $x_i \in \{0,1,\ldots,10\}$ 第i次抛硬币试验中正面(head)朝上的次数

$z = (z_1, z_2, \ldots, z_5)$, $z_i \in \{A,B\}$ 第i次抛硬币试验

$x=(x_1=5,x_2=9,x_3=8,x_4=4,x_5=7)$

Z是隐变量

②求隐变量Z的后验概率

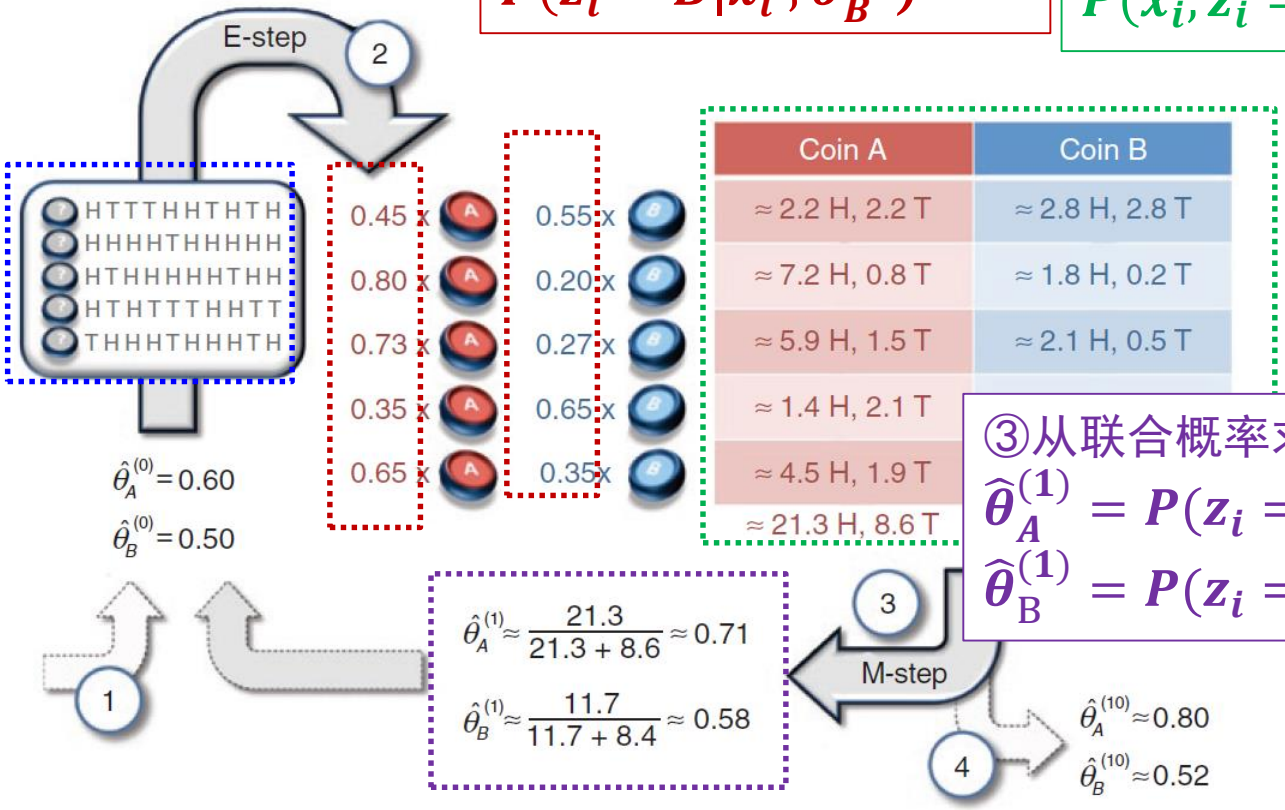
$$P(z_i = A|x_i ; \hat{\theta}_A^{(0)})$$

$$P(z_i = B|x_i ; \hat{\theta}_B^{(0)})$$

X和Z的联合概率

$$P(x_i, z_i = A|x_i ; \hat{\theta}_A^{(0)})$$

$$P(x_i, z_i = B|x_i ; \hat{\theta}_B^{(0)})$$



③从联合概率求边缘概率

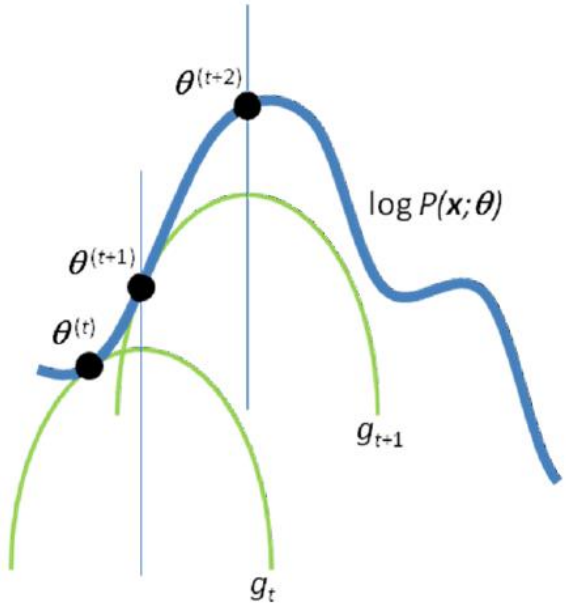
$$\hat{\theta}_A^{(1)} = P(z_i = A|x_i ; \hat{\theta}_A^{(0)})$$

$$\hat{\theta}_B^{(1)} = P(z_i = B|x_i ; \hat{\theta}_B^{(0)})$$

C. B. Do and S. Batzoglou, "What is the expectation maximization algorithm?," *Nature Biotechnology*, vol. 26, p. 897, 08/01/online 2008.

EM算法：迭代过程不断构造更好的下界

先固定当前参数，计算得到当前隐变量分布的一个下界(Lower Bound)函数，然后优化这个函数，得到新的参数，然后循环继续。



$p(X;\theta) = \sum_Z p(X,Z;\theta)$ ←从联合概率计算边缘概率

$$p(X;\theta) = \sum_Z q(Z)\frac{p(X,Z;\theta)}{q(Z)}, \quad \leftarrow q(Z) = p(Z|X;\hat{\theta}^t)，构造的先验分布$$

$$\log p(X;\theta) = \log \sum_Z q(Z)\frac{p(X,Z;\theta)}{q(Z)} \geq \sum_Z q(Z)\log\frac{p(X,Z;\theta)}{q(Z)} \leftarrow Jensen's\ inequality$$

$$g_t(\theta) \triangleq \sum_Z q(Z)\log\frac{p(X,Z;\theta)}{q(Z)} = \sum_Z p\left(Z|X;\hat{\theta}^t\right)\log\frac{p(X,Z;\theta)}{p\left(Z|X;\hat{\theta}^t\right)}$$

$\hat{\theta}^{(t+1)} = \underset{\theta}{\operatorname{argmax}}\, g_t(\theta)$ ← $g_t(\theta)$是$\log p(X;\theta)$的lower bound

C. B. Do and S. Batzoglou, "What is the expectation maximization algorithm?," *Nature Biotechnology*, vol. 26, p. 897, 08/01/online 2008.
关于EM算法的九层境界的浅薄介绍 http://www.elecfans.com/d/604076.html

EM算法：迭代逼近最优解过程分析

$p(X;\theta) = \sum_Z p(X,Z;\theta)$ ←从联合概率计算边缘概率

$$p(X;\theta) = \sum_Z q(Z)\frac{p(X,Z;\theta)}{q(Z)}, \quad \leftarrow q(Z) = p(Z|X;\hat{\theta}^t)，\text{构造的先验分布}$$

$$\log p(X;\theta) = \log \sum_Z q(Z)\frac{p(X,Z;\theta)}{q(Z)} \geq \sum_Z q(Z)\log\frac{p(X,Z;\theta)}{q(Z)} \leftarrow Jensen's\ inequality$$

$$\log p(X;\theta) \geq \sum_Z q(Z;\hat{\theta}^t)\log\frac{p(X,Z;\hat{\theta}^t)}{q(Z;\hat{\theta}^t)}$$

$$\log p(X;\theta) \geq \sum_Z p(Z|X;\hat{\theta}^t)\log\frac{p(X,Z;\hat{\theta}^t)}{p(Z|X;\hat{\theta}^t)}$$

$$\log p(X;\theta) \geq \sum_Z p(Z|X;\hat{\theta}^t)\log\frac{p(X,Z;\hat{\theta}^t)}{p(Z|X;\hat{\theta}^*)}\frac{p(Z|X;\hat{\theta}^*)}{p(Z|X;\hat{\theta}^t)}$$

$$\log p(X;\theta) \geq \sum_Z p(Z|X;\hat{\theta}^t)\log\frac{p(X,Z;\hat{\theta}^t)}{p(Z|X;\hat{\theta}^*)} + \sum_Z p(Z|X;\hat{\theta}^t)\log\frac{p(Z|X;\hat{\theta}^*)}{p(Z|X;\hat{\theta}^t)}$$

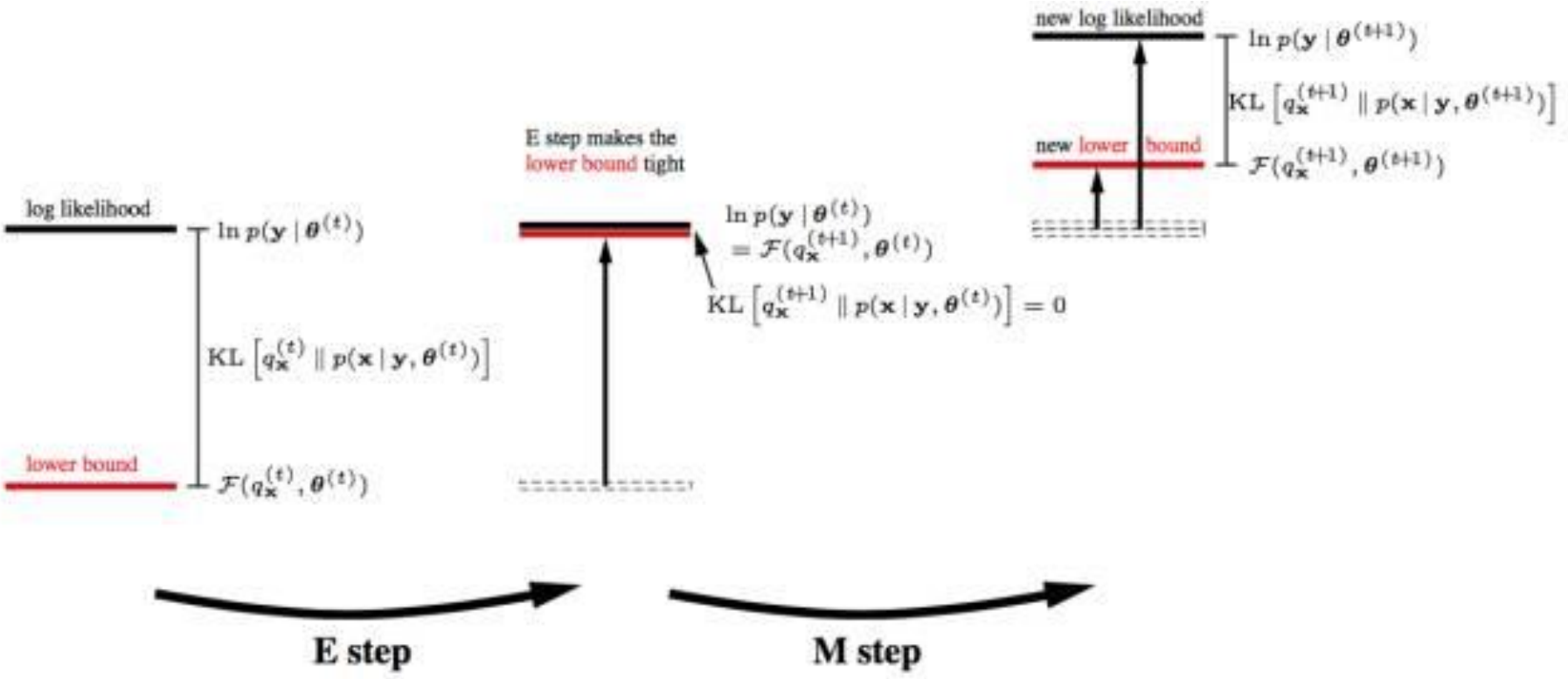
$$\rightarrow \quad \boldsymbol{\log p(X|\theta) \geq \mathcal{L}(q,\theta) + KL(q \parallel p)}$$

EM算法：迭代逼近最优解过程示意

$\log p(X|\theta) \geq \mathcal{L}(q,\theta) + KL(q \parallel p)$

Expectation：参数$\theta = \hat{\theta}^t$固定，使$KL(q \parallel p)$最小化，即更新后验概率$p(Z|X;\hat{\theta}^t)$

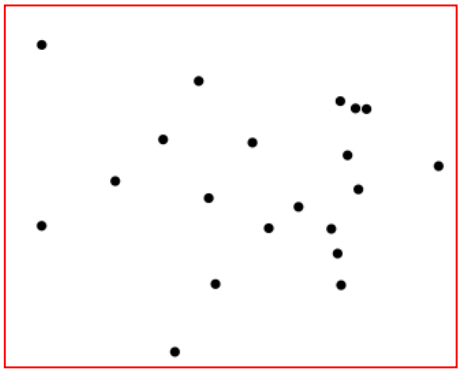
Maximization： 使$\mathcal{L}(q,\theta)$最大化，即更新参数$\hat{\theta}^{t+1}=\hat{\theta}^t$



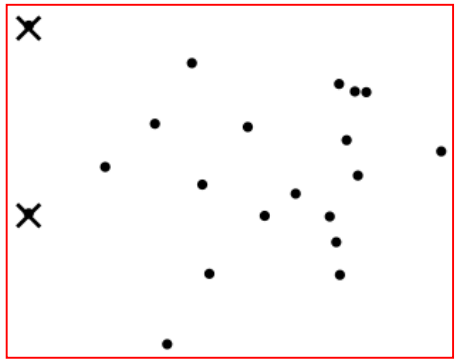
关于EM算法的九层境界的浅薄介绍 http://www.elecfans.com/d/604076.html

K-means算法是一种Hard EM算法

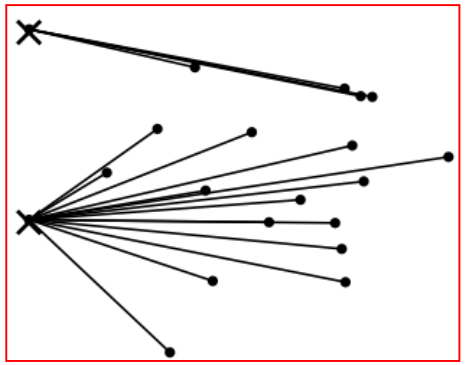
$$RSS = \sum_{k=1}^{K} RSS_k \qquad RSS_k = \sum_{x \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$



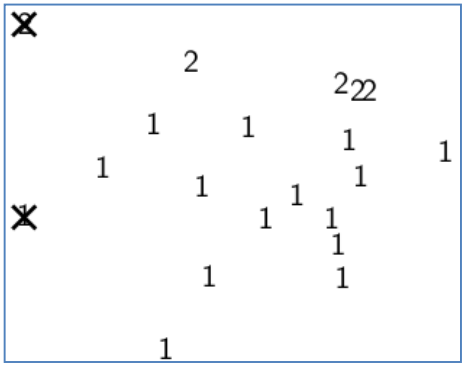
拟聚类文档



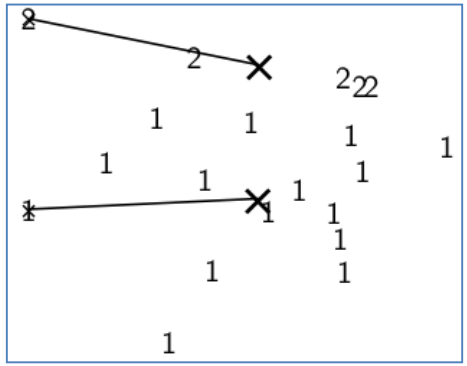
随机选择两个种子(K=2)



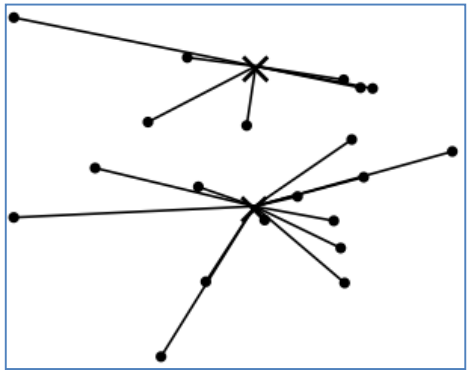
分配(第1次)



分配结果



重新计算质心向量



再重新分配(第2次)

K-means算法是一种Hard EM算法

$x = (x_1, x_2, \dots, x_N)$, x_i为第i个文档

$z = (z_1, z_2, \dots, z_N), z_i \in \{\omega_1, \omega_2, \dots, \omega_K\}$ z_i为隐变量，代表文档x_i所属的类

参数$\theta = (\theta_1, \theta_2, \dots, \theta_K) \in \{\mu(\omega_1), \mu(\omega_2), \dots, \mu(\omega_K)\}$，代表类的质心

$$\mathcal{L}(X;\theta) = \sum_Z q(Z;\hat{\theta}^t) \log \frac{p(X,Z;\hat{\theta}^t)}{q(Z;\hat{\theta}^t)}$$

Expectation：$q(Z;\hat{\theta}^t) = p(Z|X;\hat{\theta}^t) \xrightarrow{\operatorname{argmin}_j |x_i - \hat{\theta}_j{}^t|} P(Z = z_j | X = x_i; \hat{\theta}^t)$

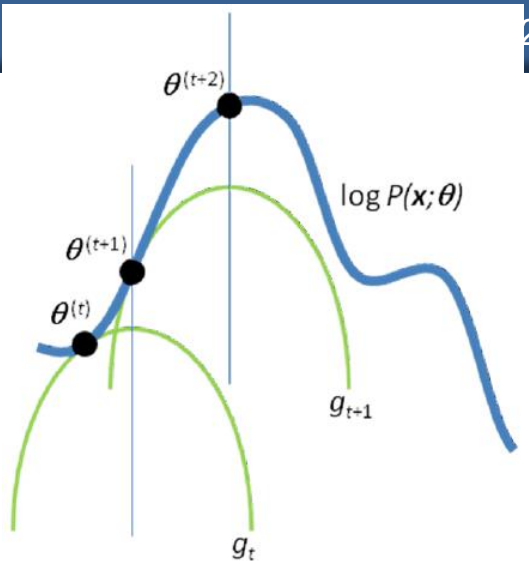
$p(X,Z;\hat{\theta}^t) \xrightarrow{\operatorname{argmin}_j |x_i - \hat{\theta}_j{}^t|} P(X = x_i, Z = z_j; \hat{\theta}^t)$

Maximization：$\hat{\theta}_j{}^{t+1} = \frac{1}{|\omega_j|} \sum_{z_i \in \omega_j} x_i$

