文章编号: 1001-0920(2010)01-0037-06

含正负项目的基于位串频繁项集挖掘算法研究

张玉芳[1a], 熊忠阳[1b], 王 灿[2], 刘春泳[1a]
(1. 重庆大学 a. 计算机学院, b. 电气工程博士后流动站, 重庆 400030; 2. 华为技术有限公司 成都研究所, 成都 610040)

摘 要: 对频繁模式树中的每个节点引入一个位串存储前缀路径,提出了包含正负项目的频繁模式树的构造方法,它不需要反复遍历节点就可获得包含正负项目的频繁项集. 与直接使用 FP-growth 算法相比,无需对原始数据库进行负项目的扩展,也不用再构造并销毁额外的数据结构,只需在原始的频繁模式树上修改,因而在时空开销上都具有一定的优势. 实验表明,所提出的算法比现有的同类挖掘算法和直接 FP-growth 算法具有更好的效率.

关键词: 负项目; 关联规则; 频繁模式树; 频繁项集
中图分类号: TP311 **文献标识码**: A

Study on association rules with negative items based on bit string

ZHANG Yu-fang[1a], *XIONG Zhong-yang*[1b], *WANG Can*[2], *LIU Chun-yong*[1a]
(1a. College of Computer Science, 1b. Post-Doctorial Research Station of Electrical Engineering, Chongqing University, Chongqing 400030, China; 2. Chengdu Institute, Huawei Technologies Company Limited, Chengdu 610040, China. Correspondent: ZHANG Yu-fang, E-mail: zhangyf@cqu.edu.cn)

Abstract: A bit string for each node of the frequent pattern tree is imported to store the prefix. A method for constructing frequent pattern tree which contains positive and negative items is proposed, and frequent item sets with negative items are mined through extending frequent patterns on the tree. Comparing to direct using FP-growth algorithms, this algorithm needs not expand negative item to original database, and construct or destruct additional data structures, which only makes some changes on the original frequent pattern tree, so it has certain advantages in time and space costs. Experiments show that the algorithm has better efficiency than the existing similar mining algorithms and direct FP-growth algorithms.

Key words: Negative item; Association rules; Frequent pattern tree; Frequent item set

1 引 言

传统关联规则[1]的研究主要用于发现大量数据中项集之间有意义的正关联关系,它对于某些问题无法解释. 例如"某些数据项的出现是否常常会导致另外一些数据项不出现","某些数据项不出现是否常常会导致另外一些数据项出现或不出现". 这就是隐藏在数据中项集之间的另外一种关系——负关联关系. 在很多商业决策中,为了最小化不利因素以及最大化有利因素,必须既要考虑有利因素出现的概率,又要考虑不利因素出现的概率以及这些因素之间的正负关联关系. 因此,关联规则的正负性可以视为一个问题的两个方面,其中含负项目的关联规则对决策的作用是不容忽视的,它能反映出更多隐含在事务中更为完整、更有价值的信息,使决策更加客观、更加真实有效. 因而研究同时包含正负项目的关联规则具有十分重要的意义.

目前,对含负属性的关联规则的研究主要集中在对规则前件和后件要么为全正、要么为非全正的笼统的负关联规则方面. 具体方法主要有两类:直接 Apriori 算法和间接 Apriori 算法.

直接 Apriori 算法[2,3]的思想是将事务集视为一个布尔矩阵,从初始的项目集和其补集中挖掘关联规则. 在理论上,它能够挖掘出所有的关联规则,但是需要扩充原始数据集的补集,这无疑将数据库变得更加庞大,在实际应用上是不可取的.

间接 Apriori 算法[4]不需要对原始的事务数据

收稿日期: 2008-12-29; **修回日期**: 2009-05-18.
基金项目: 教育部留学回国人员启动基金项目(教外司留[2007]1108-10); 中国博士后科学基金项目(20070420711).
作者简介: 张玉芳(1965—),女,上海人,副教授,博士,从事数据挖掘、远程教育等研究; 熊忠阳(1962—),男,重庆人,教授,博士,从事数据挖掘、并行处理等研究.

