Recommend
火龙果软件

关联规则挖掘基本概念与算法

从推荐系统(recommender system)说起

Frequently Bought Together

Customers buy this book with Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms by Jean-Marc Adamo

Price For Both: $160.46

Add both to Cart

频繁项集

Customers Who Bought This Item Also Bought

Page 1 of 5

The Algorithm Design Manual by Steven S. Skiena

(28) $63.96

Pattern Recognition and Machine Learning (Infor... by Christopher M. Bishop

(41) $67.96

Data Mining: Practical Machine Learning Tools a... by Ian H. Witten

(26) $41.55

Elements of Information Theory 2nd Edition (Wil... by Thomas M. Cover

(24) $71.96

C Programming Language (2nd Edition) (Prentice... by Brian W. Kernighan

(257) $48.30

关联规则

关联规则挖掘的兴起

- 1993年，Agrawal提出了关联规则(Association Rule)问题，旨在发现顾客购货篮内商品间令人感兴趣的关系。
- “啤酒和尿布”
  沃尔玛利用NCR数据挖掘工具意外的发现：跟尿布一起购买最多的商品竟是啤酒！
- 今天，关联规则已广泛应用于金融、营销以及生物信息学等领域。

主要内容

- 关联规则的基本概念
- **Apriori**算法
- 其它基于**Apriori**的关联规则挖掘算法

关联规则挖掘的动机

- 发现数据内在的关系
  - 哪些商品往往被一起购买——啤酒尿布
  - 买了PC机之后，还会购买哪些商品
  - 哪些DNA对新药较为敏感

什么是关联规则

- 关联规则是寻找给定的数据集中项目之间令人感兴趣的关系

购物栏数据库

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

例子

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

蕴含并不是因果关系

频繁项集

- **项集**
  - 一个或多个项目的集合。
    例如: {Milk, Bread, Diaper}
  - 包含k 个项目的项集称为k-项集
- **绝对支持度 (σ)**
  - 某一项集出现的次数
    比如 σ({Milk, Bread,Diaper}) = 2
- **相对支持度**
  - 包含某一项集的事务在全体事务中的比例。比如.
    s({Milk, Bread, Diaper}) = 2/5
- **频繁项集**
  - 支持度不小于给定最小支持度阈值(*minsup*)的项集

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

关联规则

关联规则

形如 X → Y的蕴涵式, 其中 X 和 Y是项集，且X∩Y=∅。

比如: {Milk, Diaper} → {Beer}

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

规则评价参数

支持度 (s)

同时包含X和Y的事务占全部事务的百分比

可信度 (c)

包含项集X的事务中也包含Y的百分比

Example:

$$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$$

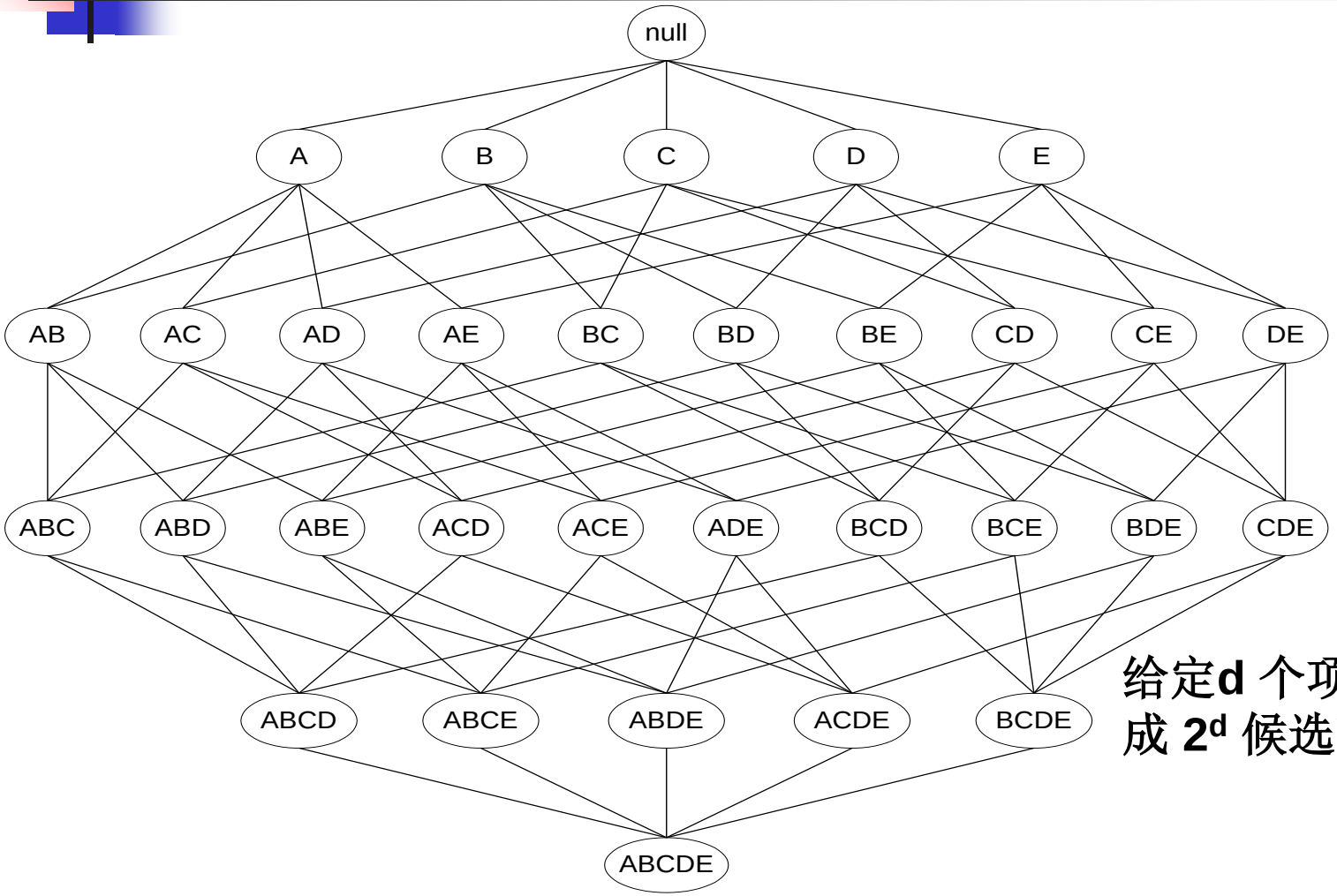
$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