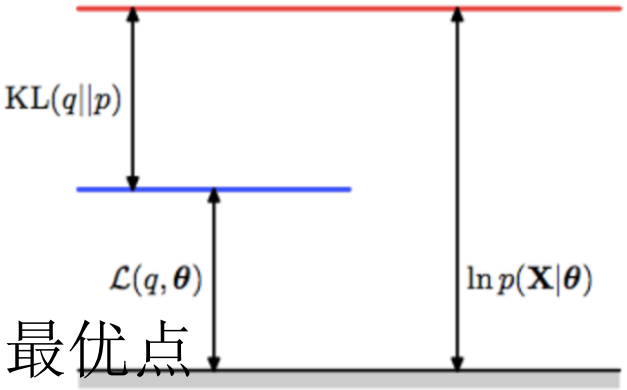
上述形式化描述受如下文献的启发导出，与该文并不相同，仅供参考

关于EM算法的九层境界的浅薄介绍 http://www.elecfans.com/d/604076.html

小结：EM (Expectation Maximization)



- 参数估计的两种情形
  - 完全信息下的MLE估计
  - 不完全信息下的参数估计
- EM算法是一种解决存在隐含变量优化问题的方法
  - E-Step：根据已经估计的参数计算隐藏变量的后验概率（即根据参数计算似然函数的期望）
  - M-Step：根据已经计算的后验概率更新参数（选择参数使似然最大化）
- 特点
  - 通过不断构造下界逐步向最优逼近
  - K-means算法是一种Hard EM算法
  - 估计参数的初值影响到是否落入局部最优点



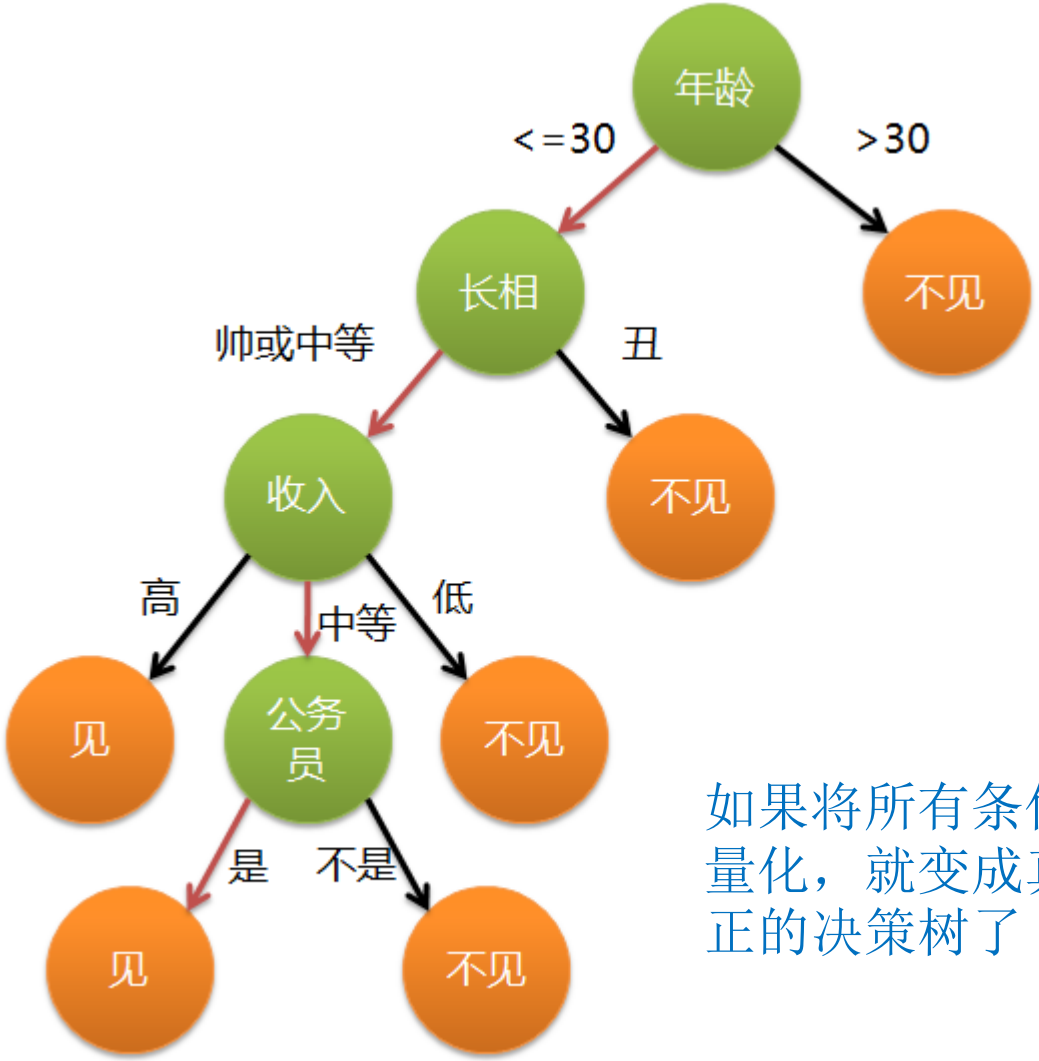
C. B. Do and S. Batzoglou, "What is the expectation maximization algorithm?," *Nature Biotechnology,* vol. 26, p. 897, 08/01/online 2008.
关于EM算法的九层境界的浅薄介绍 http://www.elecfans.com/d/604076.html

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、EM、K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

决策树（Decision Tree）

女儿：多大年纪了？
母亲：26。
女儿：长的帅不帅？
母亲：挺帅的。
女儿：收入高不？
母亲：不算很高，中等情况。
女儿：是公务员不？
母亲：是，在税务局上班呢。
女儿：那好，我去见见。



如果将所有条件量化，就变成真正的决策树了

存在问题：如何构造决策树（根节点、各级节点如何选）？

《Machine Learning》, Tom M.Mitchell, 1997, 第3章例子

示例：训练集、测试集

训练集

outlook	temperature	humidity	windy	play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

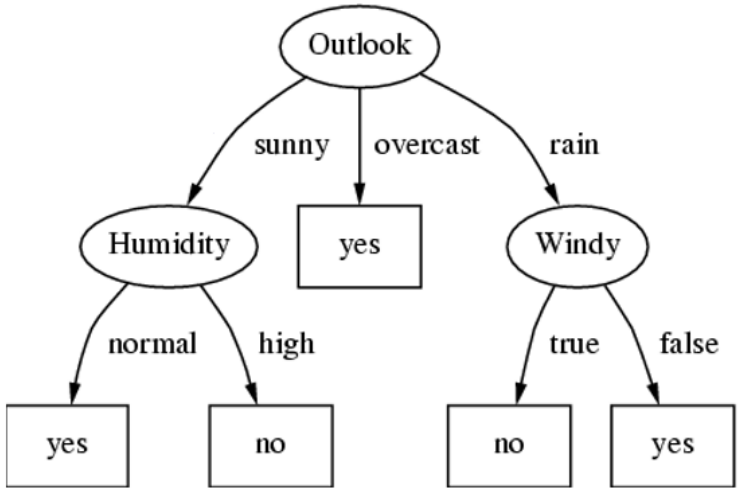
统计了14天的气象数据(指标包括outlook，temperature，humidity，windy)，并已知这些天气是否打球(play)。如果给出新一天的气象指标数据:sunny,cool,high,TRUE，判断一下会不会去打球。

这是个二分类问题

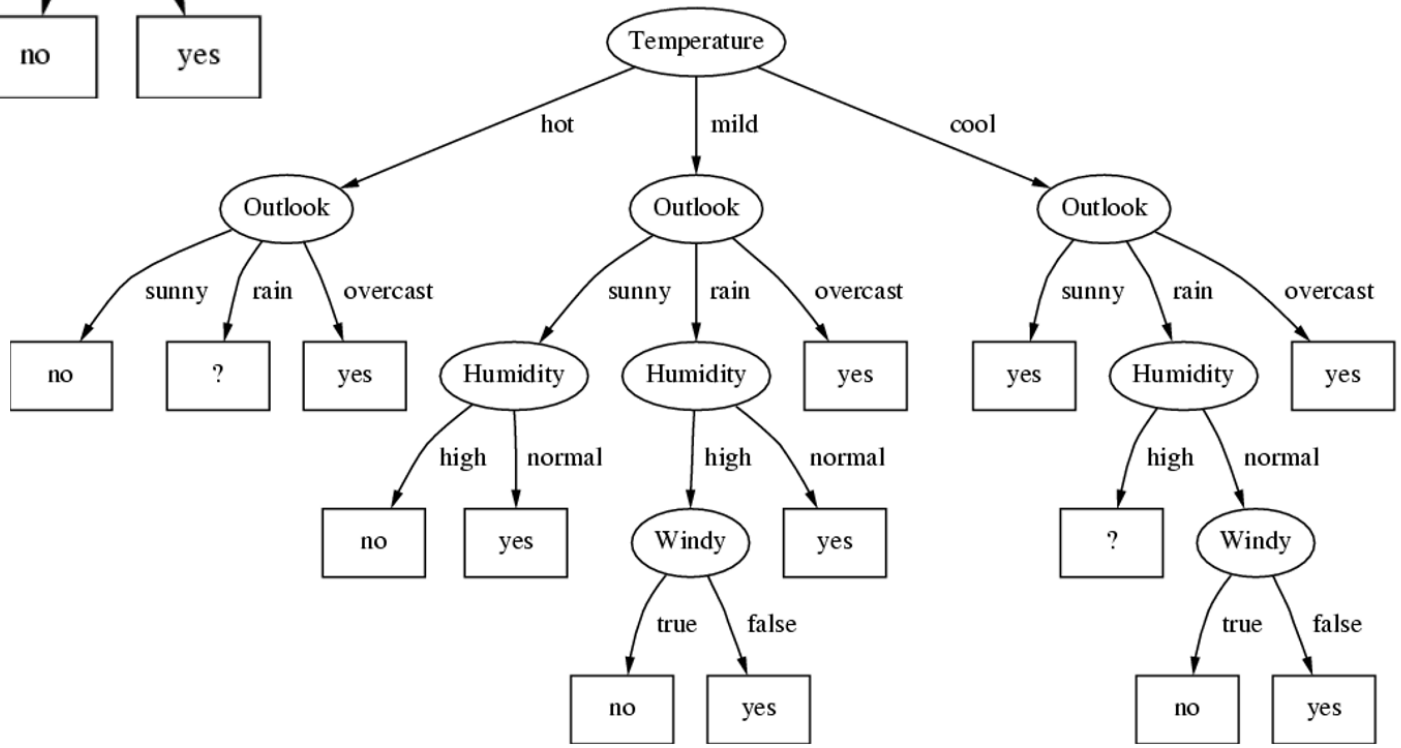
测试集

outlook	sunny
temperature	cool
humidity	high
windy	FALSE

两种不同的决策树



用什么样的规则构造决策树？



构建决策树的思路

- **构建决策树时通常采用自上而下的方法，在每一步选择一个最好的属性来分裂。**
  - “最好”的定义是使得子节点中的训练集尽量的纯。
  - “最好”也可以定义为“不纯度”尽量小。
- **在数据挖掘中，决策树主要有两种类型：**
  - 分类树：输出是样本的类别标签(label)
  - 回归树：输出是一个实数

回归，为什么称之为回归？

Galton: $y = 33.73 + 0.516\,x$

一种可能的构造方法

E1:outlook			E2: temperature			E3:humidity			E4:windy			C:play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								

事件C的不确定度→

$$H(C) = - p(c=yes)\log_2 p(c=yes) - p(c=no)\log_2 p(c=no)$$
$$= - 9/14\log(9/14) - 5/14\log2(5/14)$$
$$= 0.940$$

$$H(C|E1) = - p(e1=sunny)\,[p(c=yes|e1=sunny) + p(c=no|e1=sunny)]$$
$$- p(e1=overcast)\,[p(c=yes|e1=overcast) + p(c=no|e1=overcast)]$$
$$- p(e1=rainy)\,[p(c=yes|e1=rainy) + p(c=no|e1=rainy)]$$

$$H(C|E1) = -5/14 * [2/5\log_2(2/5) + 3/5\log_2(3/5)] = 5/14 * 0.971$$
$$-4/14 * 0$$
$$-5/14 * [3/5\log_2(3/5) + 2/5\log_2(2/5)]$$
$$= 0.693$$ ←获知outlook后的不确定度

$H(C|E2) = 0.911$ ← 获知temperature后的不确定度

$H(C|E3) = 0.778$ ← 获知humidity后的不确定度

$H(C|E4) = 0.892$ ← 获知windy后的不确定度

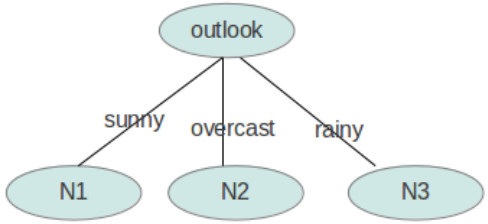
获知outlook后不确定度最小，选为根节点

ID3 (Iterative Dichotomiser 3)算法(1979)：树的构造之根节点选取 信息增益(information gain)最大

上述构造树的基本想法是随着树深度的增加，节点的熵迅速地降低。熵降低的速度越快越好，这样我们有望得到一棵高度最矮的决策树。

ID3算法基于**信息增益最大**的原则来选择节点（其实就是互信息量）
例如，计算根节点信息增益（属性E1与类别C的互信息量）
Gain(E1) = H(C) – H(C|E1) = 0.940 – 0.693 →最大，根节点
Gain(E2) = H(C) – H(C|E2) = 0.940 – 0.911
Gain(E3) = H(C) – H(C|E3) = 0.940 – 0.778
Gain(E4) = H(C) – H(C|E4) = 0.940 – 0.892

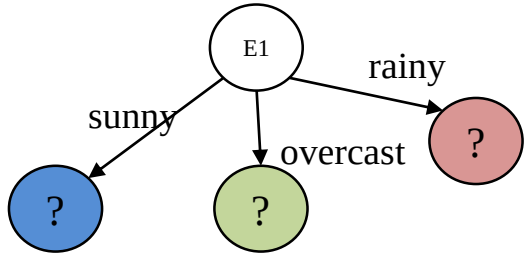
E1属性被选作根节点后，下一步，确定e1=sunny, e1=overcast, e1=rainy三个分支的下一级。



互信息量 I(X,Y) = H(X) – H(X|Y)，获知Y后对减少X不确定度的贡献

ID3算法(1979)：树的构造之子节点选取

互信息量 I(X,Y) = H(X) – H(X|Y)，获知Y后对减少X不确定度的贡献
ID3算法中的信息增益其实就是互信息量
例如，属性E1与类别C的互信息量
Gain(E1) = H(C) – H(C|E1) = 0.940 – 0.693 →最大，根节点
Gain(E2) = H(C) – H(C|E2) = 0.940 – 0.911
Gain(E3) = H(C) – H(C|E3) = 0.940 – 0.778
Gain(E4) = H(C) – H(C|E4) = 0.940 – 0.892



E1属性被选作根节点后，下一步，确定e1=sunny, e1=overcast, e1=rainy三个分支的下一级。通过计算这三种条件下的信息增益来选择。
Gain(E2, e1=sunny) 、 Gain(E3, e1=sunny) 、Gain(E4, e1=sunny)
Gain(E2, e1=overcast)、Gain(E3, e1=overcast)、Gain(E4, e1=overcast)
Gain(E2, e1=rainy)、Gain(E4, e1=rainy)、Gain(E4, e1=rainy)

E1:outlook	E2: temperature			E3:humidity			E4:windy			C:play	
sunny		yes	no		yes	no		yes	no	yes	no
	hot	0	2	high	0	3	false	0	2	1	3
	mild	1	1	normal	1	0	true	1	1		
	cool	0	0								

Gain(E2, e1=sunny) = H(C) - H(C|E2, e1=sunny) = H(C) - 0.5

Gain(E3, e1=sunny) = H(C) - H(C|E3, e1=sunny) = H(C) – 0 →最大!

Gain(E4, e1=sunny) = H(C) - H(C|E4, e1=sunny) = H(C) - 0.5

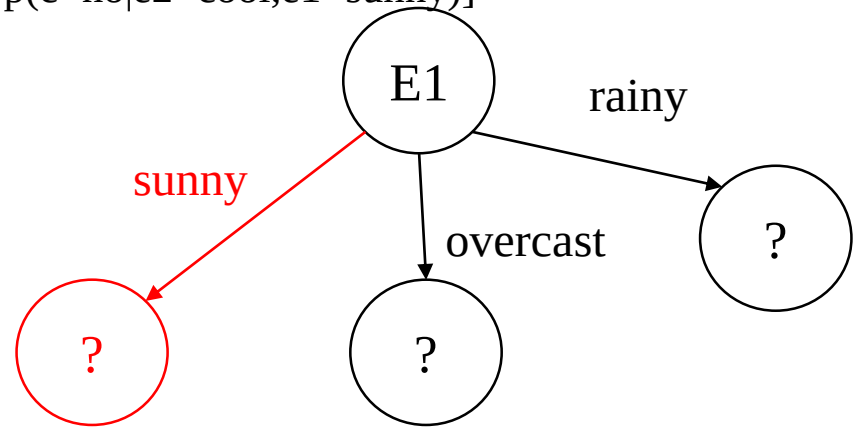
H(C|E2, e1=sunny) =

- p(e2=hot,e1=sunny) [p(c=yes|e2=hot,e1=sunny) + p(c=no|e2=hot,e1=sunny)]
- p(e2=mild,e1=sunny) [p(c=yes|e2=mild,e1=sunny) + p(c=no|e2=mild,e1=sunny)]
- p(e2=cool,e1=sunny) [p(c=yes|e2=cool,e1=sunny) + p(c=no|e2=cool,e1=sunny)]

= 2/4 * 0 – 2/4 * 1 – 0 = 0.5

Max{ Gain(E2, e1=sunny) ,
Gain(E3, e1=sunny) ,
Gain(E4, e1=sunny) }

= Gain(E3, e1=sunny)



E1:outlook	E2: temperature			E3:humidity			E4:windy			C:play	
sunny		yes	no		yes	no		yes	no	yes	no
	hot	0	2	high	0	3	false	0	2	1	3
	mild	1	1	normal	1	0	true	1	1		
	cool										
overcast		yes	no		yes	no		yes	no	yes	no
	hot	2	0	high	2	0	false	2	0	4	0
	mild	1	0	normal	2	0	true	2	0		
	cool	1	0								
rainy		yes	no		yes	no		yes	no	yes	no
	hot			high	1	1	false	3	0	3	2
	mild	2	1	normal	2	1	true	0	2		
	cool	1	1								

P(c=yes|e1=overcast) = 1

P(c=yes|e4=windy, e1=rainy) = 1

ID3算法(1979)