集进行任何处理,它只需要对原数据集进行扫描,计算正项集的支持度计数,通过一些定理和公式推导出含负项目的项集的支持度计数. 这样做的一个明显好处是不需要对事务数据集进行转换,不会加重扫描任务.

这两类方法都是基于 Apriori 算法[5]的基本思想,因而无法避免 Apriori 算法反复扫描数据集以及产生大量候选项集这两大缺陷. 针对这一问题,本文提出一种类似于 FP-growth 思想的包含正负项目的频繁模式树的挖掘算法,期望挖掘出包含正负项目的频繁项集. 实验结果表明了该算法的有效性.

2 含负项目的频繁模式树

2.1 含正负项目的关联规则表示

传统关联规则主要针对正属性之间的联系进行研究,缺乏一般性. 从逻辑角度,仅由"并且"和"蕴涵"构成的逻辑表达式在描述能力上是不完备的. 从实际需求来看,不考虑负属性(否定)可能会丢失富有意义的规则. 目前,对规则前件或后件混合正负项目的关联规则的研究还较少. 为后续描述的需要,给出一种包含正负项目的关联规则描述[6],此描述中规则前件或后件既可以是正项,也可以是负项,还可以同时包含正负项.

定义 1(含正负项目的关联规则) 设 $I = \{i_1, i_2, \cdots, i_n\}$ 为项集,$D = \{T_1, T_2, \cdots, T_m\}$ 为事务集,其中 $T_k \subseteq I(1 \leq k \leq m)$ 为事务. 含正负项目的关联规则是一个形如 $p_1 \wedge \cdots \wedge p_t \wedge \overline{p}_{t+1} \wedge \cdots \wedge \overline{p}_r \Rightarrow q_1 \wedge \cdots \wedge q_k \wedge \overline{q}_{k+1} \wedge \cdots \wedge \overline{q}_s$ 的蕴涵式. 其中:$p_i, q_j \in I(1 \leq i \leq r, 1 \leq j \leq s)$,且 $\{p_1, p_2, \cdots, p_r\} \cap \{q_1, q_2, \cdots, q_s\} = \Phi$.

为简单起见,将含正负项目的关联规则简记为 $X \Rightarrow Y$,其中 X, Y 是包含正负项目的项集. 若无特别说明,本文后面所涉及的规则 $X \Rightarrow Y$ 均指包含正负项目的关联规则.

定理 1 对于任意的负项目 $\overline{i}$,设它所对应的正项目 i 的支持度计数(即在数据库中出现的次数)为 $i.\mathrm{count}$,则 $\overline{i}$ 的支持度计数表示为

$$\overline{i}.\mathrm{count} = |\mathrm{DB}| - i.\mathrm{count},$$

其中 $|\mathrm{DB}|$ 是数据库中事务的总数.

定理 1 表示在事务数据库 DB 中,不包含项目 i 的事务数目等于整个数据库的事务数目 $|\mathrm{DB}|$ 减去包含 i 的事务数目. 因为一个项目在每条交易中只有两种状态:出现或不出现,而这两种状态在同一条交易中能且仅能出现一种,所以定理 1 成立.

原始事务数据集中只包含了交易中出现的项目,为了能够统计出其中未出现项目的支持度计数,文献[2,3]采用将各交易中所有未出现项目的互补项目(即负项目)作为特殊项目扩展到原始数据库中,再将它们同普通项目一样看待,通过扫描数据库,统计其支持度计数. 从前面分析可知,这样会使挖掘任务更加繁重.

扫描原始数据库可以统计出所有正项目的支持度计数,利用定理 1 计算出所有负项目的支持度计数,然后将所有支持度计数不小于最小支持度计数 Min-Count 的正、负项目合并成一个集合,作为含正负项目的频繁 1- 项集 L_1. 为方便起见,用正整数表示正项目,负整数表示负项目,将正负各项按照支持度计数降序排列,形成一个有序序列.