关联规则挖掘的一般流程

- 找出满足最小支持度阈值的所有频繁项集。
- 由频繁项集产生满足最小可信度阈值的强关联规则。
- 这两步中，第二步较容易。关联规则挖掘的总体性能由第一步决定。

频繁项集的生成--1



给定d 个项目,可以生成 2^d 候选项集

频繁项集的生成--2

- 频繁项集格中每个项集都作为候选频繁项集
- 扫描数据库，计算每个候选集的支持度

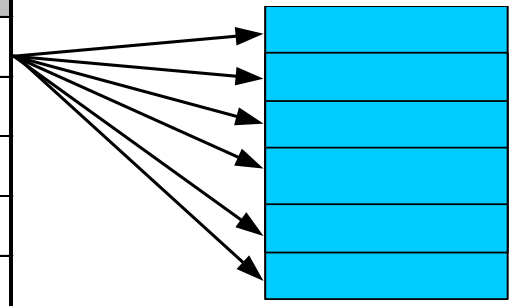
Transactions

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

N

w

List of Candidates

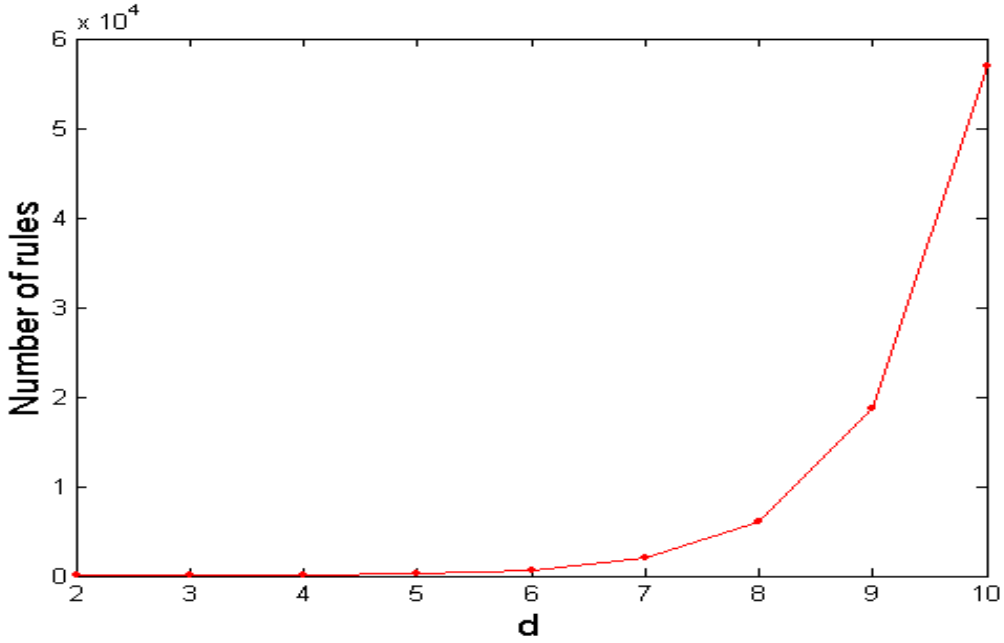


M

复杂度 ~ O(NMw) => Expensive since $M = 2^d$!!!

计算复杂度

- 假设存在 d 个不同的项目:
  - 项集总数= 2^d
  - 规则总数:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$

$$= 3^d - 2^{d+1} + 1$$

d=6, R = 602

频繁项集的生成策略

- 减少候选项集的个数 (M)
  - 利用各种剪枝方法减少M
- 减少事务的个数 (N)
  - 随着项集维度的增加，不断减少N的数目
- 减少比较的次数 (NM)
  - 使用新颖的数据结构存储事务/项集
  - 无需在每个事务中匹配每个项集

