基于粗糙集和单事务项组合的关联规则挖掘算法

王明芳[1] 蒋 芸[1] 王 勇[2] 明利特[1] 周 涛[1] 周泽寻[1]
(西北师范大学数学与信息科学学院 甘肃 730070)[1] (西北工业大学计算机学院 西安 710072)[2]

摘 要 Apriori算法必须反复地扫描数据库才能求出频繁项集,效率较低,且不支持更新挖掘。为了解决这些问题,提出了一种基于粗糙集、单事务项组合和集合运算的关联规则挖掘算法。本算法首先利用粗糙集进行属性约简,对新决策表中的每个事务进行"数据项"组合并标记地址,然后利用集合运算的方法计算支持度和置信度即可挖掘出有效规则。本算法只需要一次扫描数据库,同时有效地支持了关联规则的更新挖掘。应用实例和实验结果表明,本算法明显优于Apriori算法,是一种有效且快速的关联规则挖掘算法。

关键词 粗糙集,单事务项组合,集合运算,更新挖掘
中图法分类号 TP301.6 **文献标识码** A

Algorithm of Mining Association Rules Based on Rough Sets and Transaction Itemsets Combination

WANG Ming-fang[1] JIANG Yun[1] WANG Yong[2] MING Li-te[1] ZHOU Tao[1] ZHOU Ze-xun[1]
(College of Mathematics and Information Science, Northwest Normal University, Lanzhou 730070, China)[1]
(College of Computer Science, Northwest Polytechnical University, Xi'an 710072, China)[2]

Abstract The Apriori algorithm contains weaknesses such as often requiring a large number of repeated passes over the database to generate the frequent item sets and does not support the incremental updating. To solve these problems, a novel algorithm was proposed in this paper which is based on rough sets, single transaction combination itemsets and set operations for mining. It firstly uses the rough sets to reduce attributes, and then combines data item to each itemset from new decision table and marks it's tags. Finally, it calculates the support and confidence using set operations. This novel algorithm just needs to scanning the decision table only once, while effectively supporting the update of association rules mining. The results of application and experiments show that this novel algorithm is better than Apriori algorithm, it is an effective and fast algorithm for mining association rules.

Keywords Rough sets, Single transaction itemsets combination, Set operations, Updated mining

1 引言

数据挖掘(Data Mining)是数据库知识发现(KDD)的核心,是指从数据库中提取潜在的、有用的、最终可理解的知识的过程。关联规则挖掘则是数据挖掘的一个重要研究方向,其过程主要包含两个阶段:第一阶段必须先从数据集合中找出所有的频繁项目集,第二阶段再由这些频繁项目集产生关联规则。

Apriori算法[1]是最经典的关联规则挖掘算法,是由Rakesh Agrawal Rama和Krishnan Skrikant于1993年提出的。其核心是基于两阶段频繁集的递推算法。产生大量的候选集、需要重复扫描数据库和不支持更新挖掘是Apriori算法的重要缺陷。针对Apriori算法的固有缺点,学者们提出了许多改进算法:Improve算法[2]效率相比Apriori算法有所提高,但是需要多次扫描数据库;FP-Growth[3]算法可以不产生候选频繁项集,但是不支持更新挖掘;基于线性链表的挖掘方法[4]虽然支持更新挖掘,但在求置信度时也需要多次扫描数据库;动态树结构的CP-tree算法[5]支持更新挖掘,但是时间复杂度较高。以上算法还有一个共同的弱点,就是当数据量很大时,因为没有约简冗余的数据导致挖掘效率不高。

针对以上问题,本文提出了一种基于粗糙集、单事务项组合和集合运算的关联规则挖掘算法,是一种简单、实用、有效的挖掘算法。该算法首先利用粗糙集消除不重要的属性,从而减少了数据挖掘的规模,对大的数据库非常有效,可以快速提高挖掘效率;然后利用单事务项组合和集合运算的方法求出关联规则。本算法只需要一次扫描数据库,且由于本算法保留了备用候选项集,因此很好地支持了数据库的更新挖掘。文中给出了相关算法,并通过实例分析和实验结果分析验证

到稿日期:2010-12-06 返修日期:2011-05-16 本文受国家自然科学(60873196),甘肃省科技计划(甘肃省自然科学基金项目1010RJZA022),西北师范大学2010年第三期知识与创新工程科研骨干项目(nwnu-kjcxgc-03-67),西北师范大学2006-2010年度重点学科"网络计算"资助。