由于负项目的引入,原本稀疏的数据库可能成为稠密的数据库,而原本稠密的数据库将变得更加稠密. FP-growth 算法[7]采用的频繁模式(FP)树是一种对交易数据高度压缩的数据结构. 对稠密数据库而言,FP 树能更好地共享前缀模式,获得更大的压缩率,更能体现出算法的优越性. 因此,本文借鉴 FP 树思想,构造一棵含正负项目的频繁模式树,再进行含正负项目的频繁项集的挖掘.

2.2 含正负项目的频繁模式树的构造

对 FP 树[7]的定义进行扩充,得到含负项目的频繁模式(FPN)树[6]. 与 FP 树定义不同的是,FPN 树中节点表示的项目名可以是正项目,也可以是负项目;同时 FPN 树的频繁项头表的表项除了包含原有的两个域 item-name 和 node-head 外,还增加了项目在树中的支持度计数 item-count. 具体构造时将频繁负项目同频繁正项目一样对待,与 FP 树的构造方法类似.

由于引入了负项目,FPN 树的构造方法与 FP 树有所不同. FPN 树的构造基于以下观察:对于数据库中的每个事务 T,考察含正负项目的频繁 1- 项集 L_1 中的每个频繁项目,如果该频繁项为正项目且出现在 T 中,则说明 T 含有该正频繁项;否则,T 不含该正频繁项. 如果该频繁项为负项目,且其对应的正项目不出现在 T 中,则说明 T 中隐含有该负频繁项;否则,T 中没有隐含该负频繁项.

与文献[6]不同的是,FPN 树构造时对频繁模式树中每个节点增加一个位串来存储该项目的前缀项目,以避免在模式扩展时频繁遍历子树,从而具有更好的效率.

每个项目节点包括 4 个部分:项目名(item-name),项目计数(node-count),节点指针(node-link)和前缀路径(route-map). 其中:项目名记录节点表示的项目名字,可以是正项目,也可为负项目;项目计数记录从根节点到达该节点的路径所表示的项目集的事务数;节点指针指向下一个同名节点,若

不存在下一个同名节点,则其值为空(null);而前缀路径通过一个位串记录从根节点到达该节点的路径所表示的项目集中的各个项目. 位串长度为频繁 1-项集的个数,即频繁项目头表的长度,其中每一位对应一个频繁项目. 若从根节点到达该节点的路径中包含了某个频繁项目,则对应位为 1;否则为 0.

频繁项头表的表项包括 3 个部分:频繁项目名(item-name),项目支持度计数(item-count)和头指针(node-head). 其中:频繁项目名记录满足最小支持度的频繁项目的名字;项目支持度计数表示项目在树中的支持度计数;头指针指向树中第 1 个同名节点.

FPN 树的构造算法如下:

输入:事务数据库 DB,最小支持度 Min-Supp;

输出:FPN-tree NT.

方法:

1) 扫描事务数据库 DB 一次,将所有项目及其在数据库中出现的次数放入集合 C^+.

2) 计算 C^+, C^- 中各项目 item 对应的负项目的支持度计数.

3) 统计 C^+, C^- 中的频繁项并按支持度计数递减排序,将满足最小支持度计数条件的项目存入 L_1 构成频繁 1- 项集.

4) 创建树 T 的根节点 R,以"null"标记它;前缀路径 R.route = 0.

5) 创建频繁项头表 Table,各项目按支持度递减的顺序存入表中,对应的支持度计数初始化为 0,指向各项目同名节点的头指针 node-head 初始化为"null".