主要内容

- 关联规则的基本概念
- **Apriori算法**
- 其它基于Apriori的关联规则挖掘算法

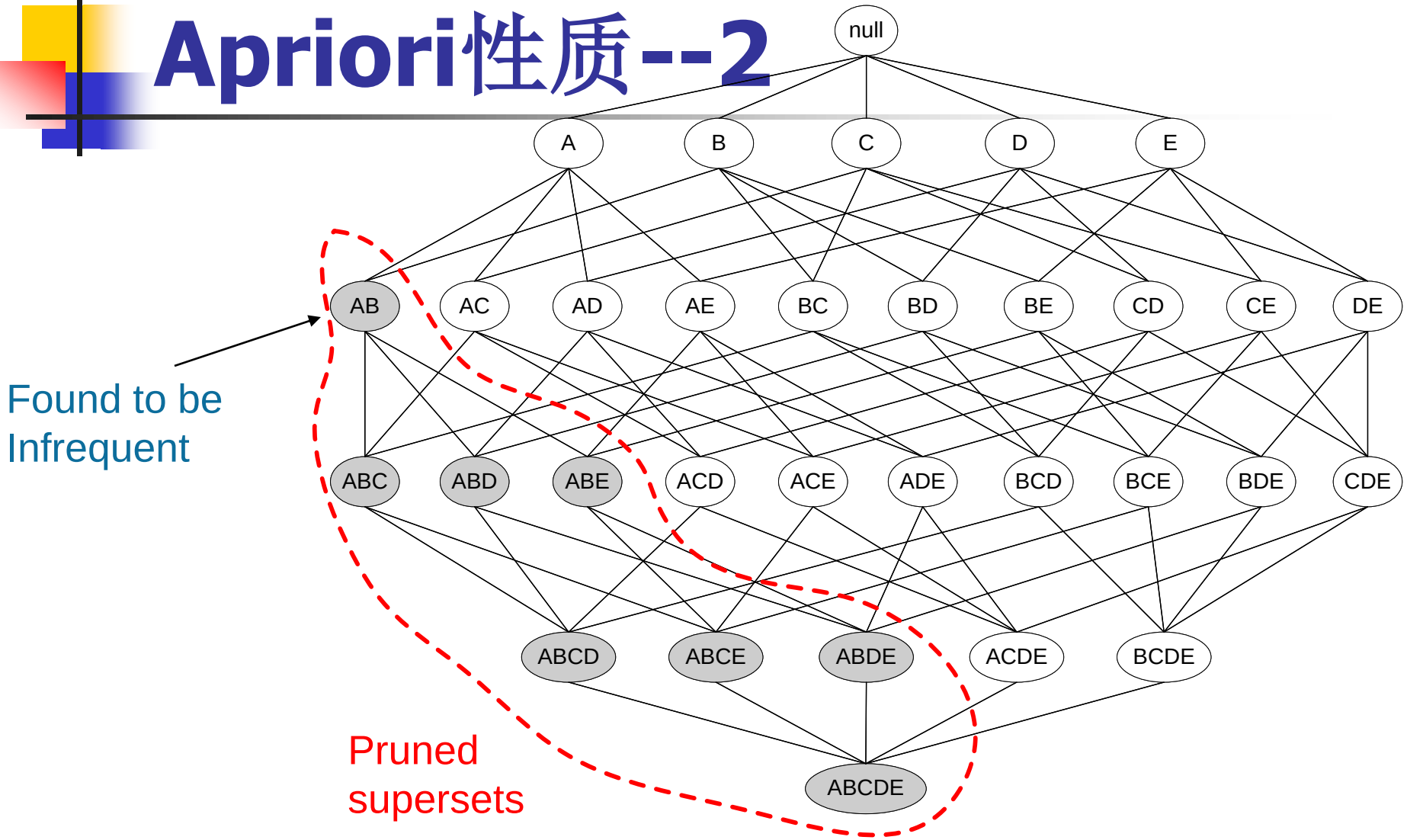
Apriori性质--1

- Agrawal R, Srikant R. Fast algorithms for mining association rules. (VLDB’94).

- Apriori 性质:
  - 频繁项集的所有非空子集都必须也是频繁的。

- Apriori 性质成立的原因:
  - 项集的支持度不超过其子集的支持度，即支持度的反单调性。

Apriori性质--2


null
A
B
C
D
E
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE
ABCD
ABCE
ABDE
ACDE
BCDE
ABCDE
Found to be Infrequent
Pruned supersets

Apriori算法--1

- 扫描数据库，找出1-频繁项集
- 连接k-频繁项集生成 (k+1)-候选项集

$$L_k \bowtie L_k = \{A \bowtie B \mid A, B \in L_k, |A \cap B| = k-1\}$$

- 在数据库中验证候选集是否频繁
- 当没有候选集或频繁项集生成时结束

Apriori算法--2

- Pseudo-code:

  C_k: Candidate itemset of size k
  L_k : frequent itemset of size k

  L_1 = {frequent items};
  for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
  increment the count of all candidates in C_{k+1}
  that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
  return $\cup_k L_k$;

Apriori算法示例

通过L_{k-1}与自己连接产生候选k-项集的集合，该候选k项集记为C_k
如果一个k-项集的(k-1)-子集不在L_{k-1}中，则该候选不可能是频繁的，可以直接从C_k删除。

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

$\xrightarrow{1^{st}\text{ scan}}$

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

$\longrightarrow$

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

$\xleftarrow{2^{nd}\text{ scan}}$

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

$\longrightarrow$

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

$\xrightarrow{3^{rd}\text{ scan}}$

L_3

Itemset	sup
{B, C, E}	2

规则生成--1

- 给定频繁项集L, 找出所有非空的$f \subset L$使得$f \rightarrow L - f$ 满足最小可信度阈值
  - 如 {A,B,C,D} 为频繁项集, 候选规则有:

    ABC →D, ABD →C, ACD →B, BCD →A,
    A →BCD, B →ACD, C →ABD, D →ABC
    AB →CD, AC → BD, AD → BC, BC →AD,
    BD →AC, CD →AB,

- 若$|L| = k$, 则存在$2^k - 2$个候选关联规则

规则生成--2

- 可信度一般不满足反单调性

  $c(ABC \rightarrow D)$ 可以比 $c(AB \rightarrow D)$大，也可以比$c(AB \rightarrow D)$小

- 定理. 若规则 $X \rightarrow Y\text{-}X$ 不满足最小可信度阈值，则规则 $X' \rightarrow Y\text{-}X'$，$X' \subset X$，也不满足最小可信度阈值。
- 比如, $L = \{A,B,C,D\}$:

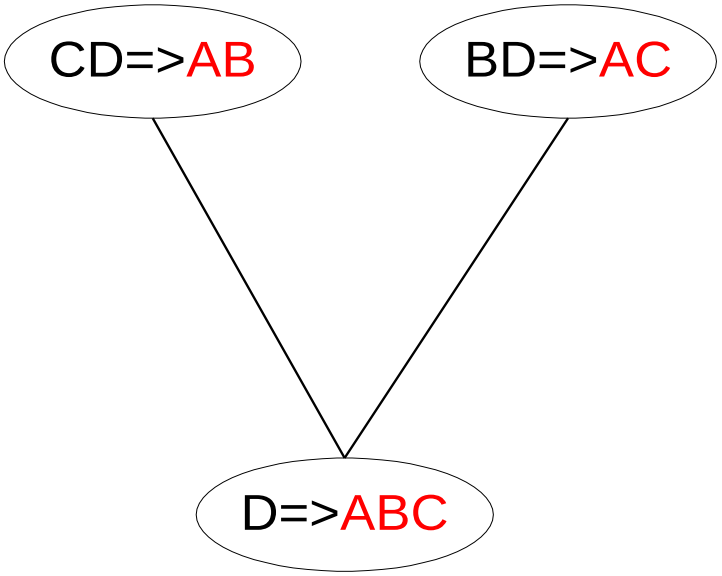
$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