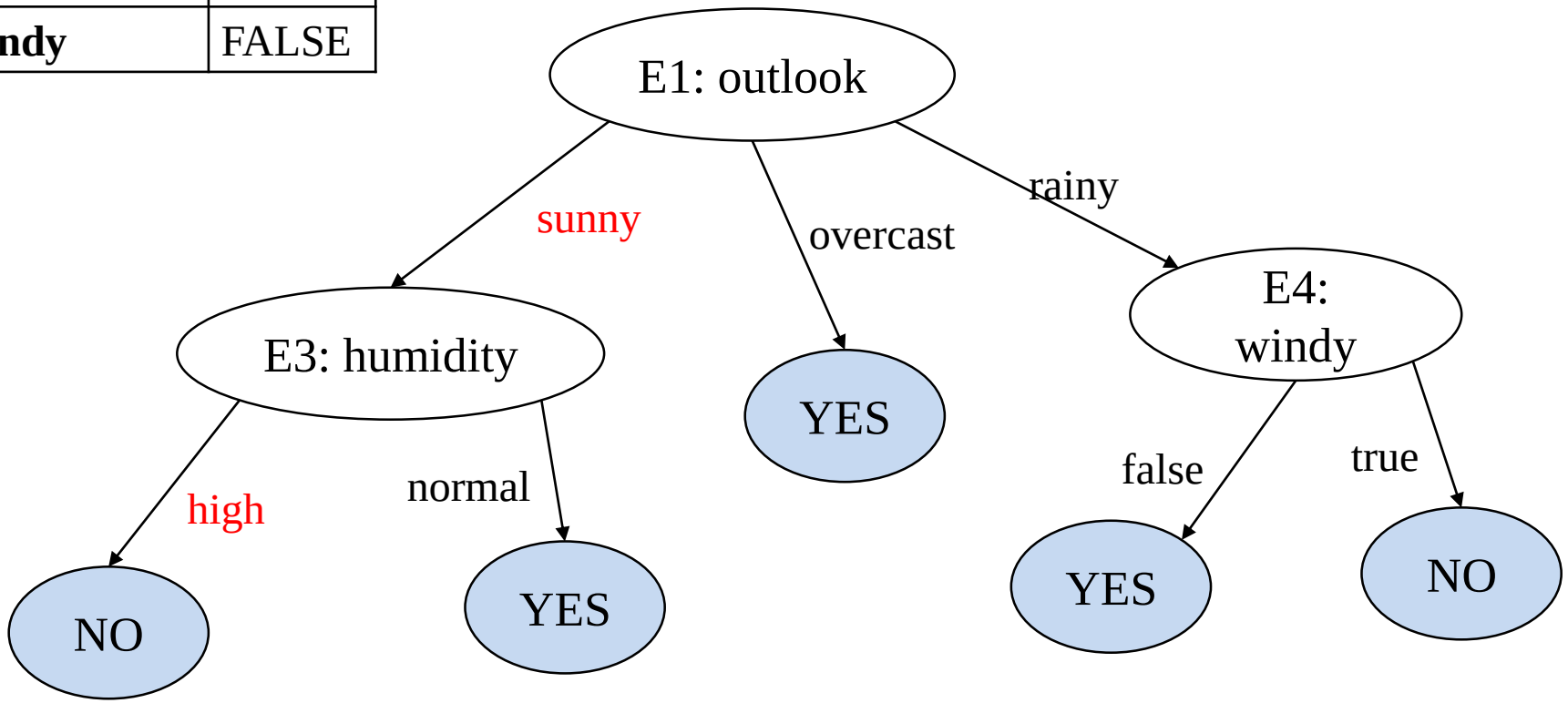
测试集

outlook	sunny
temperature	cool
humidity	high
windy	FALSE

思考：

（1）属性temperature并未出现在所生成的决策树中

（2）对于给定的测试集而言属性windy也未被利用



ID3算法(1979)缺点

- **ID3信息增益的计算依赖于特征数目较多的特征，而属性取值最多的属性并不一定最优。**
- **ID3是单变量决策树(在分枝节点上只考虑单个属性)，许多复杂概念的表达困难，属性相互关系强调不够，容易导致决策树中子树的重复或有些属性在决策树的某一路径上被检验多次。**
- **抗噪性能差，训练例子中正例和反例的比例较难控制。**

其他待解决的问题

1、如果某属性的取值为连续值？ 分段划分

temperature: -10℃ ～ 45℃ → temperature: hot, mild, cool

temperature	-5	-8	3	10	20	35
play	NO	NO	YES	YES	YES	YES

2、如果某属性的取值特别多（熵值高）？ 归一化

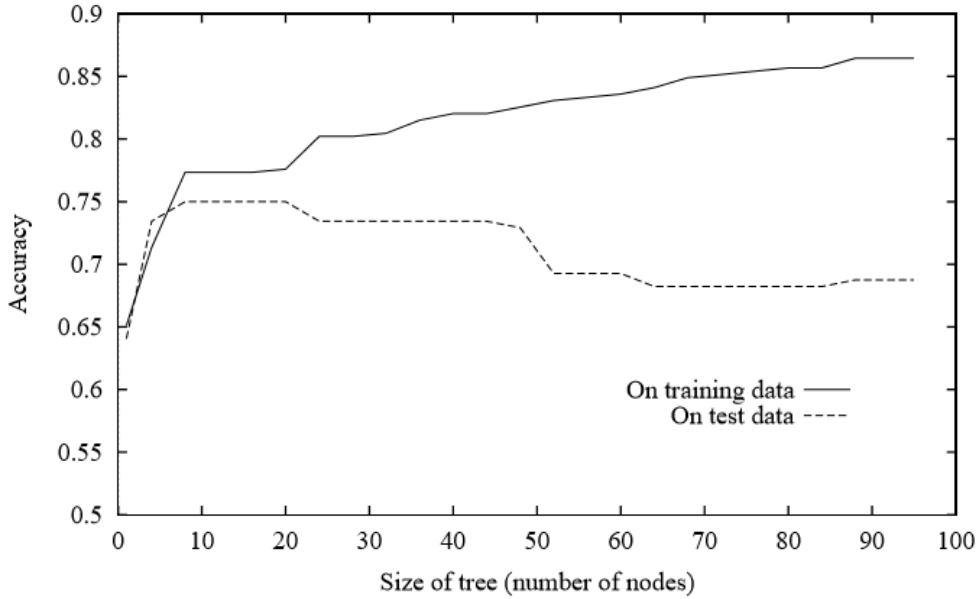
$-1/2\log_2(1/2)$ $-1/2\log_2(1/2) = 1$

$-1/4\log_2(1/4)$ $-1/4\log_2(1/4)$ $-1/4\log_2(1/4)$ $-1/4\log_2(1/4)=2$

$-1/8\log_2(1/8)$-… $-1/8\log_2(1/8)=3$

需要找到合适的归一化方式，使得取值多和取值少的属性贡献度一致……

其他待解决的问题
过拟合与剪枝



3、决策树的过拟合(Overfitting)

对于训练样本而言，树表现完好，误差率极低且能够正确得对训练样本集中的样本进行分类。训练样本中的错误数据也会被决策树学习，成为决策树的部分，但是对于测试数据的表现就没有想象的那么好，或者极差，这就是所谓的过拟合问题。

树的剪枝(Pruning)

研究表明，过拟合的决策树的错误率比经过简化的决策树的错误率要高。

剪枝可以分为两种：预剪枝(Pre-Pruning)和后剪枝(Post-Pruning)：

预剪枝，及早的停止树增长。

后剪枝，在已生成决策树上进行剪枝，得到简化的决策树。

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、 K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

C4.5(1992)：对ID3的改进

ID3算法中的**信息增益**（其实就是互信息量）

Gain(E1) = H(C) – H(C|E1)

Gain(E3, e1=sunny) = H(C) - H(C|E3, e1=sunny)

……

C4.5算法的**信息增益率**

IGR(E1) = Gain(E1) / H(E1)

IGR(E1) = [H(C) – H[C|E1]] / **H(E1)**

C4.5(1992)：对ID3的扩展

- **C4.5算法在以下几方面对ID3算法进行了改进：**
  - 1) 用信息增益率来选择属性，克服了用信息增益选择属性时偏向选择取值多的属性的不足；
  - 2) 在树构造过程中进行剪枝；
    - 一些分枝反映的是训练数据中的异常（数据中的噪声和离群点）。剪枝方法是用来处理这种过分拟合数据的问题。
  - 3) 能够完成对连续属性的离散化处理；
  - 4) 能够对不完整数据进行处理。
- **C4.5算法优点**
  - 产生的分类规则易于理解，准确率较高。
- **C4.5算法缺点**
  - 在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。此外，C4.5只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

罗斯.昆兰（Ross Quinlan）

http://www.rulequest.com/Personal/
Software available for download:
FOIL Release 6: (shell archive)
FFOIL Release 2: (shell archive)
C4.5 Release 8: (gzipped tar file)
C4.5 has been superseded by C5.0.
Source code for C5.0 is available

2011年获得了数据挖掘领域最高荣誉奖KDD创新奖，昆兰发明了著名的决策树学习算法ID3、C4.5，其个人主页公布了C4.5的C代码。

John Ross Quinlan is a computer science researcher in data mining and decision theory. He has contributed extensively to the development of decision tree algorithms, including inventing the canonical C4.5 and ID3 algorithms. He also contributed to early ILP literature with First Order Inductive Learner (FOIL). He is currently running the company RuleQuest Research which he founded in 1997.

Inductive Logic Programming, ILP

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、 K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

CART (Classification And Regression Tree, 1984)

CART算法是一种二分递归分割技术，把当前样本划分为两个子样本，使得生成的每个非叶子结点都有两个分支，因此CART生成的决策树是结构简洁的二叉树。

设$x_1,x_2,\ldots,x_n$代表单个样本的n个属性，y表示所属类别。CART算法通过递归的方式将n维的空间划分为不重叠的矩形。划分步骤大致如下：

（1）选一个自变量x_i，再选取x_i的一个值v_i，v_i把n维空间划分为两部分，一部分的所有点都满足$x_i \le v_i$，另一部分的所有点都满足$x_i > v_i$，对非连续变量来说属性值的取值只有两个，即等于该值或不等于该值。

（2）递归处理，将上面得到的两部分按步骤（1）重新选取一个属性继续划分，直到把整个维空间都划分完。

在划分时有一个问题，它是按什么标准来划分的？ 对于一个变量属性来说，它的划分点是一对连续变量属性值的中点。假设m个样本的集合一个属性有m个连续的值，那么则会有(m-1)个分裂点，每个分裂点为相邻两个连续值的均值。每个属性的划分按照能减少的杂质的量来进行排序，而杂质的减少量定义为划分前的杂质减去划分后的每个节点的杂质量划分所占比率之和。

CART (Classification And Regression Tree, 1984)
基尼指数用于指示分裂方式

GINI指数的概念：给定一个数据集D,其中包含的分类类别总数为K，类别k在数据集中的个数为$|C_k|$，其GINI指数表示为：

$$GINI(D) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2 = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2$$

若样本集合D根据特征A是否取某一可能值a被分割成D_1和D_2两部分，即：

$$D_1 = \{(x,y) \in D | A(x) = a\}, D_2 = D - D_1$$

在特征A给定的条件下，集合D的基尼指数D(D,A)定义为：

$$GINI(D,A) = \frac{D_1}{D} GINI(D_1,A) + \frac{D_2}{D} GINI(D_2,A)$$

数据集内包含的类别越杂，GINI指数就越大。

CART (Classification And Regression Tree, 1984)
CART决策树生成算法

输入：训练数据集D，停止计算条件
根据训练数据集，从根节点开始，递归地对每个节点进行以下操作，构建二叉决策树：
①计算现有特征对数据集D的基尼指数。此时，对每一个特征A，对其可能取的每一个值a，根据样本点对A=a的测试为“是”或“否”将D分割为D1和D2两部分，计算A=a时的基尼指数。
②在所有可能的特征A以及它们可能的切分点a中，选择基尼指数最小的特征及其对应的切分点作为最优特征与最优切分点，依最优特征与最优切分点，从现节点生成两个子节点，将训练数据依特征分配到两个子节点中去。
③对两个子节点递归地调用①②，直至满足停止条件。

算法的停止条件：节点中的样本个数小于预定阈值，或样本集的基尼指数小于预定阈值（样本基本属于同一类），或者没有更多特征。

CART (Classification And Regression Tree, 1984) 示例

有房者	婚姻状况	年收入	拖欠贷款者
是	单身	125K	否
否	已婚	100K	否
否	单身	70K	否
是	已婚	120K	否
否	离异	95K	是
否	已婚	60K	否
是	离异	220K	否
否	单身	85K	是
否	已婚	75K	否
否	单身	90K	是

图中，属性有3个，分别是有房情况，婚姻状况和年收入，其中有房情况和婚姻状况是离散的取值，而年收入是连续的取值。拖欠贷款者属于分类的结果。

	有房	无房
否	3	4
是	0	3

$Gini(t_1)=1-(3/3)^2-(0/3)^2=0$
$Gini(t_2)=1-(4/7)^2-(3/7)^2=0.4849$
$Gini=0.3\times 0+0.7\times 0.4898=0.343$

	单身或已婚	离异
否	6	1
是	2	1

$Gini(t_1)=1-(6/8)^2-(2/8)^2=0.375$
$Gini(t_2)=1-(1/2)^2-(1/2)^2=0.5$
$Gini=8/10\times 0.375+2/10\times 0.5=0.4$

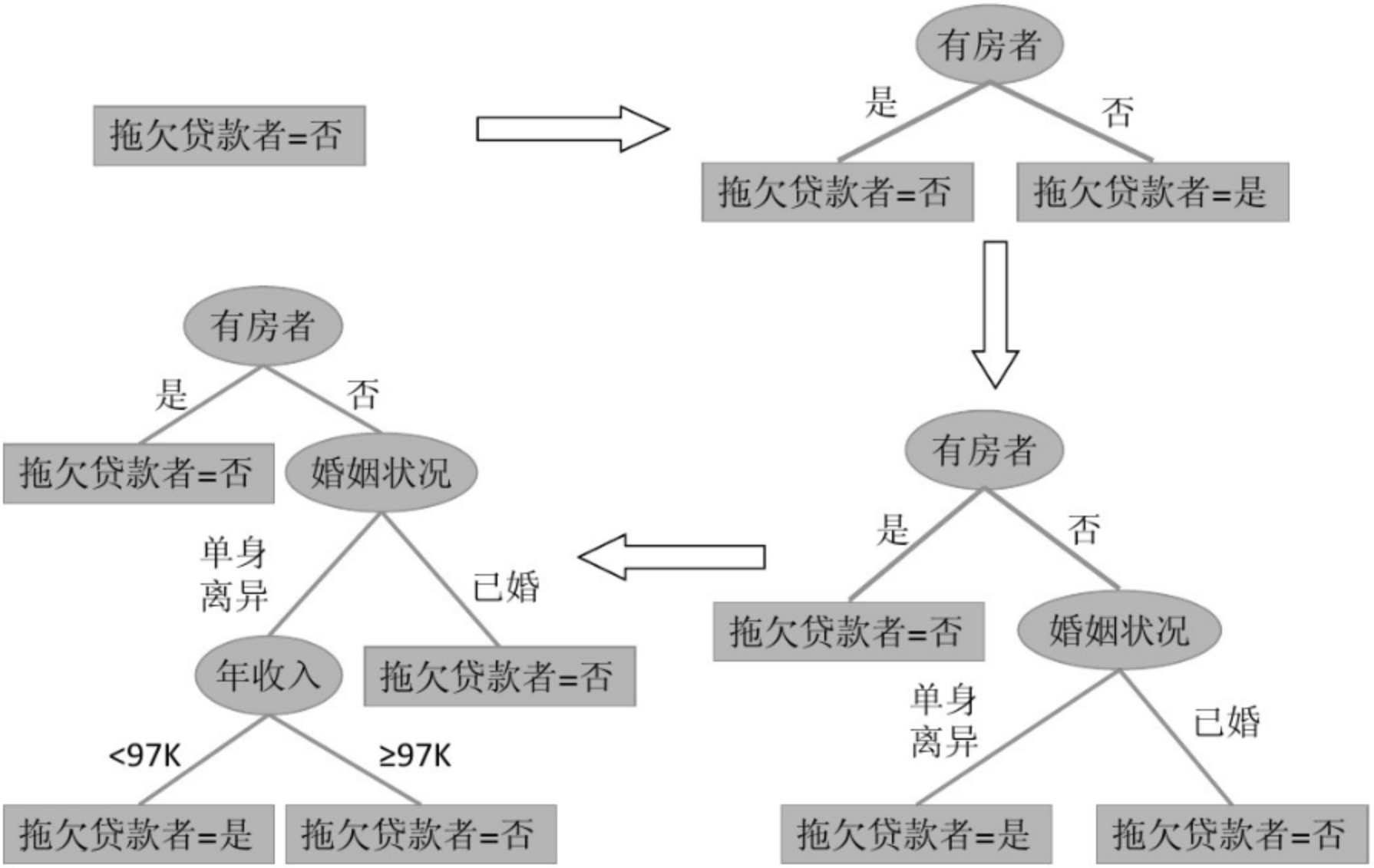
	单身或离异	已婚
否	3	4
是	3	0

$Gini(t_1)=1-(3/6)^2-(3/6)^2=0.5$
$Gini(t_2)=1-(4/4)^2-(0/4)^2=0$
$Gini=6/10\times 0.5+4/10\times 0=0.3$

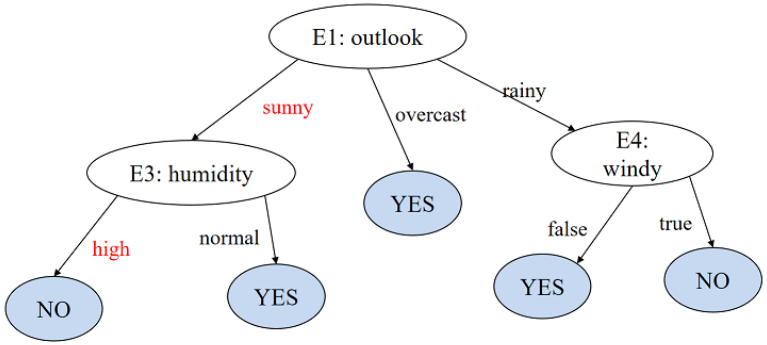
	离异或已婚	单身
否	5	2
是	1	2

$Gini(t_1)=1-(5/6)^2-(1/6)^2=0.2778$
$Gini(t_2)=1-(2/4)^2-(2/4)^2=0.5$
$Gini=6/10\times 0.2778+4/10\times 0.5=0.3667$

	60		70		75		85		90		95		100		120		125		220	
	65		72		80		87		92		97		110		122		172			
	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>	≤	>		
是	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0		
否	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1		
Gini	0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400			

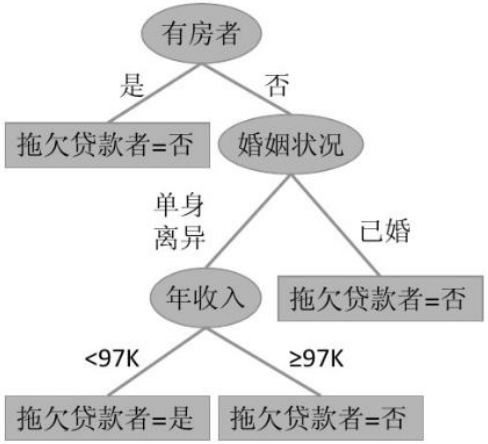

拖欠贷款者=否
有房者
是
否
拖欠贷款者=否
拖欠贷款者=是
有房者
是
否
拖欠贷款者=否
婚姻状况
单身
离异
已婚
年收入
拖欠贷款者=否
<97K
≥97K
拖欠贷款者=是
拖欠贷款者=否
有房者
是
否
拖欠贷款者=否
婚姻状况
单身
离异
已婚
拖欠贷款者=是
拖欠贷款者=否

小结：决策树算法



- **ID3，信息增益，1979**
  - Gain(E1) = H(C) – H(C|E1)
  - Gain(E3, e1=sunny) = H(C) - H(C|E3, e1=sunny)
- **C4.5，信息增益率，1992**
  - IGR(E1) = Gain(E1) / H(E1)
  - IGR(E1) = [H(C) – H[C|E1]] / H(E1)
- **CART，基尼指数，1985**

$$GINI(D) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2 = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2$$