王明芳(1986-),女,硕士生,主要研究方向为数据挖掘、粗糙集,E-mail:changdaodi@126.com;蒋 芸(1970-),女,博士,副教授,主要研究方向为数据挖掘、粗糙集;王 勇(1973-),男,博士,主要研究方向为数据挖掘;明利特(1982-),男,硕士生,主要研究方向为数据挖掘、粗糙集;周 涛(1976-),男,硕士生,主要研究方向为数据挖掘、粗糙集;周泽寻(1986-),男,硕士生,主要研究方向为数据挖掘、粗糙集。

了本文算法的优越性。

2 粗糙集及基本理论[6,7]

1982年，波兰华沙理工大学Pawlak教授提出了一种处理不精确和模糊知识的数学工具——Rough集理论[9]。属性约简是粗糙集理论中的核心内容之一，即在保持信息系统分类能力不变的条件下，剔除冗余属性，减少属性数目，提高分析效率。属性约简定义为不含多余属性并保证分类正确的最小条件属性集。

下面介绍粗糙集理论中的几个主要概念。

定义1(决策系统) 四元组$S=(U,A,V,f)$是一个决策系统。其中U称为论域，表示对象的非空有限集合，$U=\{x_1,x_2,\cdots,x_m\}$；A是属性集合，$A=C\cup D$，其中C是条件属性集，D是决策属性集；$V=\bigcup V_a(a\in A)$是属性值的集合，V_a是属性a的值域；$f:U\times A\to V$是一个信息函数，它为每个对象的每个属性赋予一个信息值，即$\forall a\in A, x\in U, f(x,a)\in V_a$。当属性集合$A$不划分为条件属性子集合和决策属性子集合时，该系统又称为信息系统。

定义2(不可分辨关系) 不可分辨关系是粗糙集理论的基础概念，它反映了观察世界的不确定性。它在信息系统中定义为对于任一属性子集$B\subseteq R$，如果对象$x_i,x_j\in U$，$\forall r\in B$，，当且仅当$f(x_i,r)=f(x_j,r)$时，x_i和x_j是不可分辨的，简记为$Ind(B)$。

定义3(等价类的表示) 对于$x\in U$，可用集合$[x]_p=\{y\in U|(x,y)\in ind(P)\}$来表示包含元素$x$的等价类。

定义4(属性依赖度) 设$S=(U,A,V,f)$为一个信息系统，属性依赖度表示为$r(B,D)=\frac{|POS_B(D)|}{|POS_C(D)|}$，其中$POS_B(D)$为根据属性集合$B$划分的正区域，而$POS_C(D)$为根据整个条件属性集$C$划分的正区域。

定义5(约简与核) 对于决策系统$S=(U,A,V,f)$，$B\subseteq A$且$a\in B$：

(1) 如果$I_B=I_{B-\{a\}}$，则称属性a在B中是冗余的，否则在B中是必要的；

(2) 如果B的所有属性都是必要的，则集合B是独立的，否则是相依的；

(3) 设$B'\subseteq B$，如果B'是独立的，且$I_B=I_{B-\{a\}}$，则B'是B的一个约简。

A中所有必要的属性组成的集合称为属性集A的核，记作$Core(A)$。可以证明核是所有约简的交集。

性质1 由定义2和定义3可得如下结论：假设$X\subset R$，$Y\subset R, a\in V_x, a\in V_y$，则有$[XY]=[X]_a\cap[Y]_a$。

推论1 由性质1推导可得如下结论：若包含X和包含Y的地址已知，则同时包含X和Y的地址为两者地址的交集。

3 关联规则及基本概念[1,8]

关联规则中常用的Apriori算法由Agrawal[1]等人于1993年提出，此算法是研究关联规则中最具代表性的方法。

定义6(项和项集) 令$AI=\{I_1,I_2,\cdots,I_m\}$是所有项的集合，其中$I_k(k=1,2,\cdots,m)$称为项。项的集合称为项集$I$。包含$k$个项的项集称为$k$项集，$k$表示项长，即$k=|I|$。

定义7(频繁项集) 给定一个支持度阈值missup，$1\leqslant$ missup$\leqslant|TDB|$，则支持度大于等于missup的项集称为频繁项集，简称频繁集，也称大项集。