规则生成--3：

- Apriori算法使用一种逐层方法来产生关联规则，其中每层对应规则后件中的项数。
- 算法首先提取规则后件只含一个项的所有高置信度规则，然后使用这些规则来产生新的候选规则。
- 例如使用$\{abc\}\rightarrow\{b\}$和$\{abd\}\rightarrow\{c\}$来产生候选规则$\{ad\}\rightarrow\{bc\}$。

规则生成--4

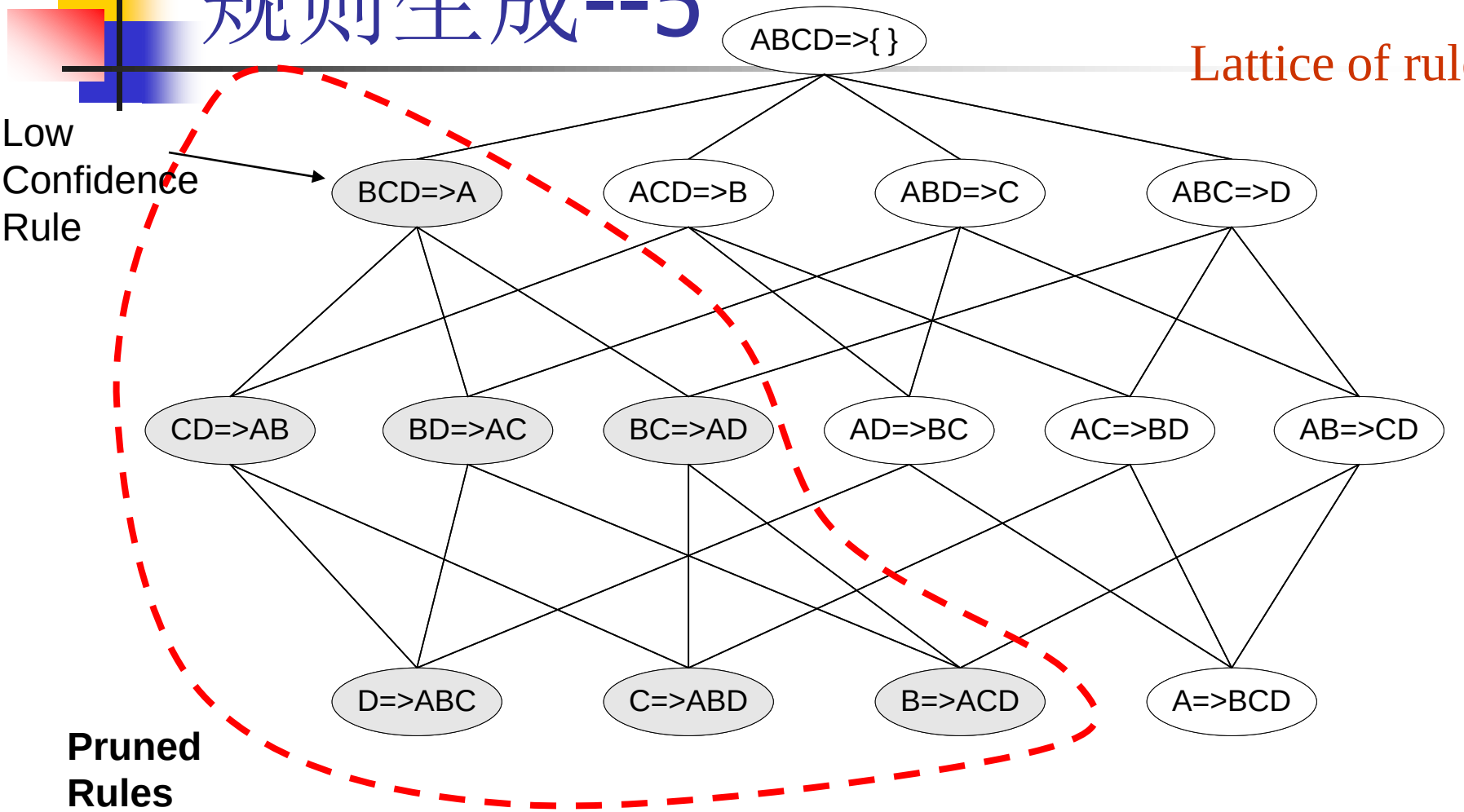
- 合并结论中具有共同前缀的规则，生成候选规则



- 连接(CD=>AB,BD=>AC)
  生成候选规则 D => ABC
- 若AD=>BC 的可信度未
  超过最小可信度阈值则删去 D=>ABC

规则生成--5

Lattice of rules



Recommend
火龙果软件

支持度—可信度框架的问题

- 强关联规则有可能不是令人感兴趣的

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Tea → Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

⇒ Although confidence is high, rule is misleading

⇒ $P(\text{Coffee}|\overline{\text{Tea}}) = 0.9375$

小结

- Apriori算法是挖掘频繁项集中最具有影响力的算法。算法有两步骤：一是发现所有的频繁项集；二是生成强关联规则。
- 发现频繁项集是关联规则挖掘中的关键步骤。在Apriori算法中利用“频繁项集的子集是频繁项集，非频繁项集的超集是非频繁项集”这一个性质有效的对频繁项集进行修剪。

小结

- **算法核心思想：**给定一个数据库，第一次扫描数据库，搜索出所有支持度大于等于最小支持度的项集组成频繁1-项集即为L1，由L1连接得到候选1-项集C1；第二次扫描数据库，搜索出C1中所有支持度大于等于最小支持度的项集组成频繁2-项集即为L2，由L2连接得到候选2-项集C2；同理第k次扫描数据库，搜索出Ck-1 中所有支持度大于等于最小支持度的项集组成频繁k-项集即为Lk，由Lk连接得到候选k-项集Ck，直到没有新的候选集产生为止。