- **一些问题**
  - 分枝有限个，如何处理连续值属性？
  - 误入歧途，如何处理过拟合问题？
  - 信息不完整，如何处理部分样本属性缺失的问题？
  - ……

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、 K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

分类器的组合 Combining Models

- It is often found that improved performance can be obtained by combining multiple models together in some way, instead of just using a single model in isolation.
  - 多个分类器组合效果优于单个分类器
- For instance, we might train L different models and then make predictions using the average of the predictions made by each model. Such combinations of models are sometimes called *committees*.
  - 多个分类器组合的权重通过投票机制（委员会）实现

组合多个分类器的一般性问题

- **弱分类器：组合前的每一个分类器**
- **强分类器：组合后最后的分类器**

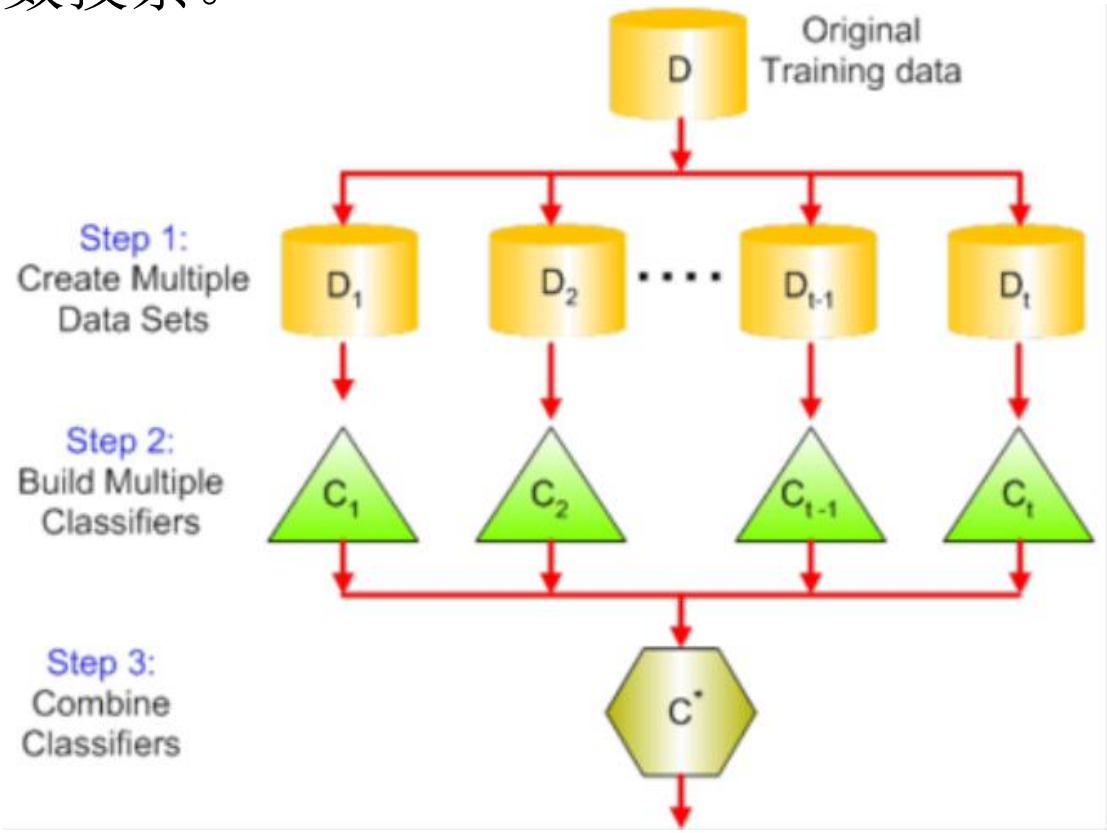
- **每个弱分类器的训练样本集？**
  - 相同还是不同，如果不同如何选取？
- **每个弱分类器的学习方法？**
  - 相同还是不同，如果不同选取的原则？
- **各个弱分类器是否相关？**
  - 训练样本是相关的？
  - 学习方法是相关的？
- **如何将训练得到的各弱分类器组合成强分类器？**

Bagging（Bootstrap aggregating）思路

- **bagging技术的过程：**
  - 假设有一个大小为n的训练集D，bagging会从D中有放回的均匀抽样，假设用bagging生成了m个新的训练集Di，每个Di的大小为j。由于有放回的进行抽样，那么在Di中的样本有可能是重复的。
  - 如果j=n，这种取样称为bootstrap取样。现在，可以用上面的m个训练集来拟合m个模型，然后结合这些模型进行预测。对于回归问题来说，我们平均这些模型的输出;对于分类问题来说，我们进行投票（voting）。
- **bagging能提升机器学习算法的稳定性和准确性，它可以减少模型的方差从而避免overfitting。**

Bagging（Bootstrap aggregating）思路示意图

首先创建随机训练数据集样本（训练数据集的子集）。然后为每个样本建立分类器。最后，这些多分类器的结果将结合起来，使用平均或多数投票。



Bagging（Bootstrap aggregating）有放回的随机抽样示例

假设有一个训练集D的大小为7，用bagging生成3个新的训练集Di,每个Di的大小为7，结果如下表：

样本索引	Bagging(D1)	Bagging(D2)	Bagging(D3)
1	2	7	3
2	2	3	4
3	1	2	3
4	3	1	3
5	5	1	6
6	2	5	1
7	6	4	1

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、 K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

Boosting，思路

- **Boosting 是一个迭代的过程，用来自适应的改变训练样本的分布，使分类器聚焦在那些很难分的样本上。**
- **在Bagging的过程中，抽取自主样本集的时候是简单随机抽样，每个样本被抽到的概率是一样的，但是Boosting算法中给每一个样本赋予一个权值，每一轮在抽取自助样本集、训练完基分类器之后会对原始样本集做一个分类，然后跟效果比对，然后对分错的训练样本，自动的调节该样本的权值。**
- **权值可以用在以下方面：**
  - （1）可以用作抽样分布，从原始数据集中选出自助样本集（注意，Bagging抽样时样本是均匀分布的）
  - （2）基分类器训练时更倾向于用高权值的样本进行训练。

Boosting，示例

例，有10个样本，开始时简单随机抽样，每个样本被抽到的概率是一样的，都是1/10，然后第一轮结束后用得到的模型对该轮数据集中的样本分类，发现样本4被分错了，就增加它的权值，这样，第二轮抽取的时候4被抽到的概率增加了，第三轮也是一样。这样做，就可以让基分类器聚焦于难被分类的样本4，增加4被正确分类的几率。

Boosting（第一轮）	7	3	2	8	7	9	**4**	10	6	3
Boosting（第二轮）	5	**4**	9	**4**	2	5	1	7	**4**	2
Boosting（第三轮）	**4**	**4**	8	10	**4**	5	**4**	6	3	**4**

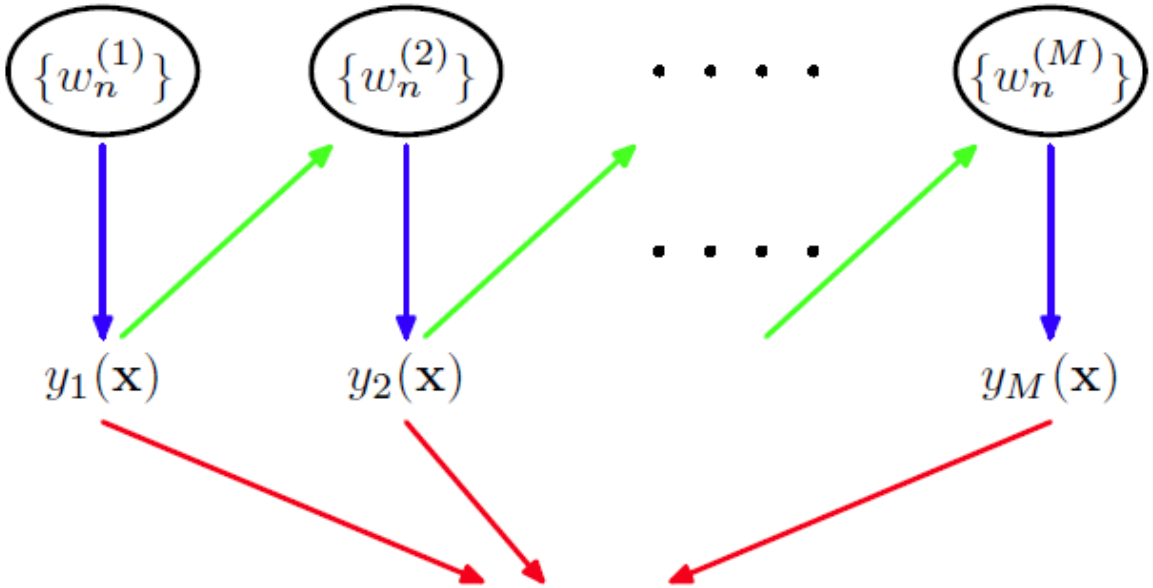
今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、 K-means、SVM、kNN
- 决策树
  - ID3
  - C4.5
  - CART
- 把若干个分类器整合为一个分类器
  - Bagging
  - Boosting
  - AdaBoost，1995
- 流数据挖掘：频繁项集
- Web中的数据挖掘

AdaBoost (Adaptive boosting), 1995

Adaboost结构：最后的分类器Y_M是由数个弱分类器（weak classifier）组合而成的，相当于最后m个弱分类器来投票决定分类，而且每个弱分类器的“话语权”α不一样。

Schematic illustration of the boosting framework. Each base classifier $y_m(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w_n^{(m)}$ depend on the performance of the previous base classifier $y_{m-1}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_M(\mathbf{x})$ (red arrows).



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

《Pattern Recognition and Machine Learning》,Christopher M. Bishop
Chapter 14. COMBINING MODELS

AdaBoost

训练数据：$\mathbf{x_1}, \ldots, \mathbf{x_N}$

对应类别：$t_1, \ldots, t_N$　$t_n \in \{-1, 1\}$

分类器：$y_1, \ldots, y_M$

第m个分类器的误差J_m

第n个样本在第m个分类器y_m中的权重：$w_n^{(m)}$

示性函数 I()

第m个分类器的归一化误差ε_m

第m个分类器的权重α_m

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.

初始化样本权重系数$w_n^{(1)}$

2. For $m = 1, \ldots, M$:

(a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

(b) Evaluate the quantities

分类器y_m归一化误差ε_m

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

and then use these to evaluate

分类器y_m权重α_m

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}. \tag{14.17}$$

(c) Update the data weighting coefficients

样本权重系数$w_n^{(1)}$更新

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}) \right). \tag{14.19}$$

AdaBoost弱分类器示例

《Pattern Recognition and Machine Learning》,Christopher M. Bishop
Chapter 14. COMBINING MODELS

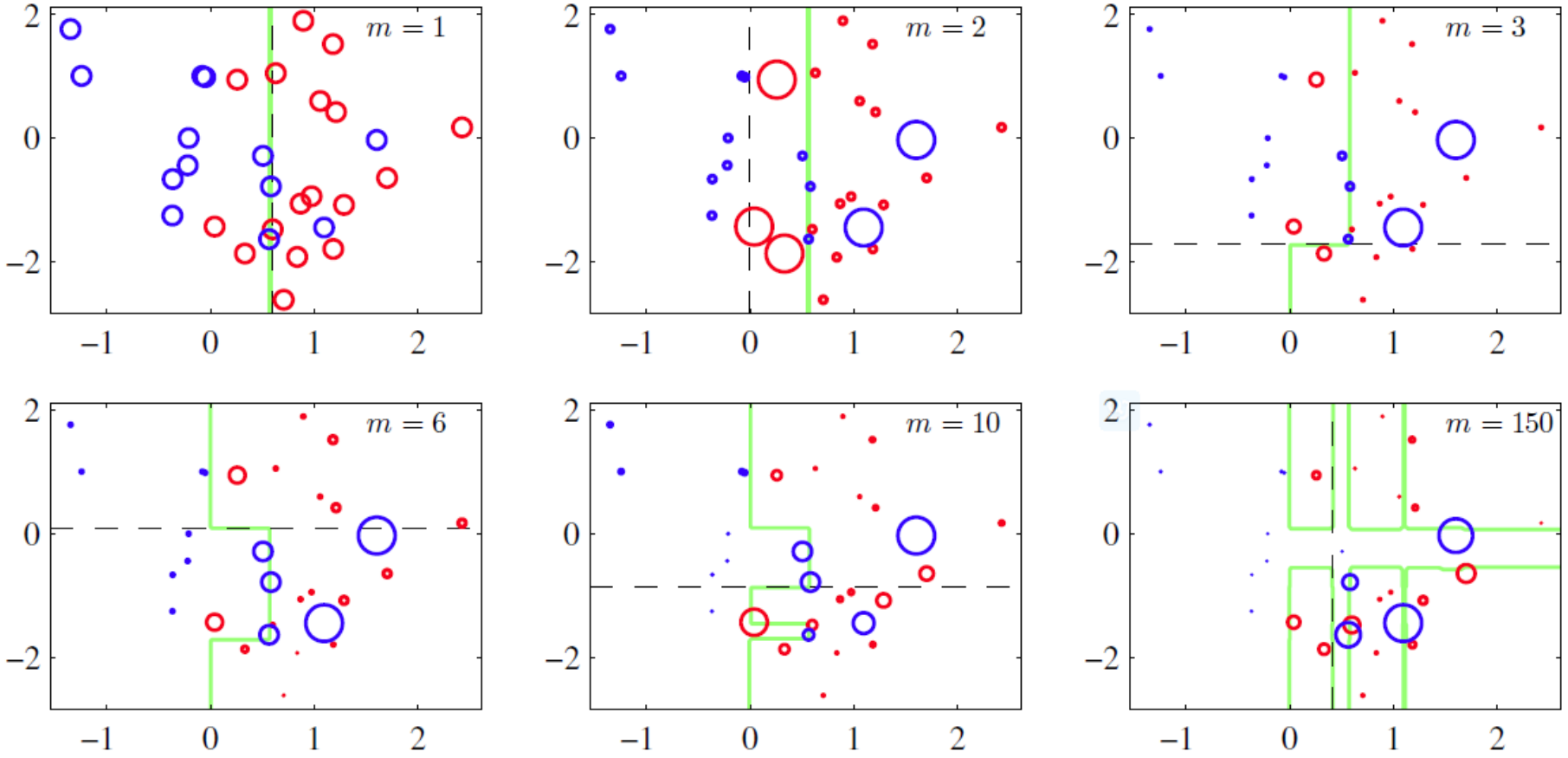


Figure 14.2 Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training the $m = 2$ base learner.

随机森林（Random Forest，1995）

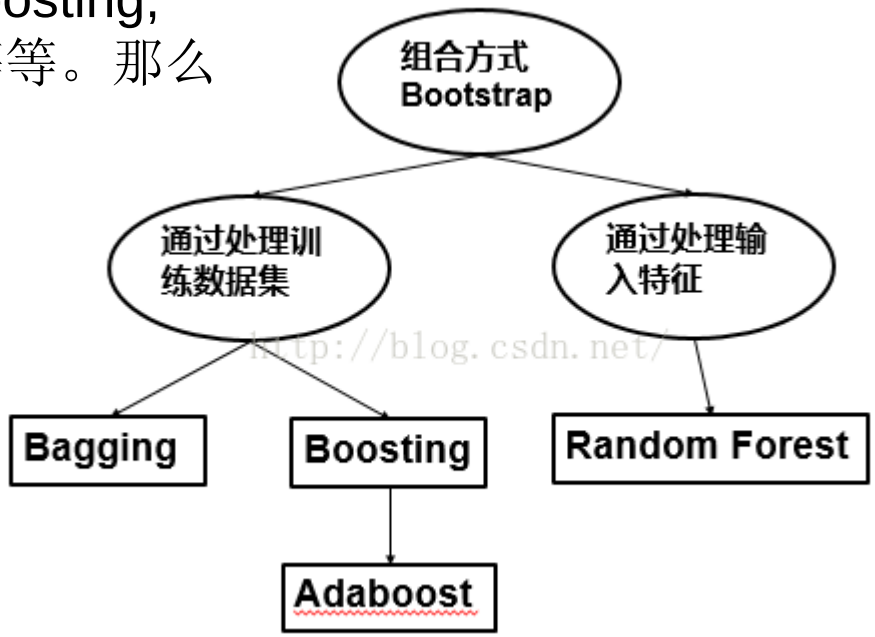
- **随机森林利用随机的方式将许多决策树组合成一个森林，每个决策树在分类的时候投票决定测试样本的最终类别。随机森林主要包括下4个部分：**
- **1.随机选择样本**
  - 给定一个训练样本集，数量为N，我们使用**有放回采样**到N个样本，构成一个新的训练集。
- **2. 随机选择特征**
  - 在随机森林中，不计算所有特征的增益，而是从总量为M的特征向量中，**随机选择m个特征**，m可以等于sqrt(M)，然后计算m个特征的增益，选择最优特征。
- **3. 构建决策树**
  - 在随机产生的样本集上使用一般决策树的构建方法，得到一棵决策树。
- **4. 随机森林投票分类**
  - 通过上面的三步走，得到一棵决策树，我们可以重复这样的过程H次，就得到了H棵决策树。然后来了一个测试样本，我们就可以用每一棵决策树都对它分类一遍，得到了H个分类结果。这时，我们可以使用简单的投票机制，或者该测试样本的最终分类结果。

小结：分类器的组合

多分类器进行组合的目的是为了将单个分类器（也叫基分类器 base classifier）进行组合，提升对未知样本的分类准确率，（依赖于基分类器的分类性能和基分类器之间的独立性）。

提到组合方法（classifier combination），有很多的名字涌现，如bootstraping, boosting, adaboost, bagging, random forest 等等。那么它们之间的关系如图：

注意：Boosting中组合在一起的多个分类器是有关联的：后一个分类器依赖于前一个分类器。



Do we Need Hundreds of classifiers to Solve Real World Classification Problems?