定义8(支持度) 关联规则$X\Rightarrow Y$的支持度指事务集$|TDB|$中同时支持项集X和Y的事务所占的百分比。支持度说明了该条规则在所有事务中具有多大的代表性，它是对关联规则重要性的衡量。显然，支持度越大，规则越重要，其数学形式为

$$\text{support}(X\Rightarrow Y)=P(X\cup Y)=\frac{|\{T|T\in TDB\wedge(X\cup Y)\subseteq T\}|}{|TDB|} \quad (1)$$

定义9(置信度) 规则$X\Rightarrow Y$的置信度指的是事务集$|TDB|$中支持项集X的事务里同时也支持Y的事务所占的百分比。简而言之，置信度就是指在出现了X的事务中Y也同时出现的概率。置信度是对关联规则准确度的衡量，其数学形式为

$$\text{confidence}(X\Rightarrow Y)=P(X\cup Y)=\frac{|\{T|T\in TDB\wedge(X\cup Y)\subseteq T\}|}{|\{T|T\in TDB\wedge X\subseteq T\}|} \quad (2)$$

同时满足支持度和置信度阈值的项集称为规则。

4 本文算法基本思想

本文算法的基本思想是，首先利用粗糙集的基于区分矩阵条件属性约简算法对决策表里的条件属性进行求核和约简，得到最小约简集，把不必要的属性约简掉，得到最小的决策表。接着对新的决策表的条件属性和决策属性分别进行扫描、组合、标记和计算4个步骤，得到条件属性频繁项集L_C和决策属性频繁项集L_D。然后连接L_D和L_C，计算$L=L_C\times L_D$，L为同时具有条件属性和决策属性的频繁项集。再利用之前标记地址的结果通过集合运算的方法计算L的置信度，避免再次扫描决策表，满足置信度的项就是所求的规则，放入规则集RS中。最后对RS中的规则进行合并优化，即为所挖掘的最终关联规则。在计算频繁项集时，保留备用候选项集，为以后的更新挖掘做好准备。

4.1 本文算法的实现步骤描述

Step1 数据概化，形成决策表，并确定条件属性和决策属性。

Step2 利用区分矩阵条件属性约简算法对决策表中的条件属性进行求核和约简，得到最小约简集，形成新的决策表。

Step3 扫描新的决策表，对表中每个事务的条件属性进行“数据项”组合，统计L_C中对所有相同的“项集”，并标记包含该“项集”的所有地址，然后通过式(1)和推论1计算支持度，得到条件属性频繁集和条件属性备用候选项集。

Step4 同理，对新决策表中的每个事务的决策属性进行扫描、组合、标记和计算，得到决策属性频繁集合决策属性备用候选项集。

Step5 通过连接L_C和L_D得到同时具有条件属性和决策属性的频繁项集，利用式(2)和推论1计算置信度，并产生规则集RS。

Step6 合并优化RS中的规则。

假设有一决策表(见表1)，A和B为条件属性，C为决策

属性。用本文算法对表 1 进行扫描、组合、标记和计算步骤的实例描述，如表 1 所列。

表 1　决策表

Tid	A	B	C
T_0	0	0	0
T_1	0	1	2
T_2	0	1	2
T_3	1	1	0
T_4	1	0	2
T_5	0	1	0
T_6	1	0	1
T_7	0	0	1
T_8	1	1	2
T_9	0	1	1

扫描决策表，组合事务项 T_0 中的条件属性并标记地址。操作后的结果如表 2 所列。

表 2　T_0 中组合项和地址

项集	地址
A=0	{0}
B=0	{0}
A=0 B=0	{0}

对 T_1 中的条件属性进行组合。如果组合的项集在表 2 中出现过，就直接把 T_1 的地址加入该项集的后面；如果没有出过该项集，则把该项集添加到表 2 中并标记它的地址。操作后的结果如表 3 所列。

表 3　T_1 中组合项集和地址

项集	地址
A=0	{0,1}
B=0	{0}
A=0 B=0	{0}
B=1	{1}
A=0 B=1	{1}

依次类推，扫描决策表 T 中的其余事务项的条件属性得到 L_C。结果如表 4 所列。

表 4　L_C

项集	地址
A=0	{0,1,2,5,7,9}
B=0	{0,4,6,7}
A=0 B=0	{0,7}
B=1	{1,2,3,5,8,9}
A=0 B=1	{1,2,5,9}
A=1	{3,4,6,8}
A=1 B=1	{3,8}
A=1 B=0	{4,6}