小结

- Apriori算法需扫描数据库的次数等于最大频繁项集的项数。
- Apriori算法有两个致命的性能瓶颈：

  1.产生的候选集过大(尤其是2-项集)，算法必须耗费大量的时间处理候选项集

  2. 多次扫描数据库，需要很大的1/0负载，在时间、空间上都需要付出很大的代价。

频繁模式挖掘的挑战

- 挑战
  - 多次扫描事务数据库
  - 巨大数量的候选项集
  - 繁重的计算候选项集的支持度工作
- 改进 Apriori: 大体的思路
  - 减少事务数据库的扫描次数
  - 缩减候选项集的数量
  - 使候选项集的支持度计算更加方便

主要内容

- 关联规则的基本概念
- **Apriori**算法
- 其它基于**Apriori**的关联规则挖掘算法

AprioriTid算法核心

- Agrawal R, Srikant R. Fast algorithms for mining association rules. (VLDB’94).
- AprioriTid算法是在Apriori算法的基础上演化而来的。该算法第一趟扫描数据库时采用Apriori算法，当再次扫描时不再是扫描整个数据库，而只是扫描上次生成的候选项集，扫描的同时还会计算出频繁项集的支持度，以减少扫描数据库的时间来提高算法的效率。

AprioriTid

- AprioriTid算法由Apriori算法改进
- 优点：只和数据库做一次交互，无须频繁访问数据库
- 将Apirori中的C_k 扩展，内容由{c}变为{TID，c}，TID用于唯一标识事务，引入B_k ，使得B_k 对于事务的项目组织集合，而不是被动的等待C_k 来匹配

AprioriTid算法

- 举例：minsupp = 2
- 数据库：
- t

TID	项目
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

AprioriTid算法示例

TID	项目集
100	{1} {3} {4}
200	{2} {3} {5}
300	{1} {2} {3} {5}
400	{2} {5}

项集	支持度
{1}	2
{2}	3
{3}	3
{5}	3

ApioriTid算法示例

TID	项目集
100	{{1 3}}
200	{{2 3} {2 5} {3 5} }
300	{{1 2} {1 3} {1 5} {2 3} {2 5} {3 5}}
400	{{2 5}}

项集	支持度
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

ApioriTid算法示例

TID	项目集
100	空
200	{{2 3 5}}
300	{{2 3 5 }}
400	空

ApioriTid算法

- 上面图中分别为B_k 和L_k ， 而C_k 和Apriori算法产生的一样，因此没有写出来
- 可以看到B_k 由B_{k-1} 得到，无须由数据库取数据
- 缺点：内存要求很大，事务过多的时候资源难以满足

Recommend
火龙果软件

例子--1

数据库

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

$\overline{C_1}$

TID	Set-of-Itemsets
1	{{Bread}, {Milk}}
2	{{Bread}, {Diaper}, {Beer}, {Eggs}}
3	{{Milk}, {Diaper}, {Beer}, {Coke}}
4	{{Bread}, {Milk}, {Diaper}, {Beer}}
5	{{Bread}, {Milk}, {Diaper}, {Coke}}

L_1

Items	Count
Bread	4
Milk	4
Beer	3
Diaper	4

C_2

Itemset	Count
{Bread, Milk}	3
{Bread, Beer}	2
{Bread, Diaper}	3
{Milk, Beer}	2
{Milk, Diaper}	3
{Beer, Diaper}	3

$\overline{C_2}$

TID	Set-of-Itemsets
1	{{Bread, Milk}}
2	{{Bread, Diaper}, {Bread, Beer}, {Beer, Diaper}}
3	{{Milk, Diaper}, {Milk, Beer}, {Beer, Diaper}}
4	{{Bread, Milk}, {Bread, Diaper}, {Bread, Beer}, {Milk, Diaper}, {Milk, Beer}, {Beer, Diaper}}
5	{{Bread, Milk}, {Bread, Diaper}, {Milk, Diaper}}

例子--2

L_2

Itemset	Count
{Bread, Milk}	3
{Bread, Beer}	2
{Bread, Diaper}	3
{Milk, Beer}	2
{Milk, Diaper}	3
{Beer, Diaper}	3