6) 对 DB 中的每个事务 Trans,执行如下过程:

// 将事务 Trans 中的正、负频繁项按照 L_1 中项目的顺序插入树 T 中

CN = 树 T 的根节点

for (i = 1; $i \leq |L_1|$; i++) // L_1 为频繁 1- 项集

{

if (($L_1[i] > 0$ && Trans 中) || ($L_1[i] < 0$ && $-L_1[i]$ 不在 Trans 中))

// 若是正频繁项且事务中含有该正频繁项

// 或者为负频繁项且事务中不含有负频繁项对应的正项目,即隐性含有该负频繁项

{

$p = L_1[i]$

if (CN 有子女 Q 使得 Q.item-name = p.item-name)

{

Q.node-count++;

CN = Q

}

else

{

创建新节点 Q

Q.item-name = p.item-name, Q.node-count = 1

Q.node-link 链接到同名节点

根据 Q.item-name 在头表 Table 中的序号,将 CN.route 的对应位置为 1 并赋给 Q.route

将 Q 链接到父亲节点 CN

CN = Q

}

}

}

为了更好地理解上述 FPN 树的构造过程,以表 1(表中左边两列为原始数据,最小支持度计数设为 3) 事务数据库为例说明 FPN 树的构造,如图 1 所示.

表 1　事务数据库

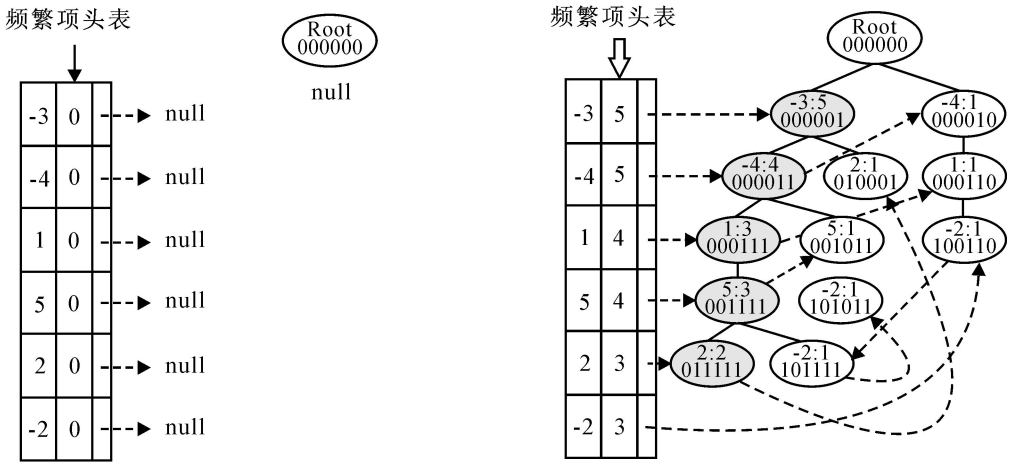
事务 TID	项目 item-name	事务中的正、负频繁项目排序 $[p \mid P]$
T_1	1,2,5	{-3,-4,1,5,2}
T_2	2,4	{-3,2}
T_3	1,3	{-4,1,-2}
T_4	1,5	{-3,-4,1,5,-2}
T_5	5	{-3,-4,5,-2}
T_6	1,2,5	{-3,-4,1,5,2}

将 C^+ 和 C^- 中支持度计数不小于 3 的正、负项目按照支持度计数递减排列,得到频繁 1- 项集 L_1 = {-3:5,-4:5,1:4,5:4,2:3,-2:3}. 创建 FPN 树的根节点 null 和频繁项头表,见图 1(a). 对各事务按正、负频繁项目排序后的 $[p \mid P]$(表 1 中右边一列),分别将其插入 FPN 树,得到如图 1(b) 所示的最终 FPN 树.

3　基于 FPN 树的频繁模式挖掘

3.1　相关定义和定理

构造好的 FP 树可以利用 FP-rowth 算法挖掘出所有频繁模式. 虽然 FP-growth 算法具有无需反复扫描数据库、不会大量生成候选模式的优点,但它需要不断地生成和销毁大量的条件频繁模式树,这在空间和时间上都造成了一定的浪费. 为克服此缺陷,本文提出一种基于 FPN 树而无需生成条件模式树的频繁模式挖掘算法 ——MFPN (Mining



(a) 初始化树 T (b) 事务全部插入后构造好的树

图 1 FPN树的构造

frequent pattern with positive and negative items). 为算法描述的需要,先引入下述定义和定理.