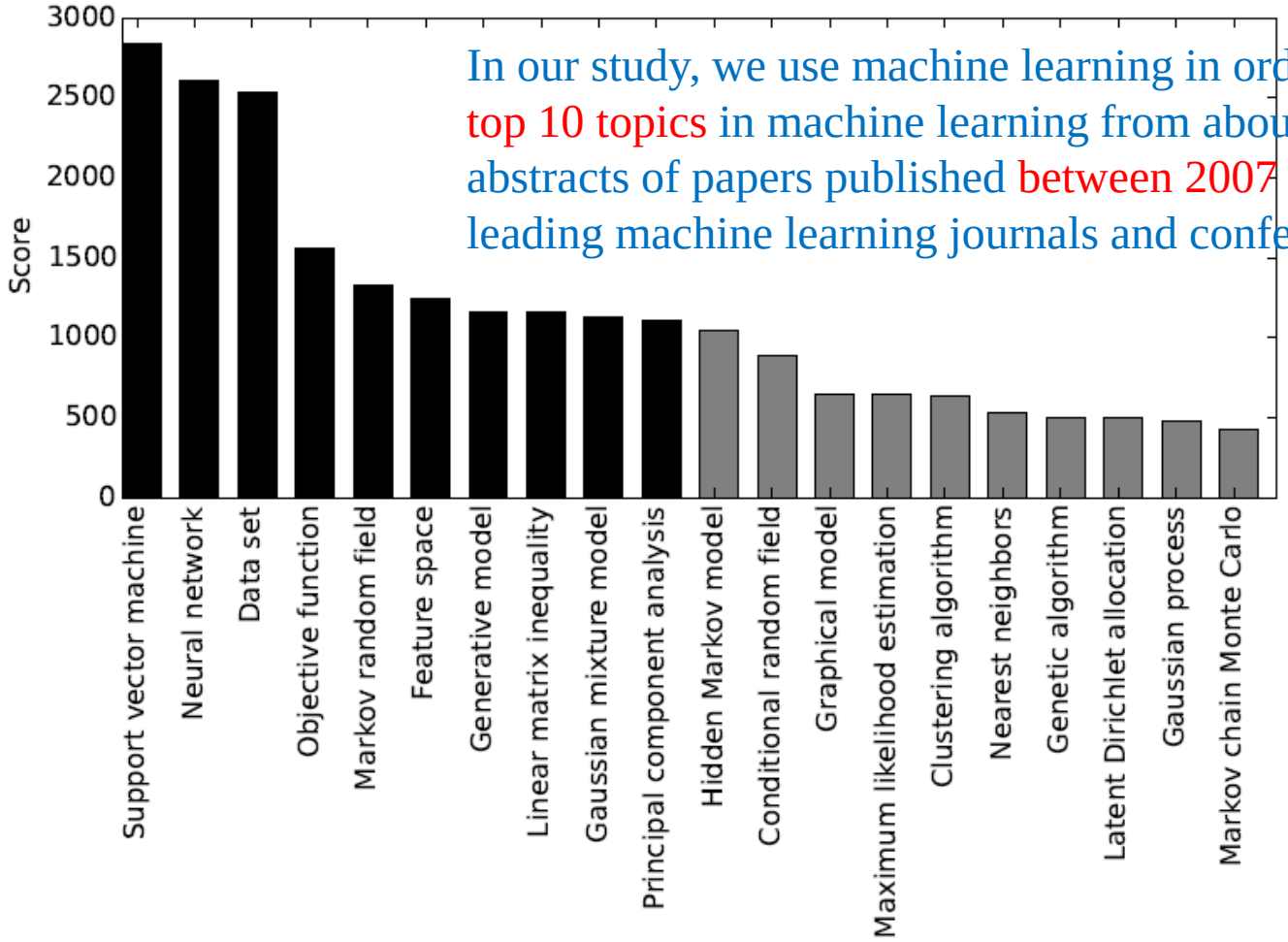
- We evaluate 179 classifiers arising from 17 families (discriminant analysis, Bayesian, neural networks, SVM, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in **Weka**, **R**, C and Matlab, including all the relevant classifiers available today.
- We use 121 data sets.
- The classifiers most likely to be the bests are the random forest (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy……..

Manuel Fern andez-Delgado et.al. **Do we Need Hundreds of classifiers to Solve Real World Classication Problems?** , Journal of Machine Learning Research 15 (2014) 3133-3181

The Top 10 Topics in Machine Learning Revisited: A Quantitative Meta-Study, 2017

https://arxiv.org/abs/1703.10121

The Top 10 Topics in Machine Learning Revisited: A Quantitative Meta-Study, 2017



In our study, we use machine learning in order to find the top 10 topics in machine learning from about 54K abstracts of papers published between 2007 and 2016 in leading machine learning journals and conferences.

C4.5
***k*-Means**
SVM
Apriori
EM
PageRank
AdaBoost
***k*NN**
Naive Bayes
CART

2007←算法原理→2017

算法实现
Weka, R, C, Matlab

Rank	Acc.	Classifier
32.9	82.0	parRF_t (RF)
33.1	**82.3**	rf_t (RF)
36.8	81.8	svm_C (SVM)
38.0	81.2	svmPoly_t (SVM)
39.4	81.9	rforest_R (RF)
39.6	82.0	elm_kernel_m (NNET)
40.3	81.4	svmRadialCost_t (SVM)
42.5	81.0	svmRadial_t (SVM)
42.9	80.6	C5.0_t (BST)
44.1	79.4	avNNet_t (NNET)

Support vector machine
Neural network
Data set
Objective function
Markov random field
Feature space
Generative model
Linear matrix inequality
Gaussian mixture model
Principal component analysis
Hidden Markov model
Conditional random field
Graphical model
Maximum likelihood estimation
Clustering algorithm
Nearest neighbors
Genetic algorithm
Latent Dirichlet allocation
Gaussian process
Markov chain Monte Carlo

Top 10 algorithms in data mining. 2007
Do we Need Hundreds of classifiers to Solve Real World Classication Problems? 2014
The Top 10 Topics in Machine Learning Revisited: A Quantitative Meta-Study. 2017

今日内容：数据挖掘经典算法概述

- 教材中有的
  - Naive Bayes、 EM、K-means、SVM、kNN
- 决策树
  - ID3、C4.5、CART
- 把若干个分类器整合为一个分类器
  - Bagging、Boosting、AdaBoost [1995]
- 流数据挖掘：频繁项集
- Web中的数据挖掘

今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- 流数据挖掘：频繁项集
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

频繁模式挖掘的内容

- **Frequent Pattern Mining**
  - Frequent Item Set Mining and Association Rule Induction
  - Frequent Sequence Mining
  - Frequent Tree Mining
  - Frequent Graph Mining
- **频繁模式**
  - 频繁项集
  - 频繁子序列
  - 频繁子结构

本节仅覆盖频繁项集的挖掘

关联规则 (association rules)分析

令$I=\{i_1, i_2, \ldots, i_n\}$ 为 items的集合（成为项集），

例如I={milk, bread, butter, beer, diapers}

令$T=\{t, t_2, \ldots, t_n\}$ 为transaction的集合（称为database），

例如上表database含有5个transaction

规则（rule）定义为：$X \Longrightarrow Y$，　$X, Y \subset I$，

例如规则 $\{butter, bread\} \Longrightarrow \{milk\}$

关联规则分析 (Association Rules，又称 Basket Analysis) 用于从大量数据中挖掘出有价值的数据项之间的相关关系。

例如，项集(itemset) X={beer, dispers}的支持度是0.2

规则 $\{butter, bread\} \Longrightarrow \{milk\}$的置信度是0.2/0.2=1

规则 $\{milk, bread\} \Longrightarrow \{butter\}$的lift是0.2/(0.4*0.4)=1.25

规则 $\{milk, bread\} \Longrightarrow \{butter\}$的Conviction是(1-0.4)/(1-0.5)=1.2

支持度 **Support**:

$supp(X) = |\{t \in T; X \subseteq t\}|/|T|$

置信度 **Confidence**

$\text{conf}(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$

提升度 Lift:

$\text{lift}(X) = supp(X \cup Y)/supp(X) \times supp(Y)$

确信度 Conviction:

$\text{conv}(X \Longrightarrow Y) = (1 - supp(Y))/(1 - conf(X \Rightarrow Y))$

Transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

频繁项集分析
Frequent Item Set Mining

- 几个定义

频繁项集分析是关联规则分析的一部分

  - 频繁项：在多个集合中，频繁出现的元素/项，就是频繁项
  - 频繁项集：有一系列集合，这些集合有些相同的元素，集合中同时出现频率高的元素形成一个子集，满足一定阈值条件，就是频繁项集。
  - 极大频繁项集：元素个数最多的频繁项集合，即其任何超集都是非频繁项集。
- 相似性分析 vs.频繁项集分析
  - 相似性分析，研究的对象是集合之间的相似性关系。而频繁项集分析，研究的集合间重复性高的元素子集。
  - 频繁项集的应用：真实超市购物篮的分析，文档或网页的关联程序分析，文档的抄袭分析，生物标志物（疾病与某人生物生理信息的关系）

频繁项集分析：频繁项示例

transaction database

1: $\{a, d, e\}$
2: $\{b, c, d\}$
3: $\{a, c, e\}$
4: $\{a, c, d, e\}$
5: $\{a, e\}$
6: $\{a, c, d\}$
7: $\{b, c\}$
8: $\{a, c, d, e\}$
9: $\{b, c, e\}$
10: $\{a, d, e\}$

frequent item sets

0 items	1 item	2 items	3 items
$\emptyset$: 10	$\{a\}$: 7 $\{b\}$: 3 $\{c\}$: 7 $\{d\}$: 6 $\{e\}$: 7	$\{a, c\}$: 4 $\{a, d\}$: 5 $\{a, e\}$: 6 $\{b, c\}$: 3 $\{c, d\}$: 4 $\{c, e\}$: 4 $\{d, e\}$: 4	$\{a, c, d\}$: 3 $\{a, c, e\}$: 3 $\{a, d, e\}$: 4

***I*={*a*, *b*, *c*, *d*, *e*}上可能有2^5个项集合(item set)**
本例中最小支持度 support是3，归一化的support是0.3
如右表所示，共有16个频繁项

频繁项集分析：购物篮、支持度示例

eight baskets, each consisting of items that are words

(1) {Cat, and, dog, bites}
(2) {Yahoo, news, claims, a, cat, mated, with, a, dog, and, produced, viable, offspring}
(3) {Cat, killer, likely, is, a, big, dog}
(4) {Professional, free, advice, on, dog, training, puppy, training}
(5) {Cat, and, kitten, training, and, behavior}
(6) {Dog, &, Cat, provides, dog, training, in, Eugene, Oregon}
(7) {"Dog, and, cat", is, a, slang, term, used, by, police, officers, for, a, male–female, relationship}
(8) {Shop, for, your, show, dog, grooming, and, pet, supplies}

Among the singleton sets, obviously {cat} and {dog} are quite frequent. "Dog" appears in all but basket (5), so its support is 7, while "cat" appears in all but (4) and (8), so its support is 6. The word "and" is also quite frequent; it appears in (1), (2), (5), (7), and (8), so its support is 5. The words "a" and "training" appear in three sets, while "for" and "is" appear in two each. No other word appears more than once. Suppose that we set our threshold at $s = 3$. Then there are five frequent singleton itemsets: {dog}, {cat}, {and}, {a}, and {training}.

频繁项集分析：双元素频繁集合

Occurrences of doubleton

	training	a	and	cat
dog	4, 6	2, 3, 7	1, 2, 8	1, 2, 3, 6, 7
cat	5, 6	2, 3, 7	1, 2, 5	
and	5	2, 7		
a	none			

Now, let us look at the doubletons. A doubleton cannot be frequent unless both items in the set are frequent by themselves. Thus, there are only ten possible frequent doubletons. There are five frequent doubletons if s = 3; they are

{dog, a} **{dog, and}** **{dog, cat}**
{cat, a} **{cat, and}**

Each appears exactly three times, except for {dog, cat}, which appears five times.

今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

确定性数据中频繁项集挖掘的相关算法

A-Priori算法, [Agrawal and Srikant 1994]

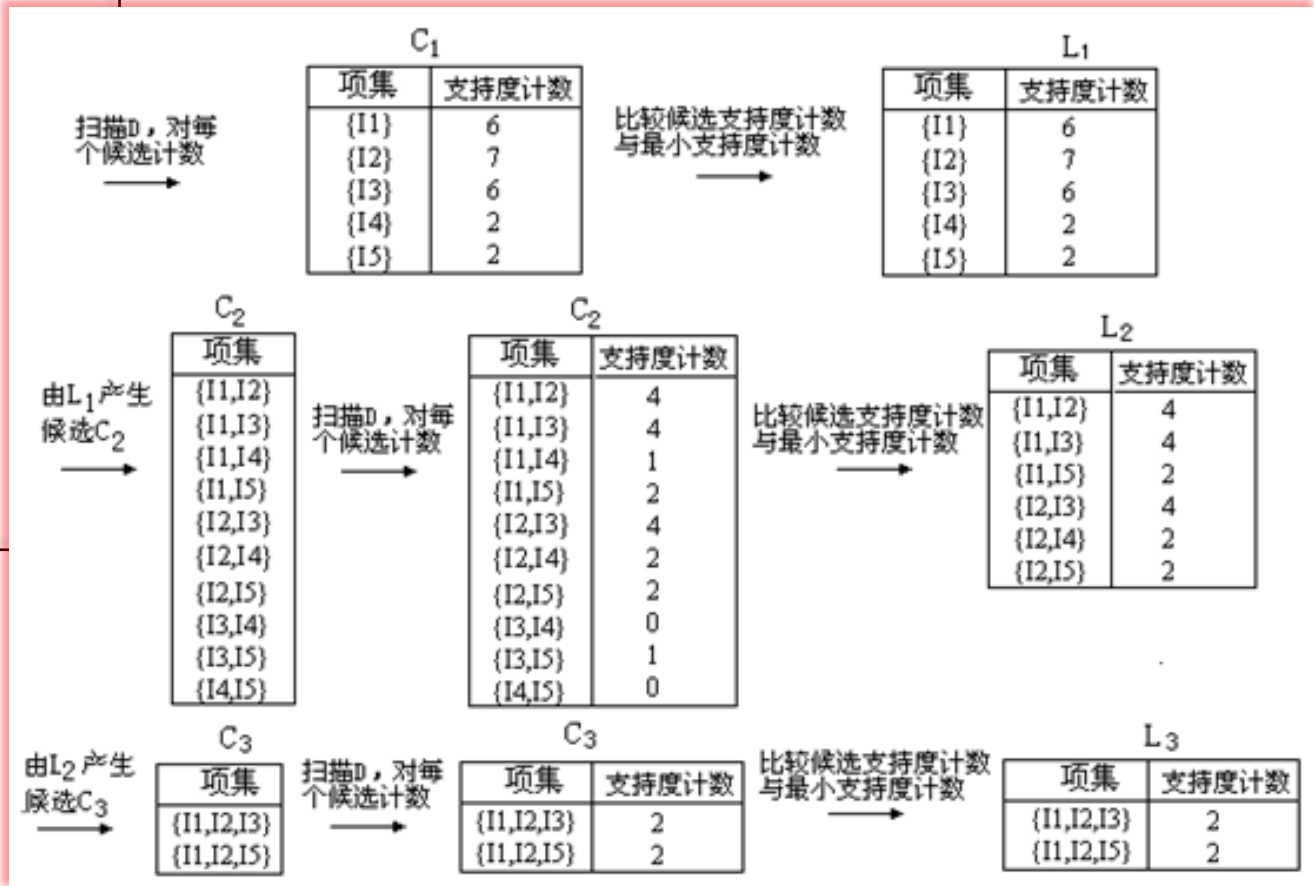
- **频繁项子集定理：频繁项集的子集都是频繁项集，而非频繁项的超集都是非频繁项集。**
- **A-Priori算法是基于先验的算法，在挖掘过程中首先产生候选项集，然后到原数据库中检验这些候选项集是否是频繁的。采用的是自底向上的产生-检验的模式来搜索所有的频繁项集。**
  - 首先，产生长度为1的候选项集C_1，计算C_1中所有候选项的支持度来确定哪些是频繁的，这些频繁的长度为1的候选项保存在L_1中；
  - 然后从L_1中产生出所有长度为2的候选项集，保存在C_2中，计算C_2中所有候选项的支持度来确定哪些是频繁的，保存在L_2中；
  - 以此类推，直到没有产生新的候选项集，即C_k为空(k>1)。这样，算法就可以输出所有的频繁项集。
- **由于算法是基于产生-检验的，在挖掘过程中要经常回到原数据库中检验候选项集是否频繁，因此当数据库比较大时，该方法效率不高。**

A-Priori算法示例

原始数据

TID	List of item_ID's
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

A-Priori算法步骤

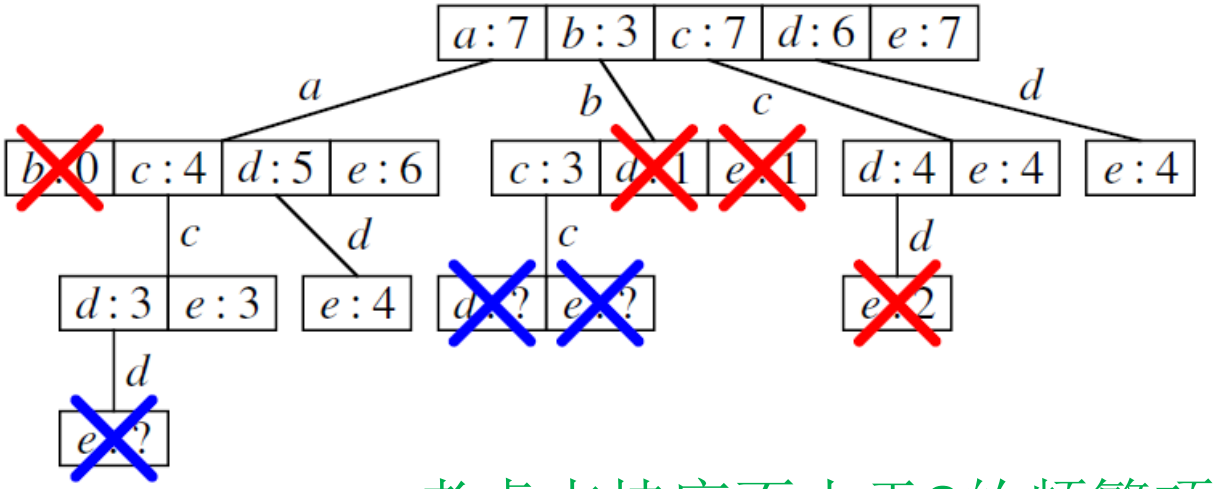


频繁项集挖掘算法 FP-Growth
http://blog.sina.com.cn/s/blog_5357c0af0101jq6z.html Retrieved: 20160429

广度优先搜索

1: $\{a, d, e\}$
2: $\{b, c, d\}$
3: $\{a, c, e\}$
4: $\{a, c, d, e\}$
5: $\{a, e\}$
6: $\{a, c, d\}$
7: $\{b, c\}$
8: $\{a, c, d, e\}$
9: $\{b, c, e\}$
10: $\{a, d, e\}$