C_3

Itemset	Count
{Bread, Milk, Diaper}	2

$\overline{C_3}$

TID	Set-of-Itemsets
4	{{Bread, Milk, Diaper}}
5	{{Bread, Milk, Diaper}}

Apriori vs. AprioriTid--1

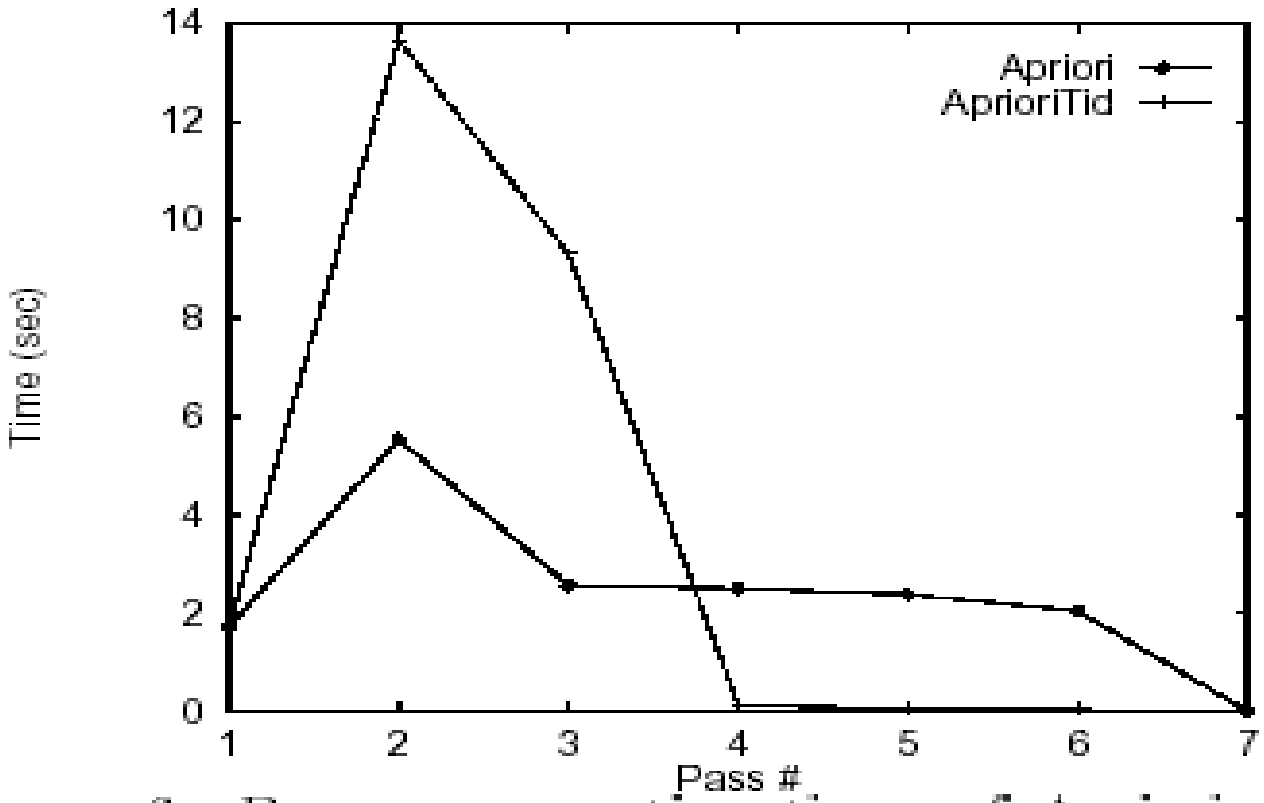


Figure 6: Per pass execution times of Apriori and AprioriTid (T10.I4.D100K, minsup = 0.75%)

Apriori vs. AprioriTid--2

- 最初几遍扫描数据库时，Apriori的性能优于AprioriTid；而从某次扫描数据库开始， AprioriTid 的性能优于Apriori

- 为什么？

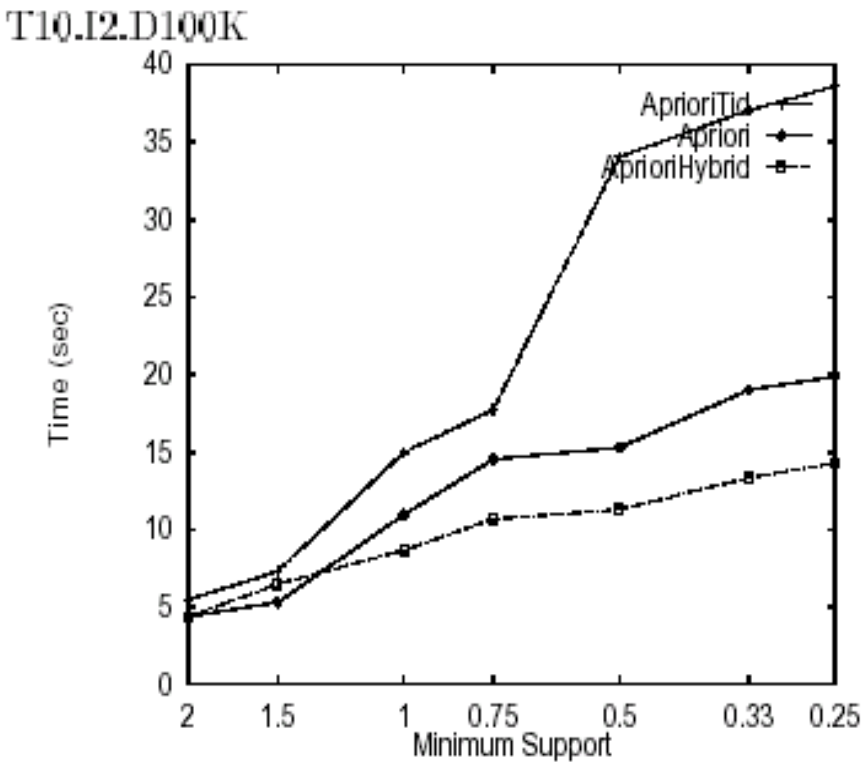
AprioriHybrid算法--1

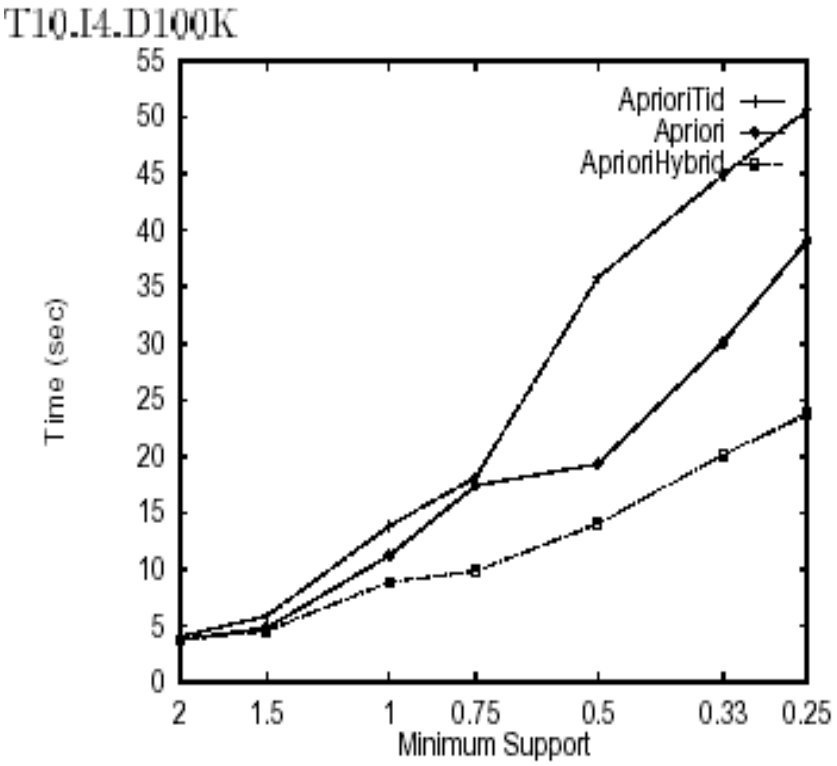
- Agrawal R, Srikant R. Fast algorithms for mining association rules. (VLDB’94).