定义 2 从根节点到树中某个节点的路径为 $P = \{i_1, i_2, \cdots, i_k\}$(项目按支持度递减排序),称节点 i_k 的项目计数为路径 P 的路径支持度计数. 设项集 $S = \{j_1, j_2, \cdots, j_m\}$(项目按支持度递减排序),若 $S \subseteq P$ 且 $i_k = j_m$,则称路径 P 包含项集 S.

定理 2 项集 S 的支持度计数等于 FP 树中所有包含 S 的路径的支持度计数之和.

证明 由支持度计数的定义可知,项集 S 的支持度计数即为包含 S 的事务个数. FP 树压缩存储了事务数据库中的所有事务,因此所有包含 S 的事务都存储在包含 S 的路径中. 由 FP 树的构造可知,某个包含 S 的路径 P 所存储的包含 S 的事务个数,为路径 P 最末节点的项目计数,因此所有这样的项目计数之和就是项集 S 的支持度计数. □

定理 3(Apriori 性质)[1] 频繁项集的所有非空子集都必须是频繁的.

推论 1(Apriori 推论) 如果一个非空项集是非频繁项集,则它的任何超集都是非频繁的.

在已经构造好的 FP 树中存储了原始数据集中出现的正项目和负项目的频度信息,从频繁 1- 项集(树中的一个节点表示) 开始,采用模式增长的方法挖掘出所有的频繁项集. 由 Apriori 性质及其推论可知,非频繁项集的任何超集都是非频繁的,因此仅需要对频繁项集进行增长,即将已确定的频繁项集增加一个频繁项并考察其频繁性,对于非频繁项集则不再做增长. 通过这样的增长方式来挖掘出所有的频繁项集.

频繁项集挖掘的基本思想是:从频繁项头表中的最后一项(即支持度最低的频繁项) 开始,从后往前的顺序对频繁项头表中的每一个频繁项 a 进行下面的操作:扫描 FPN 树,统计频繁项头表中每一个在 a 之前的项与 a 组成的项集的支持度计数,满足支持度阈值条件的并入频繁 2- 项集中;再按照频繁 2- 项集的最末项 b 在频繁项头表中从前往后的顺序考察各频繁 2- 项集,统计频繁项头表中每一个在 b 之前的项与该频繁 2- 项集组成的项集的支持度计数,满足支持度阈值条件的并入频繁 3- 项集中;如此继续递归下去,直到不能扩展为止,然后再对频繁项头表中的下一个项目进行同样的操作. 其中最关键的部分是如何统计项集扩展后的支持度计数,下面详细介绍.

在频繁 k- 项集扩展为频繁 $k+1$- 项集时,设某个频繁 k- 项集的第 1 项为 a(项集中支持度最低的项),最末项为 b(项集中支持度最高的项).

首先,将频繁项头表中位于 b 之前的各个项目的 item-count 全部清零,用于存储扩展后的项集的支持度.

然后,通过频繁项头表中 a 项的头指针找到树中的第 1 个含 a 的节点,通过该节点的前缀路径(route-map) 很容易知道其路径上包含哪些节点,若其路径包含该频繁 k- 项集,则对于其路径上的节点,若其在项目头表中的位置在 b 之前,则将其项目头表中的 item-count 加上该含 a 节点的 node-count 值(即路径支持度计数) 再赋给 item-count. 再通过该节点的节点指针(node-link) 找到下一个同名节点作相同操作,直到节点的 node-link 为空时为止.

最后,考察频繁项头表中位于 b 之前的各个项目的 item-count 值(设为 m). 对于 m 满足最小支持度计数的项就可加入 k- 项集的尾部,成为一个频繁 $k+1$- 项集,其支持度计数就是 m.

3.2 基于 FPN树的频繁模式挖掘算法

算法 1 包含正负项目的频繁模式挖掘算法

MFPN(L_k).

输入:频繁 k- 项集集合 L_k,参数 Min-Supp,构造好的频繁模式树 FPN 树;

输出:包含正负项目的频繁项集 L.