Prefix Trees



考虑支持度不小于3的频繁项集

Apriori执行Breadth-first，每次搜索的是同一层的item

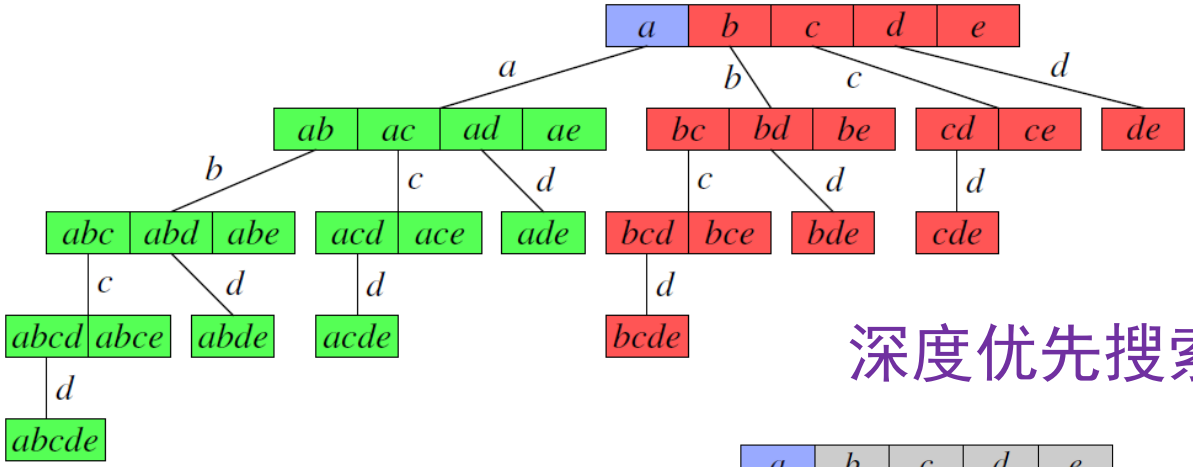
① 第1层：{a, b, c, d, e}

② 第2层：{ab, ac, ad, ae, bc, bd, be, de, ce, de}

③ 第3层：{acd, ace, ade, bcd, bce , cde}

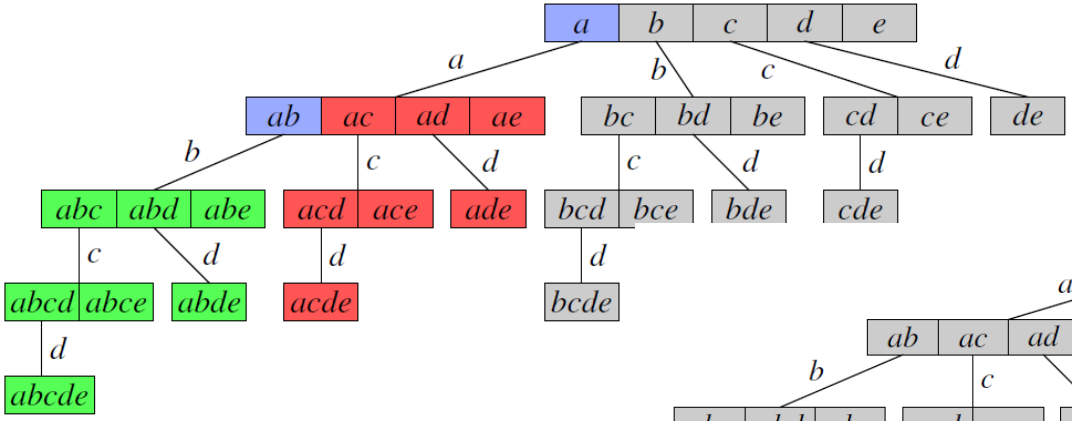
④ 第4层：{acde}

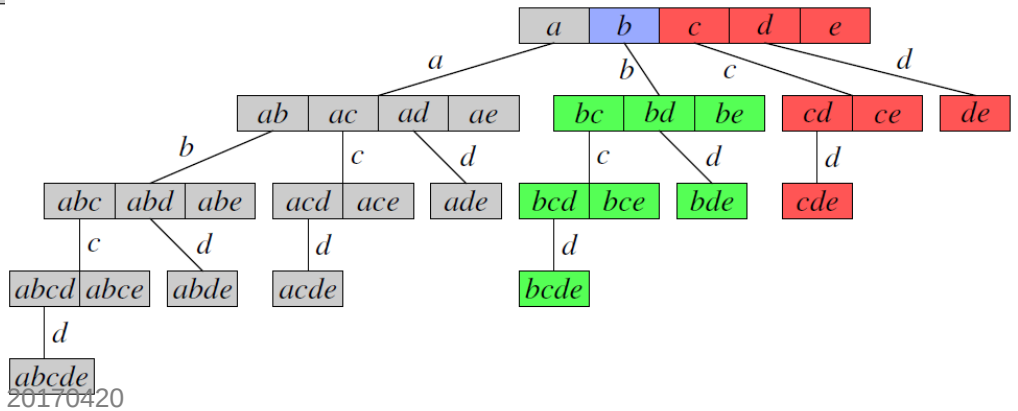
频繁项的其他查找思路：深度优先搜索



Prefix Trees

深度优先搜索：可分解为子问题

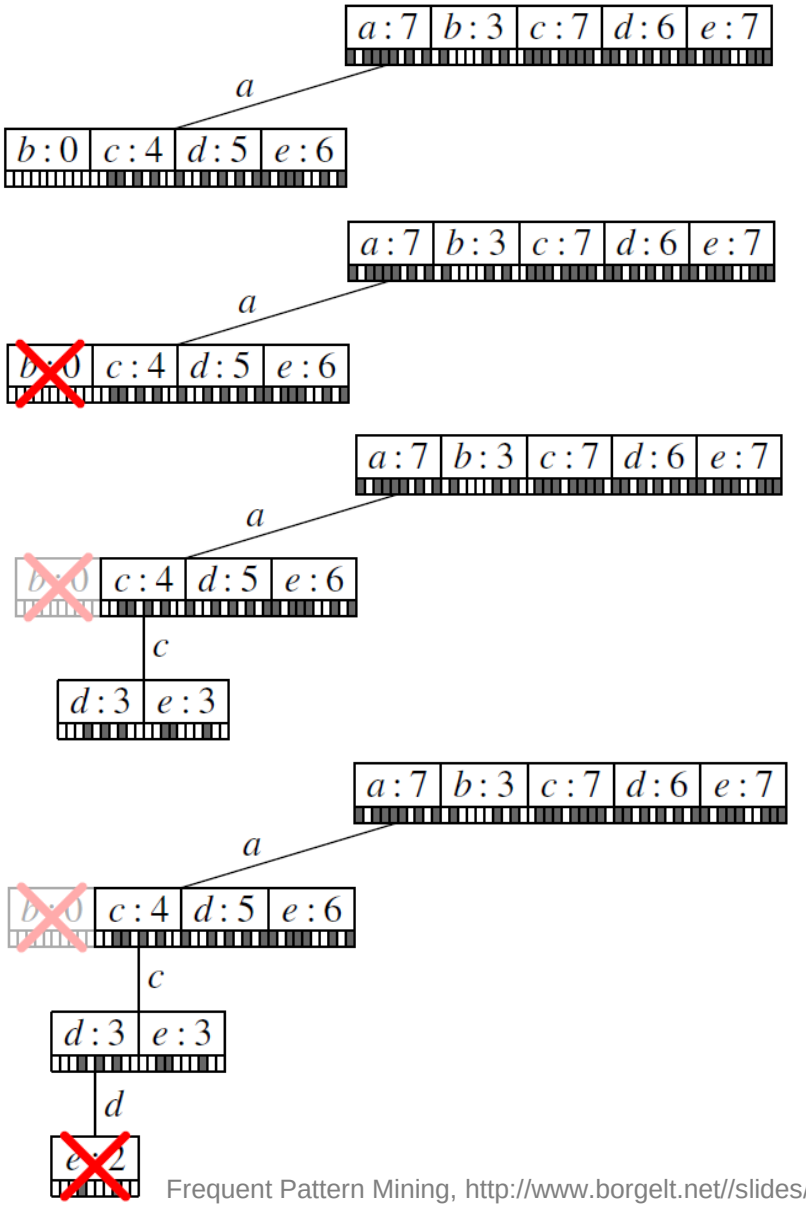




今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

Eclat算法 [Zaki, Parthasarathy, Ogihara, and Li 1997]



【示例】找出支持度>=3的频繁项集

先处理支持度最高的频繁项a：

（1）找出频繁项ab, ac, ad, ae；删除支持度<3的项ab

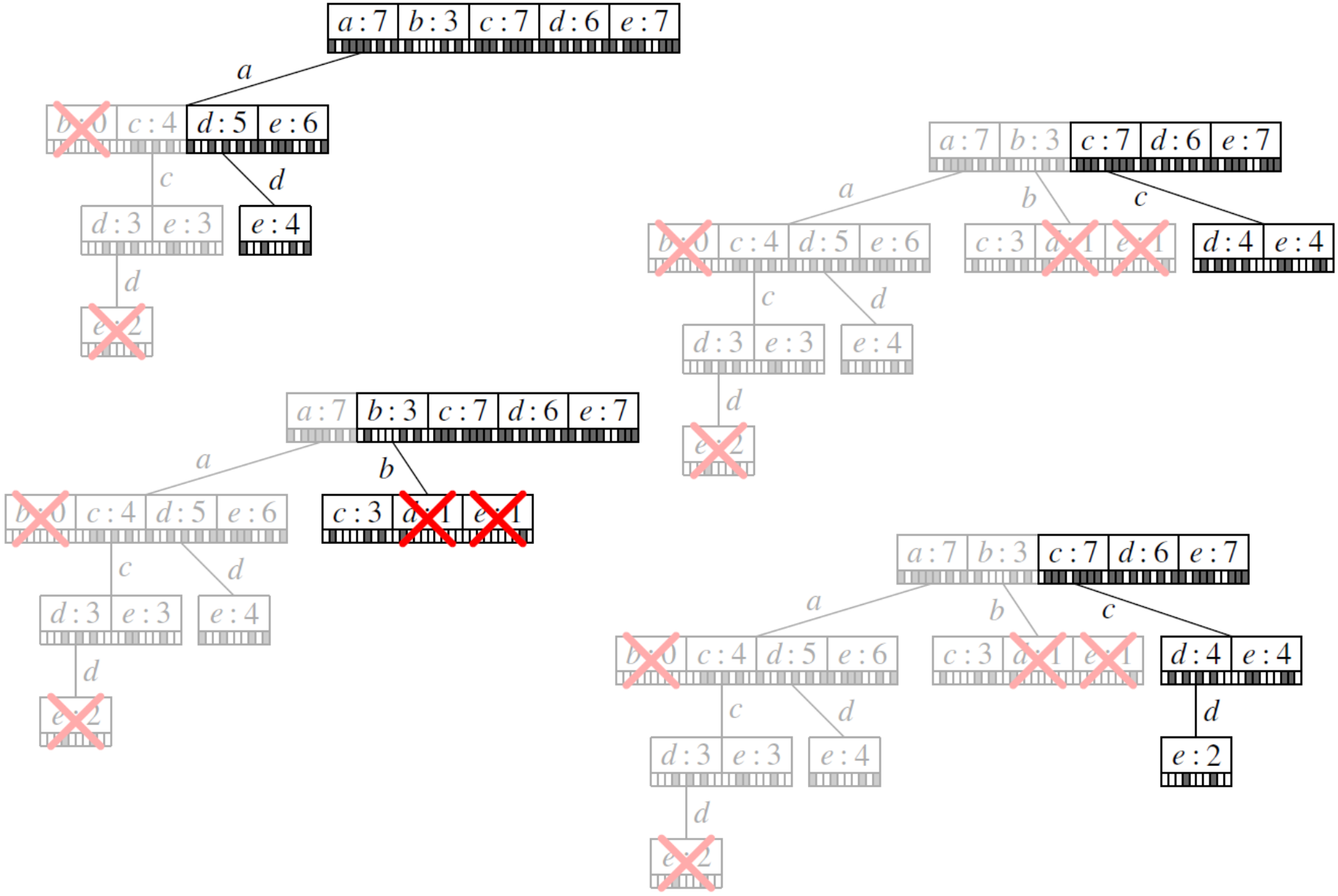
（2）找出频繁项acd, ace

（3）找出频繁项acde；删除支持度<3的项acde

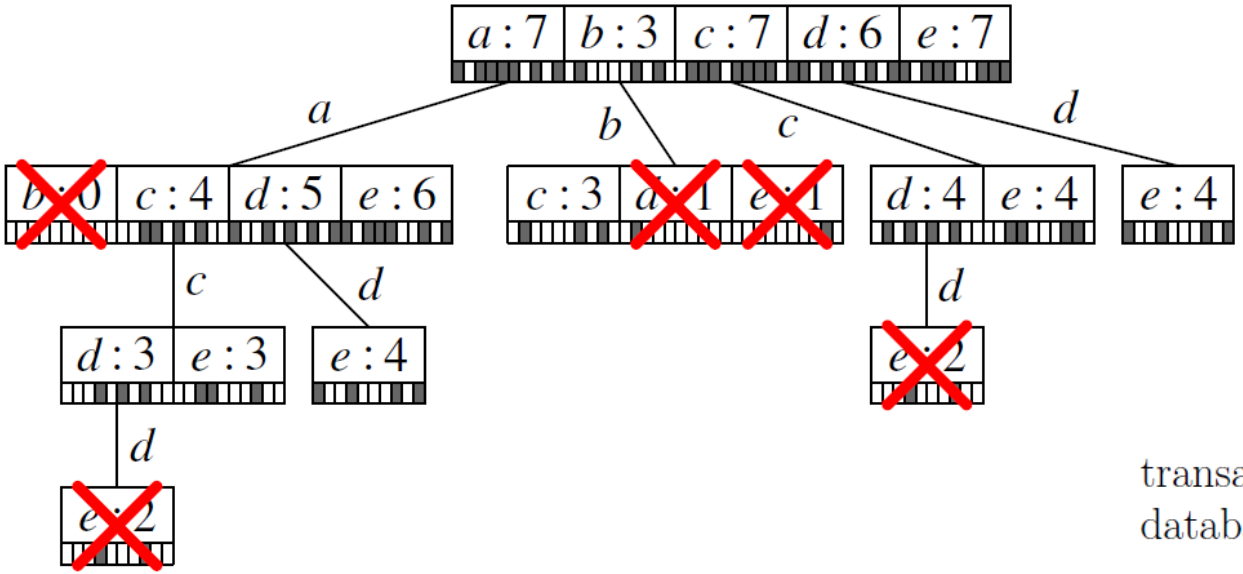
……

transaction database	lexicographically sorted
a, d, e	a, c, d
b, c, d	a, c, d, e
a, c, e	a, c, d, e
a, c, d, e	a, c, e
a, e	a, d, e
a, c, d	a, d, e
b, c	a, e
a, c, d, e	b, c
b, c, e	b, c, d
a, d, e	b, c, e

购物篮按照字母顺序排序


a : 7 | b : 3 | c : 7 | d : 6 | e : 7
a
b : 0 | c : 4 | d : 5 | e : 6
c
d
d : 3 | e : 3
e : 4
d
e : 2
a : 7 | b : 3 | c : 7 | d : 6 | e : 7
a
b
c
b : 0 | c : 4 | d : 5 | e : 6
c : 3 | d : 1 | e : 1
d : 4 | e : 4
c
d
d : 3 | e : 3
e : 4
d
e : 2
a : 7 | b : 3 | c : 7 | d : 6 | e : 7
a
b
b : 0 | c : 4 | d : 5 | e : 6
c : 3 | d : 1 | e : 1
c
d
d : 3 | e : 3
e : 4
d
e : 2
a : 7 | b : 3 | c : 7 | d : 6 | e : 7
a
b
c
b : 0 | c : 4 | d : 5 | e : 6
c : 3 | d : 1 | e : 1
d : 4 | e : 4
c
d
d : 3 | e : 3
e : 4
d
e : 2

Eclat算法[Zaki, Parthasarathy, Ogihara, and Li 1997]



【示例】找出支持度不小于3的频繁项集

目前还留在图上的频繁项就是所有支持度不小于3的频繁项集

{acd, ace, ade, ae, bc, cd, ce, de}

transaction database		lexicographically sorted
a, d, e		a, c, d
b, c, d		a, c, d, e
a, c, e		a, c, d, e
a, c, d, e		a, c, e
a, e		a, d, e
a, c, d	购物篮按照字母顺序排序	a, d, e
b, c		a, e
a, c, d, e		b, c
b, c, e		b, c, d
a, d, e		b, c, e

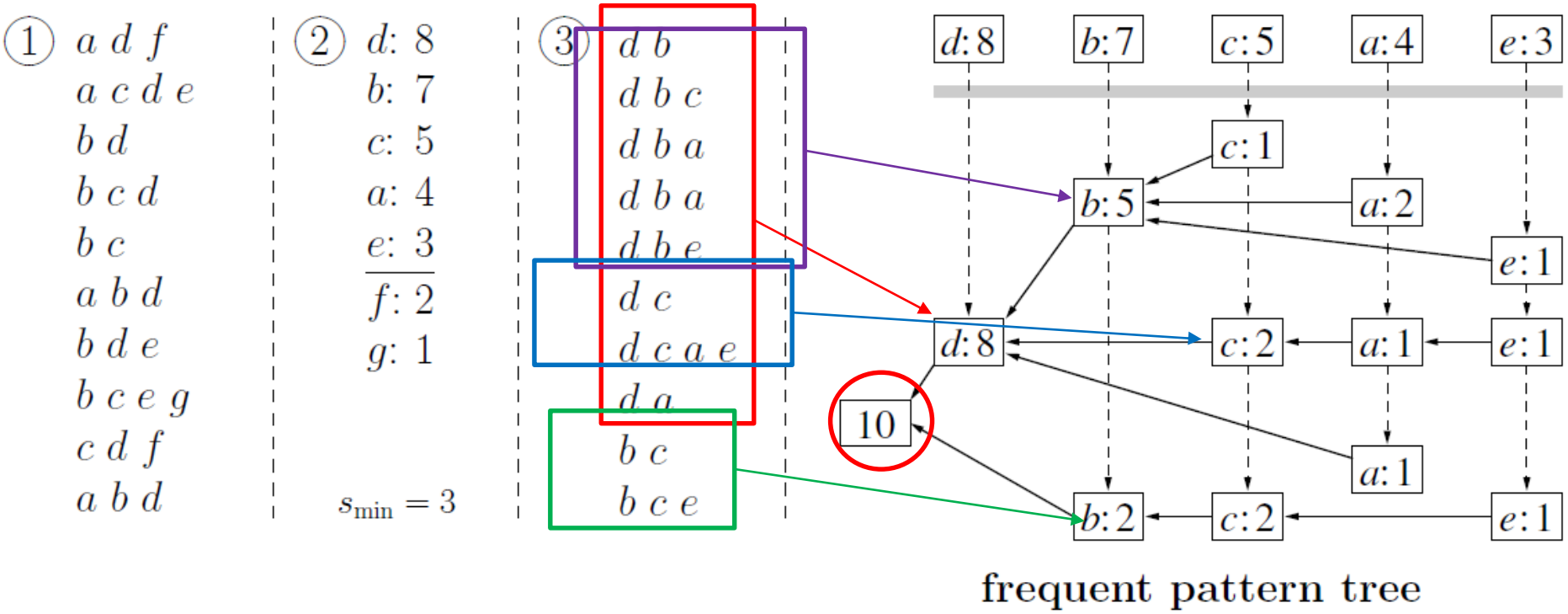
今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

FP-growth (Frequent Pattern growth) 算法, [Han, Pei, and Yin 2000]

- 与**Apriori** 不同的是，该算法不用产生候选项集，只需要测试支持度计数，因此其效率是高于**Apriori** 算法的。该算法主要由两步操作组成：（1）构造**FP-tree**；（2）挖掘频繁项集。
- 主要操作如下：首先，对数据库进行第一次扫描检验每个项的支持度来得到频繁项集，然后对数据库进行第二次扫描来建立**FP-tree**。在建树过程中，所有项的支持度按降序排列，插入到树中，这样支持度大的项可以共享，从而建立一棵最小的树，这棵树中包含的是数据库中长度为**1** 的频繁项集。然后从该树的最后一个结点***x***（叶子结点）开始，从下向上依次产生这些结点的子数据库（即以该结点***x*** 为前缀的项集），将该子数据库中每个项集的支持度与用户指定的阈值比较即可知该项集是否频繁；对于***x*** 的子数据库中的项***y*** 可以产生***{x, y}***的子数据库……，将这种方法运用到**FP-tree** 的每个结点中，就可以得到所有的频繁项集。

FP-growth算法, [Han, Pei, and Yin 2000]



