

# 关联规则挖掘

## 基本概念

### 支持度的计算

$$\begin{aligned} \text{Support}(X \rightarrow Y) &= P(X \cup Y) \\ &= \frac{\text{项集}\{X \cup Y\}\text{的支持度计数}}{\text{事务表中总的事务数}} \\ &= \frac{\text{事务表中包含项集}\{X \cup Y\}\text{的事务数}}{\text{事务表中总的事务数}} \end{aligned}$$

### 置信度的计算

$$\begin{aligned} \text{Confidence}(X \rightarrow Y) &= P(Y | X) \\ &= \frac{P(X \cup Y)}{P(X)} \\ &= \frac{\text{项集}\{X \cup Y\}\text{的支持度计数}}{\text{项集}\{X\}\text{的支持度计数}} \\ &= \frac{\text{事务表中包含项集}\{X \cup Y\}\text{的事务数}}{\text{事务表中包含项集}\{X\}\text{的事务数}} \end{aligned}$$

# 支持度与置信度的计算一示例

Transacti on-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F

最小支持度: 50%  
最小置信度: 50%

频繁模式	Support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

对于规则  $A \rightarrow C$ :

$$\text{support} = \text{support}(\{A\} \cup \{C\}) = 50\%$$

$$\text{confidence} = \text{support}(\{A\} \cup \{C\}) / \text{support}(\{A\}) = 66.6\%$$

## Apriori算法

Apriori裁剪原理: 对于任意项集, 如果它不是频繁集, 则它的任何超集不用产生/测试!  
算法流程:

- While( $L_{k-1} \neq \phi$ ) (  $L_{k-1}$  表示  $k-1$  - 频繁项集 )
- {
  - 从  $L_{k-1}$  利用 **连接** 操作产生 **候选项集  $C_k$** ;
  - 对于  $C_k$  中的每一个元素  $c$ , 扫描数据库检查  $c$  是否是  $k$ -频繁项集;
  - $L_k = \{c \mid c \text{ 是 } k\text{-频繁项集}\}$
  - $k = k + 1$ ;
- 输出  $\cup_k L_k$

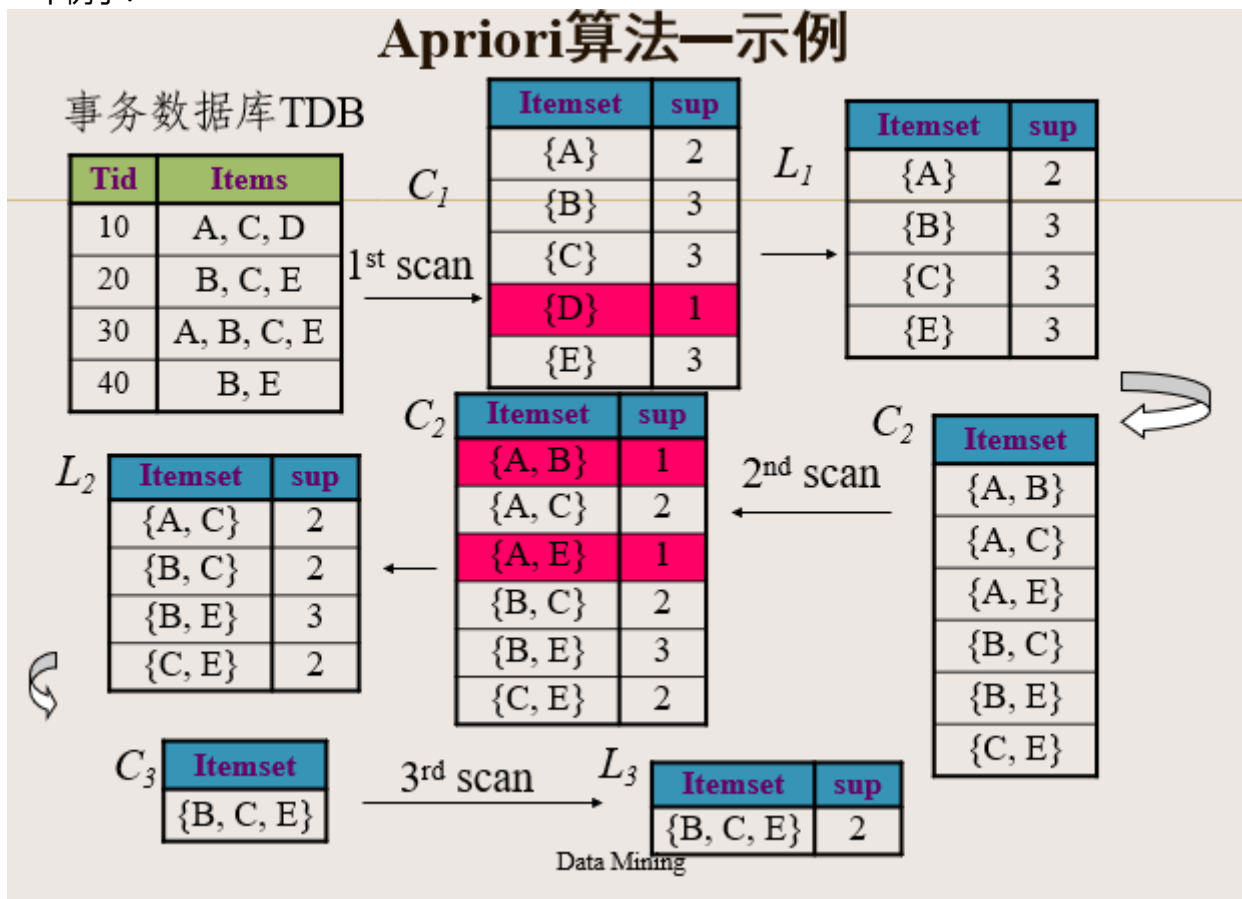
关于连接操作:

## 连接操作

- 对于两个 $k$ -项集 $\{I_1, I_2, \dots, I_{k-1}, I_k\}$ ,  $\{I_1', I_2', \dots, I_{k-1}', I_k'\}$ , 能够连接产生一个 $k+1$ -项集, 当且仅当 $I_i = I_i'$ , 其中 $i \in \{1, 2, \dots, k-1\}$ 且 $I_k \neq I_k'$ , 连接结果为( $k+1$ 项集):  

$$- \{I_1, I_2, \dots, I_{k-1}, I_k, I_k'\}$$
- 如 $\{A, C, D\}$ 与 $\{A, C, E\}$ 连接产生 $\{A, C, D, E\}$ ; 而 $\{A, C, D\}$ 与 $\{A, B, D\}$ 不能连接。

一个例子:



Apriori算法存在问题:

1. 多次扫描数据库
2. 产生大量的候选集合

## FP-Tree算法

<https://blog.csdn.net/kisslotus/article/details/80328045>

### FP-tree 算法的优点

1. FP-tree 算法只需对事务数据库进行二次扫描;
2. 避免产生大量候选集;

### FP-tree 算法的缺点

1. 要递归生成条件数据库和条件 FP-tree, 所以内存开销大;
2. 只能用于挖掘单维的布尔关联规则;

## 多维关联规则挖掘

多维关联规则: 规则中有两个以上的谓词。

例如:

$\text{Age}(X, \text{"30到40"}) \wedge \text{Income}(X, \text{"4万 - 6万"}) \rightarrow \text{Buys}(X, \text{"computer"})$