描述:

对频繁 k- 项集集合 L_k 中第 i 个 k- 项集 l_i 的第 1 项 a,最末一项 b

{

对每一个在频繁项头表中位于 b 之前的项目 p(从前往后考察),修改头表: p.item-count ←0;

通过 a 的同名节点链,扫描所有含 a 的节点,对每一个节点:

{

如果节点 a 的前缀路径包含项集 l_i(由 a.map 可以判断是否包含),则对路径中所有位于 b 之前的节点 q 修改项头表中 q 的支持度计数: q.item-count q.item-count + a.note-count;

}

对头表中 b 之前的每一个频繁项 p(从前往后扫描):

{

if p.item-count/N ≥Min-Supp(N 为数据库中事务总数) then

{

$\beta = l_i \cup p$; β.support = p.item-count/N;

$L_{k+1} \leftarrow L_{k+1} \cup \beta$;

}

}

$L \leftarrow L \cup L_{k+1}$;

调用 MFPN(L_{k+1});

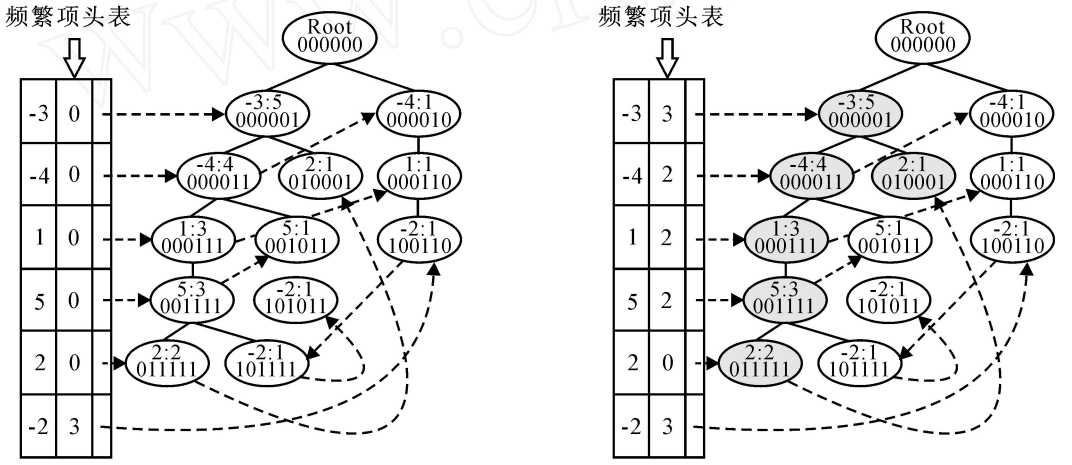
}

图 2 给出了从 FPN 树中挖掘频繁模式的过程.

4　实验及结果分析

实验平台为 P4 3.0,1 G内存 WindowsXP 操作系统.实验所采用的数据集有两个:一个是模拟数据集,由 IBM 数据生成器生成,包括 779 条事务,20 个项目,是一个比较稀疏的数据集;另一个是 Chess 数据集[9],是一个真实数据集,其数据较稠密且高度关联.

实验的比较对象有两个:1) 将负项目扩展到原始数据集后,直接采用 FP-growth 算法;2) 文献[6]



(a) 初始化 Subtree(1)　　(b) Subtree(1)扫描完成后

图 2　从 FPN 树中挖掘频繁模式

表 2　模拟数据集的频繁项集生成时间比较

数据集	支持度阈值	频繁项集总数	正频繁项集个数	挖掘时间/s		
				FP-growth	MFPPN[6]	MFPN
稀疏数据集	0.2	470175	369	201.7	14.3	8.8
	0.3	143999	164	20.1	5.0	2.6
	0.4	50735	76	2.6	2.1	0.9
	0.5	16847	46	0.4	0.7	0.3
Chess 数据集	0.84	2021078	3484	1780.2	181.5	36.5
	0.86	747119	1987	467.2	48.5	13.4
	0.88	259153	1195	66.5	15.8	4.6
	0.90	90922	622	7.6	2.4	1.7
	0.92	24487	305	1.1	0.7	0.5

提出的挖掘含负项目的频繁项集算法 MFPPN. 表 2 给出了对两个数据集设置不同的支持度阈值的实验结果.