- 开始使用Apriori算法
- 当 $\overline{C_k}$ 能够调入内存时，开始使用AprioriTid算法

Recommend
火龙果软件

AprioriHybrid算法--2





DHP算法--1

- Park J S, Chen M S, Yu P S. An Effective Hash-Based Algorithm for Mining Association Rules (SIGMOD’95)
- DHP是Directed Hashing and Pruning的缩写
- Apriori算法的执行代价由最初几次迭代决定
- 每次扫描数据库
  - 减少事务的数目
  - 减少与候选集的生成无关的项目

Recommend
火龙果软件

- DHP算法采用哈希(Hash)表技术对数据集和候选项集进行修剪来降低算法的时间和空间的开销。它利用哈希表在计算(k-1)-项集时先粗略计算出k-项集的支持度，排除无意义的候选k-项集来减少候选k-项集的数量，尤其是对候选2-项集的数量控制特别突出。

DHP算法--2

- 知识 1 ：任何候选项集先被Hash函数分配到某个桶中，且该桶的计数不低于最小支持度阈值
- 该知识用于减少候选项集的数目

Making a hash table

100 {A C}, {A D}, {C D}
200 {B C}, {B E}, {C E}
300 {A B}, {A C}, {A E}, {B C}, {B E}, {C E}
400 {B E}

h{{x y}) = ((order of x)*10 + (order of y)) mod 7;

{C E} {C E} {A D}	{A E}	{B C} {B C}		{B E} {B E} {B E}	{A B}	{A C} {C D} {A C}
3	1	2	0	3	1	3
0	1	2	3	4	5	6

Hash table H_2
Hash address

The number of items hashed to bucket 2

suppose the min support count is 3, then itemsets hashed to those buckets will not be considered in further processing

DHP算法--3

- 知识2：任何包含(k+1)-频繁集的事务至少要包含(k+1)个k-候选集
- 该知识用于减少事务的数目
- 知识3：参与(k+1)-频繁集生成的项目至少在k-候选集中出现k次
- 该知识用于裁减事务中不必要的项目
- 知识4：参与(k+1)-频繁集生成的项目至少在一个(k+1)-项集中出现，且该(k+1)-项集的全部k-子集均是k-候选集
- 该知识用于裁减事务中不必要的项目

例子--1

- 根据知识2裁减事务

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

{A C} → Discard

{B C} {B E} {C E} → Keep {B C E}

{A C} {B C} {B E} {C E} → Keep {B C E}

{B E} → Discard

D_3 = {<200, B C E>, <300, B C E>}

Recommend
火龙果软件

例子--2

- 对事务t=ABCDEF,若该事务包含5个2-候选集(AC, AE, AF, CD, EF)
- 根据知识3，B和D将被裁减掉，则t’=ACEF
- 根据知识4，C将被裁减掉，则t”=AEF

其它对Apriori的改进--1

- 事务约简

  若某个事务的长度小于k，则该事务将不用于k-频繁集的生成与检测。

- 数据的分割

  若某个项集可能为全局频繁项集，则它至少在数据库的一个分库中是频繁的。

Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases. VLDB'95.

其它对Apriori的改进--2

- 抽样

  在被抽样的数据库的子库中使用较小的支持度，并确保挖掘的完整性

Toivonen H. Sampling large databases for association rules. VLDB'96.

- 动态项集计数

  当某个项集的全部子集都被认定为频繁项集时，便把该项集加入候选频繁项集

Brin S, Motwani R, Ullman J D, Tsur S. Dynamic itemset counting and implication rules for market basket analysis. SIGMOD'97.

Recommend
火龙果软件

谢　谢

提高Apriori算法的有效性(1)

- Apriori算法主要的挑战
  - 要对数据进行多次扫描；
  - 会产生大量的候选项集；
  - 对候选项集的支持度计算非常繁琐；
- 解决思路
  - 减少对数据的扫描次数；
  - 缩小产生的候选项集；
  - 改进对候选项集的支持度计算方法
- 方法1：基于hash表的项集计数
  - 将每个项集通过相应的hash函数映射到hash表中的不同的桶中，这样可以通过将桶中的项集技术跟最小支持计数相比较先淘汰一部分项集。

提高Apriori算法的有效性(2)

- 方法2：事务压缩（压缩进一步迭代的事务数）
  - 不包含任何k-项集的事务不可能包含任何(k+1)-项集，这种事务在下一步的计算中可以加上标记或删除。
- 方法3：划分
  - 挖掘频繁项集只需要两次数据扫描
  - D中的任何频繁项集必须作为局部频繁项集至少出现在一个部分中。
    - 第一次扫描：将数据划分为多个部分并找到局部频繁项集
    - 第二次扫描：评估每个候选项集的实际支持度，以确定全局频繁项集

提高Apriori算法的有效性(3)

- 方法4：选样（在给定数据的一个子集挖掘）
  - 基本思想：选择原始数据的一个样本，在这个样本上用Apriori算法挖掘频繁模式
  - 通过牺牲精确度来减少算法开销，为了提高效率，样本大小应该以可以放在内存中为宜，可以适当降低最小支持度来减少遗漏的频繁模式
    - 可以通过一次全局扫描来验证从样本中发现的模式
    - 可以通过第二此全局扫描来找到遗漏的模式
- 方法5：动态项集计数
  - 在扫描的不同点添加候选项集，这样，如果一个候选项集已经满足最少支持度，则在可以直接将它添加到频繁项集，而不必在这次扫描的以后对比中继续计算。