【示例】找出支持度>=3的频繁项集

① 原始database

② database中item的支持度降序排列

③ item set根据item的支持度降序排列

生成FP Tree

根节点

依次产生以节点x为前缀的子树

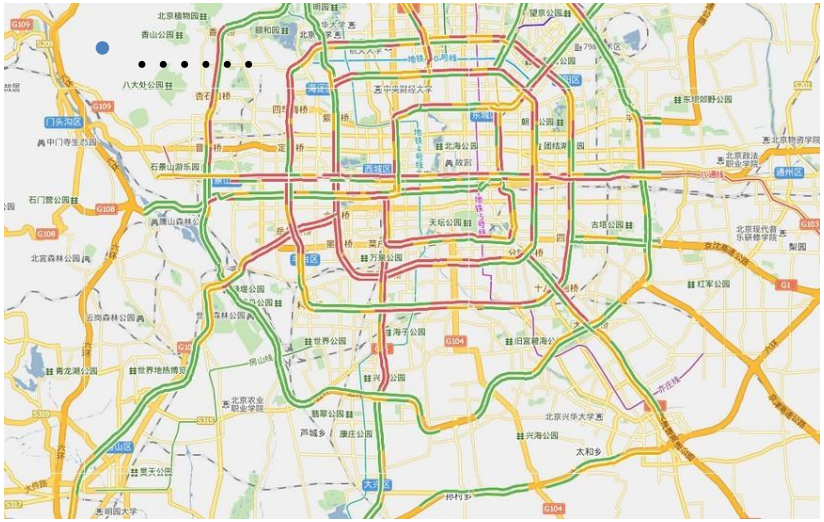
……直至完成整棵树的建立

今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

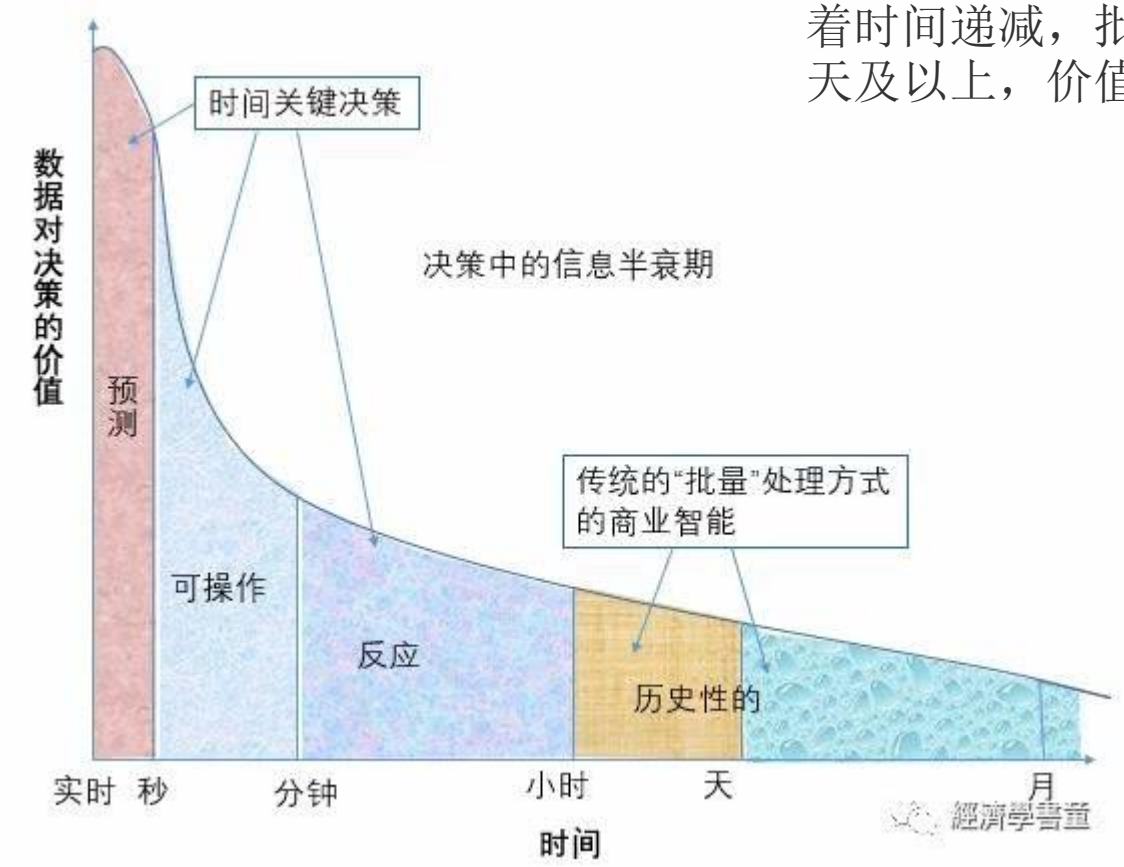
不确定数据频繁项集挖掘

- **不确定数据广泛存在（动态数据、流数据）**
  - 用户位置数据如来自 GPS 数据的位置信息
  - 用户兴趣
  - 热点信息、产品
  - 实时路况
  - 物联网中各类传感器的数据
  - ……



为什么需要对流数据进行分析?

规律：数据用于决策的价值随着时间递减，批处理：针对一天及以上，价值没有那么大了



学习笔记DB1：流数据分析与处理
https://cloud.tencent.com/developer/news/313236 Retrieved:20190429

基于不确定数据的频繁项集的挖掘方法

- 在传统的数据库中, 如果X 是一个频繁项集, 那么它的支持度sup(x)必须大于用户给定的最小支持度min_sup。在不确定性数据库中, 由于项集以概率形式存在, 所以用期望支持度esup(x)代替项集x 在不确定性数据库中的支持度。
  - 最为经典的算法是基于先验的Apriori 算法和基于树结构的FP_Growth 算法均已有改进算法
  - A-priori →U-Apriori
  - FP-growth→ UF-growth

数据流中的频繁项集分析

- **面临的问题：**
- **1）数据流可能速率很高，无法存储整个流进行分析**
- **2）随时间的推移，频繁项集也会发生变化**
- **解决方法：**
- **1）抽样技术**
  - 抽样：定期收集数据流，存为文件后，再分析。分析期间，不再将数据流放入正在分析的文件。不过，可以存储为另一个文件，供下次分析。
- **2）窗口的衰减模型**
  - 窗口衰减模型：可以将倒数第i个出现的购物篮赋予$(1-c)^i$ 的权值。这样就形成了流中的一个衰减窗口。当某个项集出现在当前购物篮中，并且其所有真子集已经开始计数，那么此时开始对此项集开始计数。这样需要计数的项集的规模就不会太大。由于窗口不断衰减，将所有计数值都乘以(1-c)，并去掉计数低于1/2的项集。

小结：频繁项集挖掘

- **确定数据的频繁项集的挖掘**
  - A-Priori [1994]
  - FP-growth (Frequent Pattern growth) [2000]
- **不确定数据的频繁项集的挖掘**
  - 支持度为概率值→期望支持度esup(x)
  - A-priori →U-Apriori
  - FP-growth→ UF-growth

  - 1）抽样技术
  - 2）窗口的衰减模型

今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

哈希函数：定义

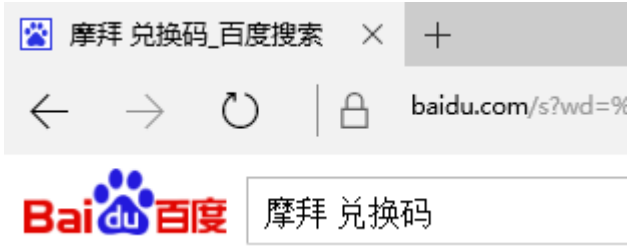
- 散列函数H 是一个公开的函数，它将任意长度的消息M变换为固定长度的散列码h 。
- h＝H (M)
- M：变长消息，H(M)：定长的散列值

- 散列函数是一种算法，算法的输出内容称为散列码或者消息摘要。
- 散列函数又称为：哈希（ Hash ）函数、数字指纹（Digital fingerprint)、压缩（Compression)函数、数据认证码（Data Authentication Code）等

哈希函数：哈希的常用算法

- **将一个消息串映射为一个整数的简单思路**
  - h(k) = k mod m
  - h(k) = k(k+3) mod m
  - ……
- **著名的Hash算法**
  - MD5 和 SHA1 应用最广泛，它们都是以 MD4 为基础设计。
  - **MD4** (RFC 1320）是 MIT 的Ronald L. Rivest在 1990 年设计的，MD 是 Message Digest 的缩写。它适用在**32位**字长的处理器上用高速软件实现--它是基于 32位操作数的位操作来实现的。
  - **MD5** (RFC 1321）是 Rivest 于1991年对MD4的改进版本。它对输入仍以512位分组，其输出是**4个32位**字的级联，与 MD4 相同。MD5比MD4来得复杂，并且速度较之要慢一点，但更安全，在抗分析和抗差分方面表现更好
  - **SHA1** 是由NIST NSA设计为同DSA一起使用的，它对长度小于2^64位的输入，产生长度为**160bit**的散列值，因此抗穷举（brute-force）性更好。SHA-1 设计时基于和MD4相同原理，并且模仿了该算法。

为什么要使用哈希函数



https://www.baidu.com/s?wd=%E6%91%A9%E6%8B%9C%20%E5%85%91%E6%8D%A2%E7%A0%81&rsv_spt=1&rsv_iqid=0xcb35933800255f78&issp=1&f=8&rsv_bp=1&rsv_idx=2&ie=utf-8&rqlang=cn&tn=98012088_5_dg&rsv_enter=1&oq=%25E5%2590%2588%25E8%2582%25A5%25E6%2591%25A9%25E6%258B%259C%2520%25E5%2585%2591%25E6%258D%25A2%25E7%25A0%2581&rsv_t=41b0PNbxN9XytGpnTAti%2F17a8oudrVsIKYpbtyrKmjJwqpQ9Xaxs3wrzGgHzT4%2BwvxrTRw&inputT=223&rsv_pq=d567c8990027fc9e&rsv_sug3=135&rsv_sug2=0&rsv_sug4=223

哈希函数应用示例：信息指纹

- 任何一段信息文字，都可以对应一个不太长的随机数，作为区别它和其它信息的指纹（Fingerprint)。只要算法设计的好，任何两段信息的指纹都很难重复，就如同人类的指纹一样。信息指纹在加密、信息压缩和处理中有着广泛的应用。
- URL的信息指纹：（网络爬虫需要存储URL）假定网址的平均长度为一百个字符，那么存贮 200 亿个网址本身至少需要 2 TB，即两千 GB 的容量。如果能够找到一个函数，将这 200 亿个网址随机地映射到128 二进位即 16 个字节的整数空间，这样每个网址只需要占用 16 个字节。

哈希函数应用示例

【百度面试题】一个大的含有50M个URL的记录，一个小的含有500个URL的记录，找出两个记录里相同的URL。

- 先用包含500个URL的文件创建一个hash_set。
- 然后遍历50M的URL记录，如果URL在hash_set中，则输出此URL并从hash_set中删除这个URL 。
- 所有输出的URL就是两个记录里相同的url。

数据的存储：数组、链表、哈希表

- 数组：存储区间是连续的，占用内存严重，故空间复杂的很大。但数组的二分查找时间复杂度小，为O(1)；数组的特点是：寻址容易，插入和删除困难；
- 链表：存储区间离散，占用内存比较宽松，故空间复杂度很小，但时间复杂度很大，达O(N)。链表的特点是：寻址困难，插入和删除容易。
- 哈希表：能否综合两者的特性，做出一种寻址容易，插入删除也容易的数据结构？答案是肯定的，这就是哈希表。哈希表（(Hash table）既满足了数据的查找方便，同时不占用太多的内容空间，使用也十分方便。

哈希表(Hash table)

- 散列表（**Hash table**，也叫哈希表），是根据关键码值(**Key value**)而直接进行访问的数据结构。它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。
- 哈希法又称散列法、杂凑法以及关键字地址计算法等，相应的表称为哈希表。这种方法的基本思想是：首先在元素的关键字**k**和元素的存储位置**p**之间建立一个对应关系**f**，使得**p=f(k)**，**f**称为哈希函数。创建哈希表时，把关键字为**k**的元素直接存入地址为**f(k)**的单元；以后当查找关键字为**k**的元素时，再利用哈希函数计算出该元素的存储位置**p=f(k)**，从而达到按关键字直接存取元素的目的。
- 当关键字集合很大时，关键字值不同的元素可能会映象到哈希表的同一地址上，即 **k1≠k2** ，但 **H(k1)=H(k2)**，这种现象称为冲突，此时称**k1**和**k2**为同义词。

哈希表(Hash table)：哈希函数构造

- 构造原则：①函数本身便于计算；②计算出来的地址分布均匀，即对任一关键字k，f(k) 对应不同地址的概率相等，目的是尽量减少冲突。
- 1. 直接寻址法：取关键字或关键字的某个线性函数值为散列地址。即H(key)=key或H(key) = a ŀkey + b，其中a和b为常数。
- 2. 数字分析法：如果事先知道关键字集合，可以从关键字中选出分布较均匀的若干位，构成哈希地址。
- 3. 平方取中法：当无法确定关键字中哪几位分布较均匀时，可以先求出关键字的平方值，然后按需要取平方值的中间几位作为哈希地址。
- 4. 折叠法：将关键字分割成位数相同的几部分，最后一部分位数可以不同，然后取这几部分的叠加和（去除进位）作为散列地址。
- 5. 随机数法：选择一随机函数，取关键字的随机值作为散列地址，通常用于关键字长度不同的场合。
- 6. 除留余数法：取关键字被某个不大于散列表表长m的数p除后所得的余数为散列地址。即 H(key) = key MOD p, p<=m。

哈希表(Hash table)：冲突处理

- 1.开放定址法。也称再散列法，当关键字key的哈希地址p=H(key)出现冲突时，以p为基础，产生另一个哈希地址p1，如果p1仍然冲突，再以p1为基础，产生另一个哈希地址p2，…，直到找出一个不冲突的哈希地址pi。
- 2.再哈希法。同时构造多个不同的哈希函数：$H_i=RH_1(key)$ i=1,2,…,k。当哈希地址$H_i=RH_1(key)$发生冲突时，再计算$H_i=RH_2(key)$…，直到冲突不再产生。
- 3.链地址法。将所有哈希地址为i的元素构成一个称为同义词链的单链表，并将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。
- 4.建立公共溢出区。将哈希表分为基本表和溢出表两部分，凡是和基本表发生冲突的元素，一律填入溢出表。

哈希表应用示例

百度面试题：搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。请统计最热门的10个查询串，要求使用的内存不能超过1G。

虽然有一千万个Query，但是由于重复度比较高，因此事实上只有300万的Query，每个Query255Byte，因此我们可以考虑把他们都放进内存中去，而现在只是需要一个合适的数据结构，Hash Table是一种可能的选择，因为Hash Table的查询速度非常的快，几乎是O(1)的时间复杂度。

那么，我们的算法就有了：维护一个Key为Query字串，Value为该Query出现次数的HashTable，每次读取一个Query，如果该字串不在Table中，那么加入该字串，并且将Value值设为1；如果该字串在Table中，那么将该字串的计数加一即可。

小结：Hash function、Hash table

- **Hash function**
  - 又称为：哈希（ Hash ）函数、数字指纹（Digital fingerprint)、压缩（Compression)函数、数据认证码（ Data Authentication Code）等
  - h＝H (M), M：变长消息，H(M)：定长的散列值
  - MD5 和 SHA1
- **Hash table**
  - 创建哈希表时，把关键字为k的元素直接存入地址为f(k)的单元；以后当查找关键字为k的元素时，再利用哈希函数计算出该元素的存储位置p=f(k)，从而达到按关键字直接存取元素的目的。
  - 冲突：k1≠k2 ，但 H(k1)=H(k2)

今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

需求：判断一个元素是否在一个集合中

- **在日常生活中，包括在设计计算机软件时，我们经常要判断一个元素是否在一个集合中。**
  - 比如在字处理软件中，需要检查一个英语单词是否拼写正确（也就是要判断它是否在已知的字典中）；
  - 在 FBI，一个嫌疑人的名字是否已经在嫌疑名单上；
  - 电子邮件系统中，判断Email地址是否在黑名单中；
  - 在网络爬虫里，一个网址是否被访问过等等。
- **最直接的方法就是将集合中全部的元素存在计算机中，遇到一个新元素时，将它和集合中的元素直接比较即可。**

集合巨大时哈希表效率低

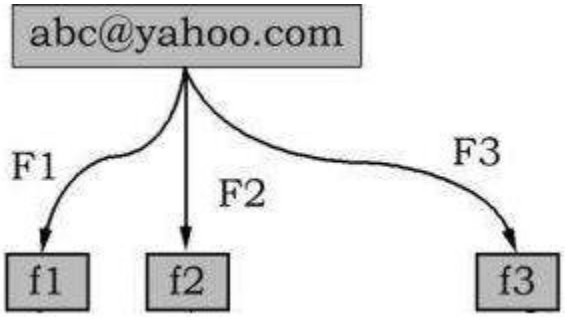
- 集合巨大：如一个象 yahoo, Hotmail 和 Gmail 那样的email提供商，总是需要过滤来自发送垃圾邮件的人（spamer）的垃圾邮件。一个办法就是记录下那些发垃圾邮件的 email 地址。由于那些发送者不停地在注册新的地址，全世界少说也有几十亿个发垃圾邮件的地址，将他们都存起来则需要大量的网络服务器。
- 一般来讲，计算机中的集合是用哈希表来存储的。好处是快速准确，缺点是费存储空间。当集合比较小时，问题不显著，但是当集合巨大时，哈希表存储效率低的问题就显现出来了。
- 如果用哈希表，每存储一亿个 email 地址， 就需要 1.6GB 的内存（用哈希表实现的具体办法是将每一个 email 地址对应成一个八字节的信息指纹，然后将这些信息指纹存入哈希表，由于哈希表的存储效率一般只有 50%，因此一个 email 地址需要占用十六个字节。一亿个地址大约要 1.6GB， 即十六亿字节的内存）。因此存贮几十亿个邮件地址可能需要上百 GB 的内存。

布隆过滤器（Bloom Filter）

- 布隆过滤器是由布隆（Burton Howard Bloom）在1970年提出的。它实际上是由**一个很长的二进制向量**和**一系列随机映射函数**组成，布隆过滤器可以用于检索一个元素是否在一个集合中。
- 它的优点是空间效率和查询时间都远远超过一般的算法，缺点是**有一定的误识别率**（假正例False positives，即Bloom Filter报告某一元素存在于某集合中，但是实际上该元素并不在集合中）和删除困难，但是没有识别错误的情形（即假反例False negatives，如果某个元素确实没有在该集合中，那么Bloom Filter 是不会报告该元素存在于集合中的，所以不会漏报）。