说明：通过对事务数据库 T 的一次扫描，获得了 T 中的所有项集和包含该项集的所有地址。

然后进行判断，把不满足支持度的项集从 L_C 中取出，放入备用候选项目集 L_C'中(集合 L_C'用于数据库的更新挖掘)，得到的新的 L_C 就是满足支持度的条件属性频繁项集。若 minsup=20%，则得到 L_C={A=0{0,1,2,5,7,9},B=0{0,4,6,7},B=1{1,2,3,5,8,9},A=0 B=1 {1,2,5,9},A=1 {3,4,6,8}}，条件属性备用候选项集为 L_C'={A=0B=0{0,7},A=1 B=1{3,8},A=1 B=0{4,6}}。

同理通过扫描、组合、标记和计算的步骤求出满足支持度的决策属性频繁项集 L_D={C=0 {0,3,5},C=1 {6,7,9},C=2 {1,2,4,8}}，决策属性备用候选项集 $L_D'=\emptyset$。

4.2　算法伪代码描述

T为决策表，C为条件属性，D为决策属性，L_C 为条件属性频繁项集，L_D 为决策属性频繁项集，L_C'为条件属性备用候选项集，L_D'为决策属性备用候选项集。

```
1)将 L_C、L_D、L_C'、L_D'置空；
2)打开决策表 T；
3)For T 中的每一个事务记录 T_i
4){ 求出 T_i 的所有条件属性的组合 C_j；
5)  If(C_j∈L_C)
6)    把 C_j 的地址添加到 L_C 中 C_j 的地址标记中；
7)  Else
8)    {
9)    L_C=L_C∪C_j；//把 C_j 添加到 L_C 中.
10)    把 C_j 的地址添加到 L_C 中 C_j 的地址标记中；
11)     }
12) }
13) For 每个组合 C_j∈L_C
14) {
15)   If ( | C_j | / | T | )≤minsup
16)     {L_C'=L_C'∪C_j；//把 C_j 添加到 L_C'中.
17)     L_C=L_C-C_j；
18)       }
19)   }//现在 L_C 中放的是所有条件属性频繁项集.
20) For T中的每一个事务记录 T_i
21){ 求出 T_i 的所有决策属性的组合 D_i；
22)   If(D_i∈L_D)
23)     把 D_i 的地址添加到 L_D 中 D_i 的地址标记中；
24)   Else
25)     {
26)   L_D=L_D∪D_i；//把 D_i 添加到 L_D 中.
27)   把 D_i 的地址添加到 L_D 中 D_i 的地址标记中；
28)     }
29)   }
30)   For 每个组合 D_i∈L_D
31)     {
32)       If ( | D_i | / | T | )≤minsup
33)         {L_D'=L_D'∪D_i；//把 D_i 添加到 L_D'中.
34)         L_D=L_D-D_i；
35)           }
36)       } //现在 L_D 中放的是所有决策属性频繁项集.
37) For(i=1;i< | L_C | ;i++)//遍历 L_C 中的所有的元素.
38) For(j=1;j< | L_D | ;j++)//遍历 L_D 中的所有的元素.
39){
40) L=L_C×L_D;// L 为同时具有条件属性和决策属性的频繁项集.
41)   If ((L)/(L_C)≥minconf)
42)     把 L 放到规则集 RS 中；
43)   }
44) 对规则集 RS 中的规则进行合并优化.
```

4.3　更新挖掘

当发生以下 3 种情况时需要进行更新挖掘：一是最小支持度发生改变时，二是当决策表中新增事务项时，三是当决策表中删除事务项时。然而传统的 Apriori 算法无法支持更新挖掘，它必须重新扫描整个决策表，才能得到更新后的频繁项集，这对海量的数据来说开销是巨大的。本算法可以利用对各项集标记的地址进行交运算的方法更好地支持更新挖掘。

当最小支持度变大时，只需要扫描一遍 L_C 和 L_D，把不满足条件的项集放入备用候选项集中即可；当最小支持度变小

时，只需要扫描一遍 L_C' 和 L_D'，把满足条件的项集分别放入 L_C 和 L_D 中即可；当新增事务项时，只需要扫描一遍新增的事务项，然后对它进行组合、标记和计算即可；当删除事务项时，也只扫描一遍要删除的事务项，然后对它进行组合，并在 L_C、L_D、L_C' 和 L_D' 中对它的组合进行删除地址即可。