关联规则价值衡量与发展

- 关联规则价值衡量
- 关联规则最新进展

规则价值衡量

- 对关联规则的评价与价值衡量涉及两个层面：
  - 系统客观的层面
  - 用户主观的层面

Recommend
火龙果软件

系统客观层面

- 使用“支持度和信任度”框架可能会产生一些不正确的规则。只凭支持度和信任度阈值未必总能找出符合实际的规则。

关联规则的兴趣度度量

- 客观度量
  - 两个流行的度量指标
    - 支持度
    - 置信度
- 主观度量
  - 最终，只有用户才能确定一个规则是否有趣的，而且这种判断是主观的，因不同的用户而异；通常认为一个规则（模式）是有趣的，如果：
    - 它是出人意料的
    - 可行动的（用户可以使用该规则做某些事情）
- 挖掘了关联规则后，哪些规则是用户感兴趣的？强关联规则是否就是有趣的？

对强关联规则的批评（1）

	打篮球	不打篮球	合计
喝麦片	2000	1750	3750
不喝麦片	1000	250	1250
合计	3000	2000	5000

- 例1：(Aggarwal & Yu, PODS98)
  - 在5000个学生中
    - 3000个打篮球
    - 3750个喝麦片粥
    - 2000个学生既打篮球又喝麦片粥
  - 然而，打篮球 => 喝麦片粥 [40%, 66.7%]是错误的，因为全部学生中喝麦片粥的比率是75%，比打篮球学生的66.7%要高
  - 打篮球 => 不喝麦片粥 [20%, 33.3%]这个规则远比上面那个要精确，尽管支持度和置信度都要低的多

对强关联规则的批评（2）

	买游戏	不买游戏	合计
买录像	4000	3500	7500
不买录像	2000	500	2500
合计	6000	4000	10000

- 例1：
  - 上述数据可以得出

buys(X, “computer games”) => buys(X, “videos”) [40%, 60%]

  - 但其实全部人中购买录像带的人数是75%，比60%多；事实上录像带和游戏是负相关的。
  - 由此可见*A* => *B*的置信度有欺骗性，它只是给出A,B条件概率的估计，而不度量A,B间蕴涵的实际强度。

用户主观层面

- 只有用户才能决定规则的有效性、可行性。所以，应该将用户的需求和系统更加紧密地结合起来。
- 可以采用基于约束（Consraint-based）的数据挖掘方法。具体约束的内容有：数据约束、 限定数据挖掘的维和层次、规则约束。
- 如果把某些约束条件与算法紧密结合，既能提高数据挖掘效率，又能明确数据挖掘的目标。

关联规则新进展

- 在基于一维布尔型关联规则的算法研究中先后出现了AIS、SETM等数据挖掘算法。
- R.Agrawal等人提出的Apriori 是经典算法。随后的关联规则发现算法大多数建立在Apriori算法基础上，或进行改造，或衍生变种。比如AprioriTid和AprioriHybrid算法。
- Lin等人提出解决规则挖掘算法中的数据倾斜问题，从而使算法具有较好的均衡性。Park等人提出把哈希表结构用于关联规则挖掘。

关联规则新进展（续）

- 数据挖掘工作是在海量数据库上进行的，数据库的规模对规则的挖掘时间有很大影响。
- Agrawal首先提出事务缩减技术，Han和Park等人也分别在减小数据规模上做了一些工作。
- 抽样的方法是由Toivonen提出的。
- Brin等人采用动态项集计数方法求解频繁项集。
- Aggarwal提出用图论和格的理论求解频繁项集的方法。Prutax算法就是用格遍历的办法求解频繁项集。

关联规则新进展（续）

- 关联规则模型有很多扩展，如顺序模型挖掘，在顺序时间段上进行挖掘等。
- 还有挖掘空间关联规则，挖掘周期性关联规则，挖掘负关联规则，挖掘交易内部关联规则等。
- Guralnik提出顺序时间段问题的形式描述语言，以便描述用户感兴趣的时间段，并且构建了有效的数据结构SP树（顺序模式树）和自底向上的数据挖掘算法。
- 最大模式挖掘是Bayardo等人提出来的。

关联规则新进展（续）

- 随后人们开始探讨频率接近项集。Pei给出了一种有效的数据挖掘算法。
- B.Özden等人的周期性关联规则是针对具有时间属性的事务数据库，发现在规律性的时间间隔中满足最小支持度和信任度的规则。
- 贝尔实验室的S.Ramaswamy等人进一步发展了周期性关联规则，提出挖掘符合日历的关联规则（Calendric Association Rules）算法，用以进行市场货篮分析。
- Fang等人给出冰山查询数据挖掘算法。

关联规则新进展（续）

- T.Hannu等人把负边界引入规则发现算法中，每次挖掘不仅保存频繁项集，而且同时保存负边界，达到下次挖掘时减少扫描次数的目的。
- Srikant等人通过研究关联规则的上下文，提出规则兴趣度尺度用以剔除冗余规则。
- Zakia还用项集聚类技术求解最大的近似潜在频繁项集，然后用格迁移思想生成每个聚类中的频繁项集。
- CAR，也叫分类关联规则，是Lin等人提出的一种新的分类方法，是分类技术与关联规则思想相结合的产物，并给出解决方案和算法。

关联规则新进展（续）

- Cheung等人提出关联规则的增量算法。
- Thomas等人把负边界的概念引入其中，进一步发展了增量算法。如，基于Apriori框架的并行和分布式数据挖掘算法。
- Oates等人将MSDD算法改造为分布式算法。还有其他的并行算法，如利用垂直数据库探求项集聚类等。

关联规则应用

- Web挖掘
- 文档分析
- 通信警告分析
- 网络入侵检测
- 生物信息学等

参考文献

- Agrawal R, Imielinski T, and Swami A. *Mining association rules between sets of items in large databases*. SIGMOD, 207-216, 1993.
- Agrawal R, and Srikant R. *Fast algorithms for mining association rules in large databases*. VLDB, 478-499, 1994.
- Han J W, Pei J, Yin Y W. *Mining frequent patterns without candidate generation*. SIGMOD, 1-12, 2000.
- Han J W, Pei J, Yin Y W, and Mao R Y. *Mining frequent patterns without candidate generation: a frequent-pattern tree approach*. Data Mining and Knowledge Discovery. 8, 53-87, 2004