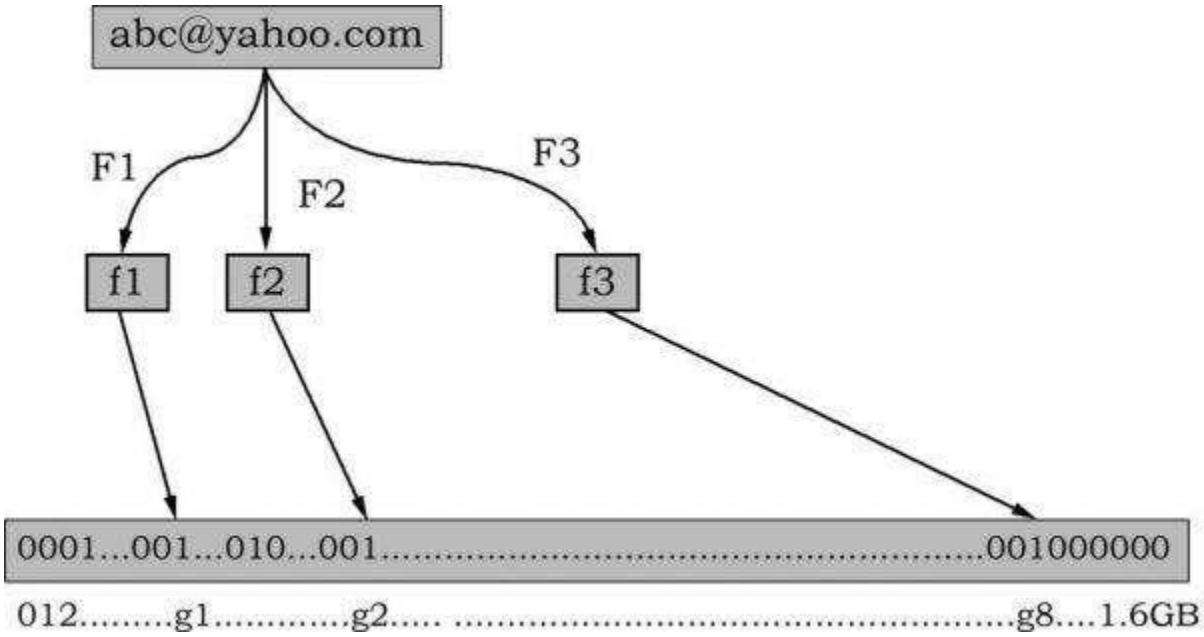
布隆过滤器工作原理
布隆过滤器的建立

- 假定我们存储一亿个电子邮件地址，我们先建立一个十六亿二进制（比特），即两亿字节的向量，然后将这十六亿个二进制全部设置为零。
- 对于每一个电子邮件地址 X，我们用八个不同的随机数产生器（F1,F2, ...,F8） 产生八个信息指纹（f1, f2, ..., f8）。



布隆过滤器工作原理
布隆过滤器的建立

- 再用一个随机数产生器 G 把这八个信息指纹映射到 1 到十六亿中的八个自然数 g1，g2，...，g8。
- 现在我们把这八个位置的二进制全部设置为一。当我们对这一亿个 email 地址都进行这样的处理后。一个针对这些 email 地址的布隆过滤器就建成了。



布隆过滤器原理
检测一个可疑的电子邮件地址

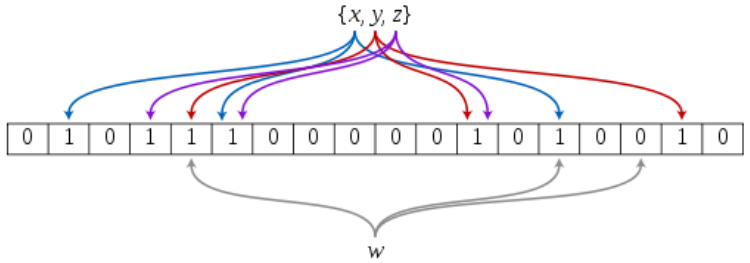
- 现在，让我们看看如何用布隆过滤器来检测一个可疑的电子邮件地址 Y 是否在黑名单中。我们用相同的八个随机数产生器（F1, F2, ..., F8）对这个地址产生八个信息指纹 s1,s2,...,s8，然后将这八个指纹对应到布隆过滤器的八个二进制位，分别是 t1,t2,...,t8。如果 Y 在黑名单中，显然，t1,t2,..,t8 对应的八个二进制一定是一。这样在遇到任何在黑名单中的电子邮件地址，我们都能准确地发现。
- 布隆过滤器决不会漏掉任何一个在黑名单中的可疑地址。但是，它有一条不足之处。也就是它有极小的可能将一个不在黑名单中的电子邮件地址判定为在黑名单中，因为有可能某个好的邮件地址正巧对应个八个都被设置成一的二进制位。好在这种可能性很小。我们把它称为误识概率。在上面的例子中，误识概率在万分之一以下。

布隆过滤器思想应用示例

【百度面试题】2.5亿个int数，可能有相同的。统计出这里头不同的数有多少个？只有2g内存。

【百度面试题】给40亿个不重复的unsigned int的整数，没排过序的，然后再给几个数，如何快速判断这几个数是否在那40亿个数当中?

小结： Bloom Filter



- **Bloom Filter 优点**
  - 相比于其它的数据结构，布隆过滤器在空间和时间方面都有巨大的优势。布隆过滤器存储空间和插入/查询时间都是常数。另外, Hash函数相互之间没有关系，方便由硬件并行实现。布隆过滤器不需要存储元素本身，在某些对保密要求非常严格的场合有优势。
- **Bloom Filter 缺点**
  - 误算率（False Positive）。随着存入的元素数量增加，误算率随之增加。但是如果元素数量太少，则使用散列表足矣。
- **Bloom Filter 用例**
  - Google 著名的分布式数据库 Bigtable 用布隆过滤器来查找不存在的行或列，以减少磁盘查找的IO次数。
  - Squid 网页代理缓存服务器在 cache digests 中使用了也布隆过滤器。
  - Venti 文档存储系统也采用布隆过滤器来检测先前存储的数据。
  - SPIN 模型检测器用布隆过滤器在大规模验证问题时跟踪可达状态空间。
  - Google Chrome浏览器使用了布隆过滤器加速安全浏览服务。

布隆过滤器(Bloom Filter)详解
http://www.cnblogs.com/haippy/archive/2012/07/13/2590351.html

今日内容：数据挖掘经典算法概述(2)

- 教材中有的:Naive Bayes、 EM、K-means、SVM、kNN
- 决策树:ID3、C4.5、CART
- 若干个分类器整合为一个:Bagging、Boosting、AdaBoost
- **流数据挖掘：频繁项集**
  - 确定性数据中频繁项集挖掘的相关算法
    - A-Priori [1994]
    - Eclat [1997]
    - FP-growth (Frequent Pattern growth) [2000]
  - 不确定数据的频繁项集的挖掘
- **Web中的数据挖掘**
  - 哈希(hash) 与哈希表(Hash Table)
  - 布隆过滤器(Bloom Filter)
  - 近似重复检测

重复文档/近似重复文档

- **网上到处都是相同的内容**
- **完全复制 *Duplication***
  - 可以通过指纹（fingerprints）来检测精确匹配
- **大多数情况是近似重复 *Near-Duplication***
  - E.g., 两份文本仅仅是日期不同
  - 通过编辑距离计算语法上的相似性
  - 通过一定的阈值来检测近似复制
    - E.g., Similarity > 80% => 文档近似复制
    - 通过阈值来检测是不可传递的 （AB近似，BC近似不能推断AC近似）

相似性计算

- **特征**
  - 对文档进行分割 (自然或人造的断点来断开)
  - 搭叠Shingles (N元词Word N-Grams)
  - ***a rose is a rose is a rose →***

    a_rose_is_a

    rose_is_a_rose

    is_a_rose_is

    a_rose_is_a …
- **文档(即Shingles集合)间相似性**
  - 交集Set intersection
  - 交集的大小/并集的大小

给定正整数k及文档d的一个词项序列，可以定义文档d的**k-shingle为**d中所有k个连续词项构成的序列。

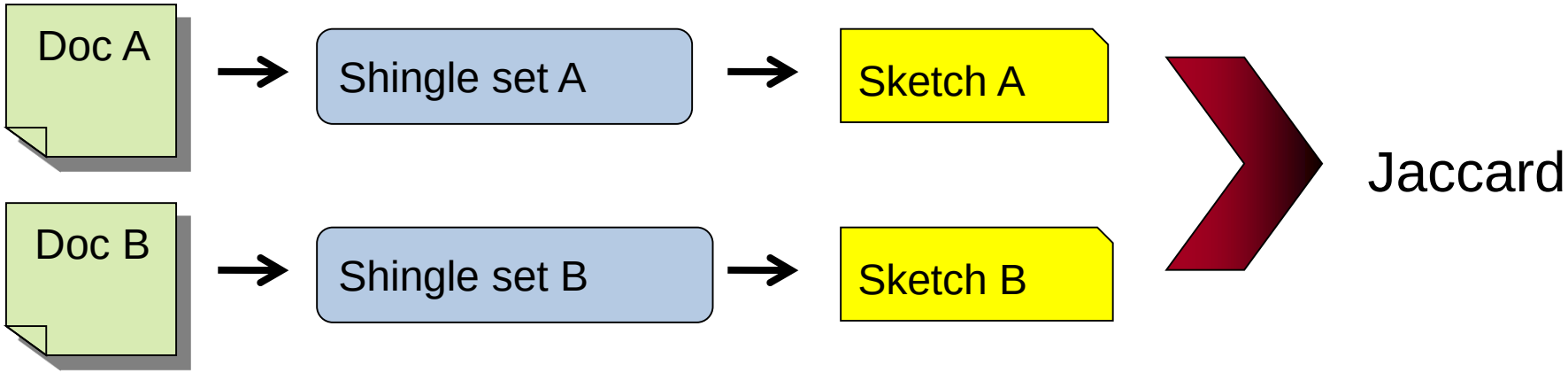
直观上看，如果两个文档的shingle集合几乎一样，那么它们就满足近似重复。

Jaccard系数：衡量重复度

搭叠 + 交集

$$Jaccard(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

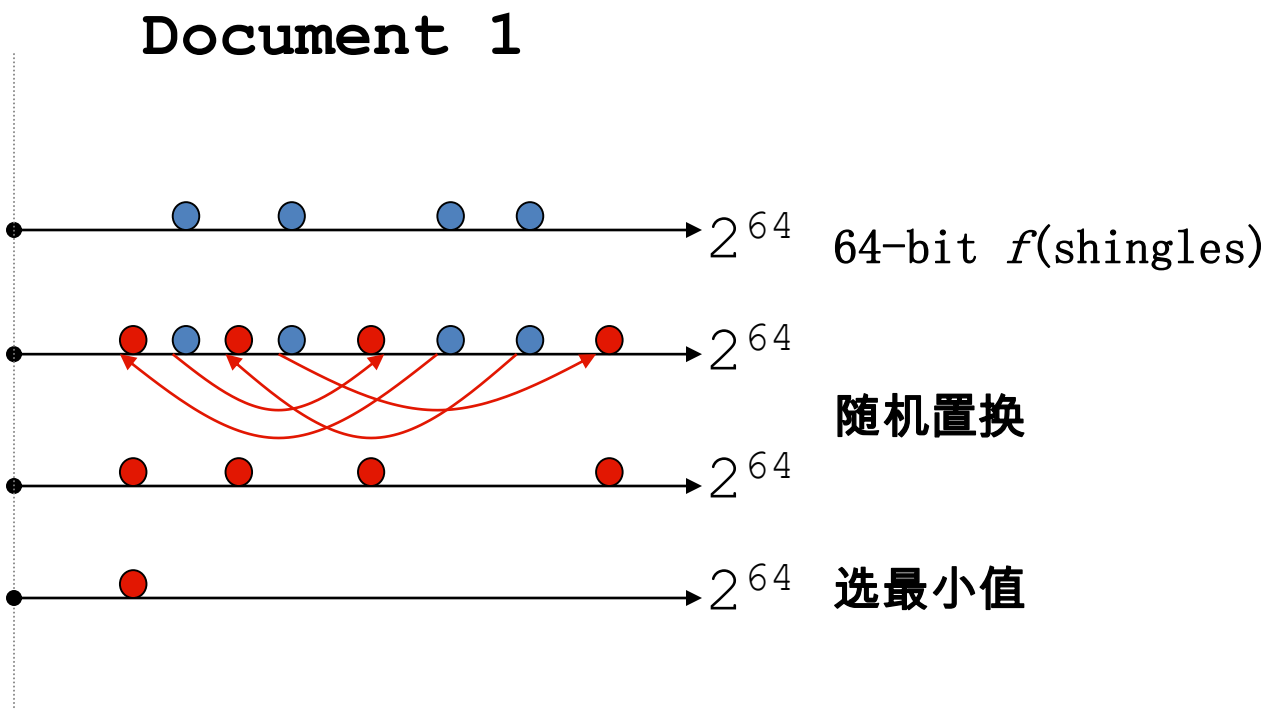
- 计算所有文档对之间搭叠的*精确*交集是非常费时而且难以处理的
  - 使用一种聪明的方式从Shingles中选出一个子集(素描*sketch*)来近似计算
- **在素描Sketch上计算 交集大小/并集大小**



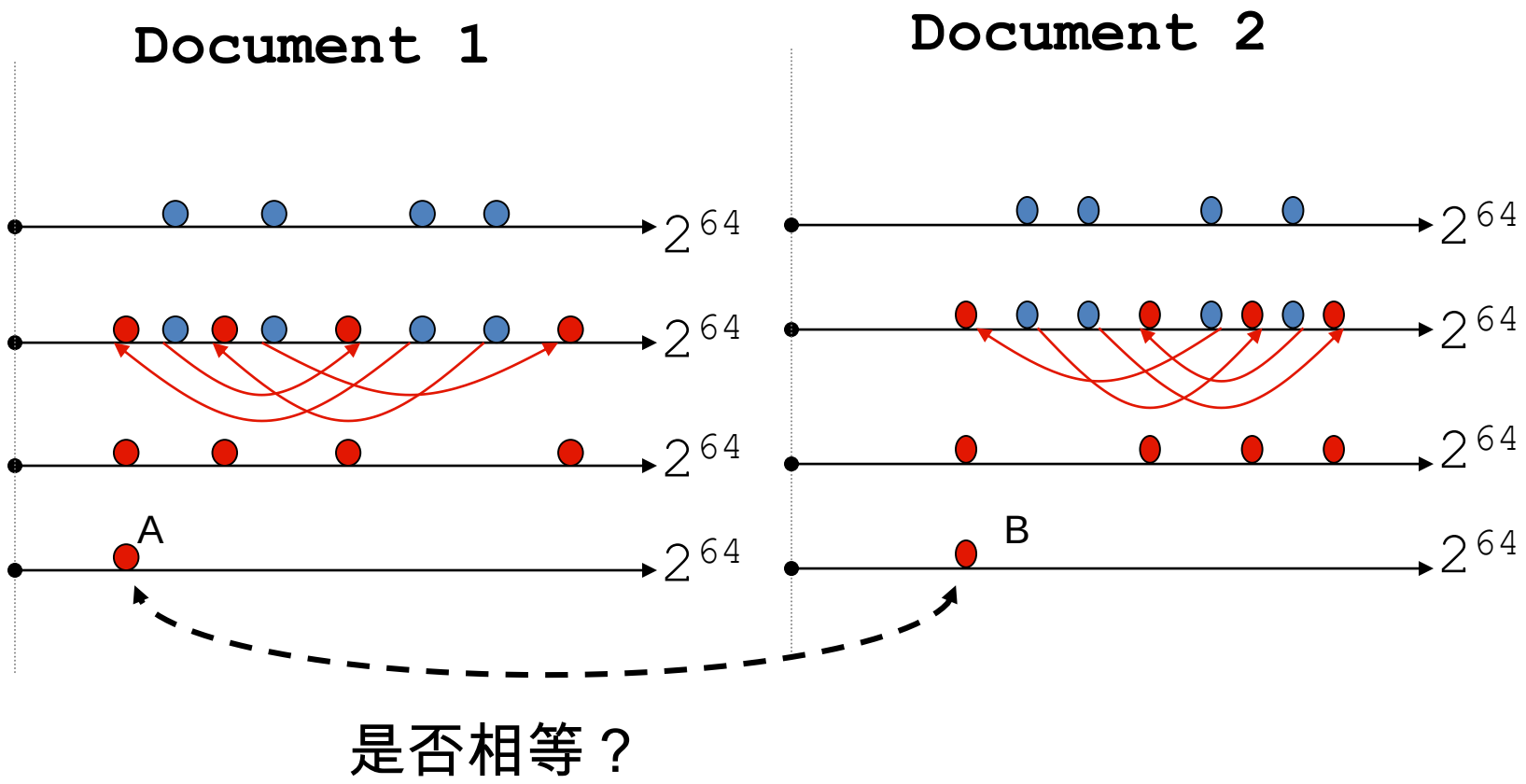
文档的素描（Sketch）

- **为每篇文档生成一个素描向量“sketch vector” (大小约为 ~200)**
  - 相同向量个数 ≥ t (一般 80%) 判定为近似near duplicates
  - 对文档D, $\text{sketch}_D[\ i\ ]$ 如下:
    - f 把所有的搭叠 shingles 映射到 {0..2^m }(e.g., f = fingerprinting 计算一个m位的摘要)
    - p_i 是 {0.. 2^m}的随机置换（集合对象随机排序）
    - 对D中所有singles s选择 MIN {p_i(f(s))}

计算 Sketch[i] for Doc1

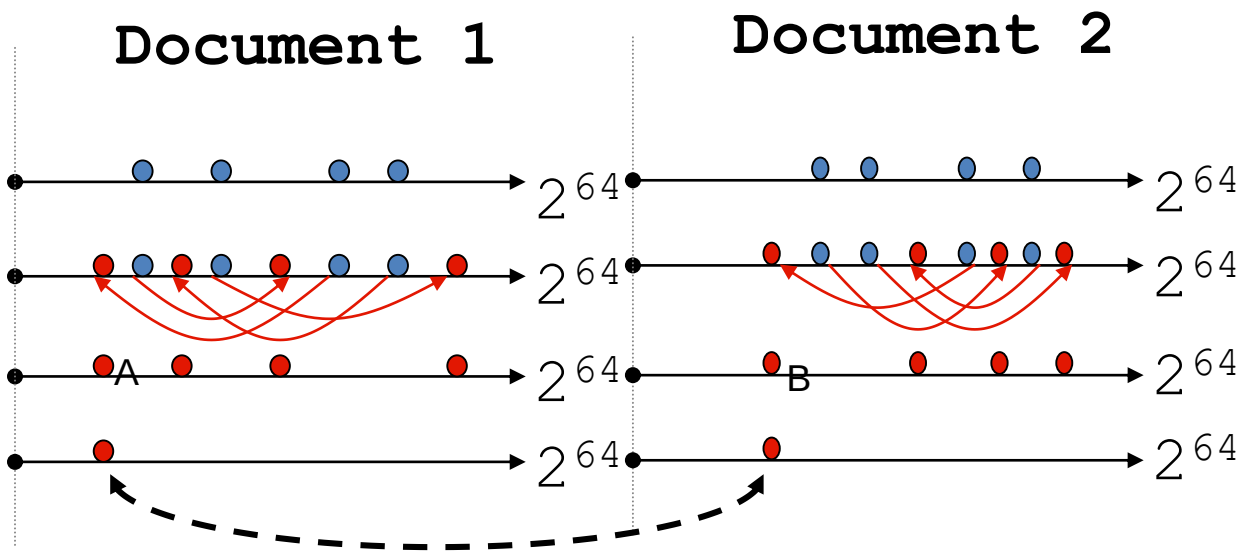


测试 if Doc1.Sketch[i] = Doc2.Sketch[i]



进行200次随机置换：p_1， p_2，… p_{200}

本质上是对shingle集合进行抽样



A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)

定理19-1: A = B发生的概率 = 交集大小/并集大小

（Size_of_intersection / Size_of_union）

小结：近似重复检测

- **Shingle算法的核心思想是将文件相似性问题转换为集合的相似性问题**
- **数量较大时，对shingle集合进行抽样，以降低空间和时间计算复杂性**
- **shingle取样主要有三种方法，即Min-Wise，Modm，和Mins**
  - Mins技术先将shingle和整数集进行映射，然后从中选择最小s个元素组成取样集合。此外，还可以使用shingle的hash值代表shingle进行相似性计算，能够节省一定计算开销。