4.4 效率比较与分析

设 m 表示事物数据库的规模，n 表示属性的个数，l 表示约简后的属性个数，C_k 表示候选频繁 k 项集，L_k 表示频繁 k 项集，则 Apriori 算法第一遍数据库扫描时间为 $O(mn)$，在每一步中连接时间开销为 $O(|L_{k-1}|\times|L_{k-1}|)$，剪枝时间是 $O(|C_k|)$，扫描计数时间为 $O(m\times|C_k|)$，所以 Apriori 算法总的时间开销为 $O(mn)+\sum_{k\geqslant 2}(O(|L_{k-1}|\times|L_{k-1}|)+O(|C_k|)+O(m\times|C_k|))$。

本文方法中第一步使用区分矩阵约简冗余属性的时间为 $O(nm^2)$，然后使用本文算法挖掘规则的时间复杂度为 $O(m\times 2^l)$。由此可见本文算法挖掘规则的时间开销远远小于 Apriori 算法。但是通过分析也发现，使用区分矩阵属性约简算法预处理数据的时间开销不是很小，所以在处理不同大小的数据集时最好选择合适的属性约简算法。由于本文实验中选取的是 UCI 中的 5 个数据集，因此选用了区分矩阵属性约简算法进行数据预处理。

5 实例分析

下面用表 5 所列的经典天气决策表为例挖掘其中的关联规则，来说明本文提出的算法的有效性和可行性。

表 5 经典天气决策表

U	天气	温度	湿度	风	决策
0	晴	热	高	否	N
1	晴	热	高	真	N
2	多云	热	高	真	N
3	雨	温暖	高	否	P
4	雨	冷	正常	否	P
5	雨	冷	正常	真	N
6	多云	冷	正常	真	P
7	晴	温暖	高	否	N
8	晴	冷	正常	否	P
9	雨	温暖	正常	否	P

为方便使用区分矩阵对该决策表进行约简和求核，对表 5 进行概化得到表 6，其中 A 代表天气、B 代表温度、C 代表湿度、D 代表风、E 代表决策，0、1、2 分别为各个属性在上表中对应位置的属性值。

表 6 天气决策表的概化

U	A	B	C	D	E
0	0	0	0	0	0
1	0	0	0	1	0
2	1	0	0	1	0
3	2	2	0	0	1
4	2	1	1	0	1
5	2	1	1	1	0
6	1	1	1	1	1
7	0	2	0	0	0
8	0	1	1	0	1
9	2	2	1	0	1

确定 A、B、C、D 为条件属性，E 为决策属性。根据区分矩阵对表 6 进行约简和求核，分辨函数为 $F=(A\wedge C\wedge D)\vee(A\wedge B\wedge D)$，即约简集为 $\{A,C,D\}$ 和 $\{A,B,D\}$，核为 $\{A,D\}$。选取 $\{A,B,D\}$ 为最小约简集，删除多余的属性，形成新的决策表为表 7。

表 7 新的决策表

U	A	B	D	E
0	0	0	0	0
1	0	0	1	0
2	1	0	1	0
3	2	2	0	1
4	2	1	0	1
5	2	1	1	0
6	1	1	1	1
7	0	2	0	0
8	0	1	0	1
9	2	2	0	1

对表 7 进行扫描、组合和标记地址，得到 L_C（如表 8 所列）。

表 8 L_C

A=0	{0,1,7,8}	A=2 B=2	{3,9}
B=0	{0,1,2}	A=2 D=0	{3,4,9}
D=0	{0,3,4,7,8,9}	B=2 D=0	{3,7,9}
A=0 B=0	{0,1}	A=2 B=2 D=0	{3,9}
A=0 D=0	{0,7,8}	B=1	{4,5,6,8}
B=0 D=0	{0}	A=2 B=1	{4,5}
A=0 B=0 D=0	{0}	B=1 D=0	{4,8}
D=1	{1,2,5,6}	A=2 B=1 D=0	{4}
A=0 D=1	{1}	A=2 D=1	{5}
B=0 D=1	{1,2}	B=1 D=1	{5,6}
A=0 B=0 D=1	{1}	A=2 B=1 D=1	{5}
A=1	{2,6}	A=1 B=1	{6}
A=1 B=0	{2}	A=1 B=1 D=1	{6}
A=1 D=1	{2,6}	A=0 B=2	{7}
A=1 B=0 D=1	{2}	A=0 B=2 D=0	{7}
A=2	{3,4,5,9}	A=0 B=1	{8}
B=2	{3,7,9}	A=0 B=1 D=0	{8}

假设 missup=20%，通过式(1)和推论 1 计算表 8 中各数据项的支持度后得到 L_C 和 L_C'（如表 9 和表 10 所列）。

表 9 L_C

A=0	{0,1,7,8}	B=2	{3,7,9}
B=0	{0,1,2}	A=2 B=2	{3,9}
D=0	{0,3,4,7,8,9}	A=2 D=0	{3,4,9}
A=0 B=0	{0,1}	B=2 D=0	{3,7,9}
A=0 D=0	{0,7,8}	A=2 B=2 D=0	{3,9}
D=1	{1,2,5,6}	B=1	{4,5,6,8}
B=0 D=1	{1,2}	A=2 B=1	{4,5}
A=1	{2,6}	B=1 D=0	{4,8}
A=1 D=1	{2,6}	B=1 D=1	{5,6}
A=2	{3,4,5,9}		

表 10 L_C'

B=0 D=0	{0}	A=1 B=1	{6}
A=0 B=0 D=0	{0}	A=0 B=2	{7}
A=0 D=1	{1}	A=2 B=1 D=1	{5}
A=0 B=0 D=1	{1}	A=1B=1 D=1	{6}
A=1 B=0	{2}	A=0 B=2 D=0	{7}
A=1 B=0 D=1	{2}	A=0 B=1	{8}
A=2 B=1 D=0	{4}	A=0 B=1 D=0	{8}
A=2 D=1	{5}		

同理得到 L_D：$E=0$ {0,1,2,5,7}，$E=1$ {3,4,6,8,9}。

L_C 与 L_D 进行连接，假设 misconf=60%，则根据式(2)和推论 1 计算连接后各数据项的置信度。满足最小置信度的放入规则集中，得到规则集 RS。

$RS=\{\{B=2,D=0,E=1\},\{B=0,D=1,E=0\},\{B=0,E=0\},\{A=2,B=2,D=0,E=1\}\{A=2,B=2,E=1\},\{A=0,B=0,E=0\},\{B=1,E=1\},\{B=2,$

$E=1\}$,$\{B=1,D=0,E=1\}$,$\{A=2,E=1\}$,$\{A=2,D=0,E=1\}$,$\{A=0,E=0\}$,$\{D=0,E=1\}$,$\{D=1,E=0\}$,$\{A=0,D=0,E=0\}\}$

合并优化 RS 中的规则，得到 $RS=\{B=0\rightarrow E=0, B=1\rightarrow E=1, B=2\rightarrow E=1, A=2\rightarrow E=1, A=0\rightarrow E=0, D=0\rightarrow E=1, D=1\rightarrow E=0\}$

6 实验与结果分析

用本算法与 Apriori 算法在关联规则挖掘执行时间上和效率上进行比较，结果表明本算法明显优于 Apriori 算法。实验机器配置：CPU 为 Intel Pentium4 3.06GHz，内存为 DDR 512MB，操作系统为 Windows XP SP3，开发语言采用 JAVA。测试数据集来自 5 个标准 UCI 数据集[9]，Tic-tac、Breast、Monk-1、Zoo 和 mushroom。试验中首先采用属性约简算法对数据集进行属性约简，结果如表 11 所列。

表 11 实验数据集和属性约简结果

Database	总记录数	原属性数	约简后属性数
Tic-tac	985	10	9
Breast	699	10	5
Monk-1	432	7	4
Zoo	101	17	6

然后再对以上数据集分别执行 Apriori 算法和本算法，比较挖掘规则数和时间效率。理论分析是当本算法中的最小置信度设置越小，挖掘的结果就会越接近于 Apriori 算法，所以设置 minsup=30%，minconf=0%。由于在挖掘时本算法是在约简后的数据集上运行的，而 Apriori 算法是在原数据集上运行的，条件属性个数不相同，因此挖掘出来的规则会稍有差别。本算挖掘出来的规则可能和 Apriori 算法挖掘出来的规则相同或者略少一些。挖掘结果如表 12 所列。

表 12 Apriori 算法和本算法挖掘规则数和用时

数据集	Apriori 算法		本文算法	
	规则数/个	时间/s	规则数/个	时间/s
Tic-tac	8	1.575	8	0.735
Breast	8	1.762	6	0.297
Monk-1	17	0.358	12	0.219
Zoo	10	4.454	9	0.187

从表 12 可以看出，对于同一数据集，当本算法的最小置信度设置为 0 时，挖掘出的关联规则数和 Apriori 的很接近。但是由于 Apriori 算法需要多次扫描数据库才能得到频繁规则，而本算法只需要扫描一次即可，因此在时间上本算法要比 Apriori 算法快很多。

图 1 为 Apriori 算法和本算法同时在 mushroom 数据集上的挖掘结果。可以看到，随着记录数的增加，Apriori 算法的用时增加很快，而本文算法挖掘用时相对稳定且很短。

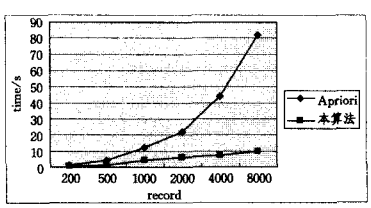

图 1 mushroom 数据集的测试结果

图 2 为两种算法最小支持度设置与执行时间的性能比较。由于在 Apriori 算法中采用的是按层次搜索的方法，挖掘过程是一个迭代的过程，需要频繁扫描数据库，而且在挖掘频繁项集的过程中产生了大量的候选项集，从而造成算法的时间开销增加，执行效率低下。而本文算法因为具有约简冗余属性、只一遍扫描数据库和只用简单的交运算即可求置信度等优点，所以挖掘用时相对较少，效率较高。实验结果表明，本文算法明显优于传统的 Apriori 算法。

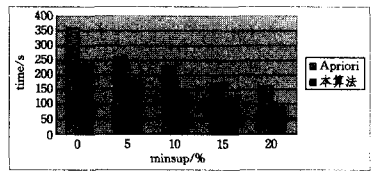

图 2 connect 数据集的测试结果

结束语 本文提出了一种基于粗糙集、单事务项集组合和集合运算的关联规则挖掘算法，解决了传统的 Apriori 算法需要多次扫描数据库、产生庞大的候选项集和不支持更新挖掘的问题。本算法的优势在于不仅约简了数据集中的冗余属性，而且一次扫描数据库，因此会大大缩短算法所需的时间和空间，所以它的时间复杂度和空间复杂度都比 Apriori 算法小很多，对大型多媒体数据库效果会非常显著；且由于本算法保留了备用候选项集，因此很好地支持了数据库的更新挖掘。实验表明，本文算法明显优于 Apriori 算法，是一种有效且快速的关联规则挖掘算法。

参 考 文 献

[1] Agrawal R, Imieunski T, SwAMl A. Mining association rules between sets of items in large database[A]// Proc. of the ACM SIGMOD Intl Conf. on Management of Data[C]. Washington D C., 1993: 207-216

[2] 王鑫. 一种改进的数据挖掘算法——Improve 算法[J]. 计算机应用, 2007, 27(6): 14-15

[3] Han Jia-wei, Pei Jian. Mining frequent patterns without candidate generation[C]// Proceedings of the 20th ACM SIGMOD International Conference on Management of Data. New York: ACM, 2000: 1-12

[4] 李晓虹，杨有. 一种基于线性链表的关联规则挖掘算法[J]. 计算机科学, 2007, 34(9): 142-144

[5] Seochun-dong, Kihung-gu. Efficient single-pass frequent pattern mining using a prefix-tree[J]. Information Sciences, 2008, 179: 559-583

[6] Pawlak Z. Rough sets[J]. International Journal of Computer Information Science, 1982, 11(5): 341-356

[7] Pawlak Z. Rough Sets-Theoretical Aspects of Reaoning About Data[M]. Dordrecht: Kluwer Academic Publishers, 1991: 1-51

[8] 巴斯蒂安 M. 数据仓库与数据挖掘[M]. 武森，高学东，译. 北京：冶金工业出版社, 2003

[9] Frank A, Asuncion A. UCI Machine Learning Repository [DB/OL]. http://archive.ics.uci.edu/ml, 2010