Chess 是一个较大的真实数据集,并且相当稠密,尤其是考虑了负项目后,数据集变得更为稠密,因此在实验中主要比较了支持度较高的情况. 从实验结果数据可以看出,无论原始数据集是稀疏还是稠密,当支持度阈值较高,频繁项集个数较少时,3 种算法的挖掘时间相差不大;而当频繁项集个数较多时,MFPN 算法的时间远远小于直接 FP-growth 算法;而与 MFPPN 算法相比也有一定提高. 由此可见,MFPN 算法对于挖掘含负项目的频繁项集是有效的,且具有较高的效率.

5 结 论

在不扩充原始数据库的条件下,将原始数据库中的频繁正、负项目插入频繁模式树中,并对 FP 树的构造作了适当改进,为每个节点引入一个位串来存储其前缀路径,因此挖掘过程中不需要反复遍历节点,使挖掘效率得到提高. 实验结果表明,本文算法比现有的同类挖掘算法和直接 FP-growth 算法具有更好的效率.

由于事务项中引入负项目后,事务中的各项存在极大的相关性,虽然 MFPN 挖掘出了完备的包含正负项目的频繁项集,但是数目较多,同时很多含正负项目的频繁项集都是由频繁的负项目构成,甚至全是频繁负项目,这些对应的规则可能在某些领域没有太大的应用价值. 如何有效地限制频繁项集,只挖掘出某领域有实用价值的频繁项集是今后进一步研究的方向.

参考文献(References)

[1] Jiawei Han, Micheline Kambr. Data mining concepts and techniques[M]. Beijing: Higher Education Press, 2001.

[2] Jean-Franqois Boulicaut, Artur Bykowski, Baptiste Jeudy. Towards the tractable discgvery of association rules with negations[C]. FQAS′00. Warsaw, 2000: 425-434.

[3] 武鹏程, 袁兆山. 混合关联规则及其挖掘算法[J]. 小型微型计算机系统, 2003, 24(5): 895-898.
(Wu P C, Yuan Z S. Hybrid association rules and their mining algorithms[J]. Mini-micro Systems, 2003: 24(5): 895-898.)

[4] 卢生炎, 饶丹. 一种挖掘带否定关联规则的算法[J]. 计算机工程与科学, 2004, 26(10): 63-65.
(Lu Y S, Rao D. A mining algorithm for association rules with negation[J]. Computer Engineering & Science, 2004, 26(10): 63-65.)

[5] Rakesh Agrawal, Ramakrishnan Srikant. Fast algorithms for mining association rules[C]. Proc of the 20th VLDB Conf. Chile, 1994: 487-499.

[6] 张玉芳, 熊忠阳, 彭燕, 等. 基于 FP-Tree 含正负项目的频繁项集挖掘算法[J]. 模式识别与人工智能, 2008, 21(2): 246-253.
(Zhang Y F, Xiong Z Y, Peng Y, et al. Study association rules with negative items based on FP-Tree[J]. Pattern Recognition and Artificial Intelligence, 2008, 21(2): 246-253.)

[7] Jiawei Han. Mining frequent patterns without candidate generation[C]. Proc of 2000 ACM SIGMOD Int Conf. Dalas, 2000: 1-20.

[8] Agrawal R, Imielinaki T, Swami A. Mining association rules between sets of items in large databases[C]. Proc of the ACM SIGMOD Conf on Management of Data. Washington DC, 1993: 207-216.

[9] http://fimi.cs.helsinki.fi.

[10] 李彤岩, 肖海林, 李兴明. 通信网告警加权关联规则挖掘算法的研究[J]. 电子科技大学学报, 2008, 37(6): 807-810.)
(Li T Y, Xiao H L, Li X M. Algorithm for mining weighted alarm association rules in telecommunication networks[J]. J of University of Electronic Science and Technology of China, 2008, 37(6): 807-810.)

[11] 刘君强, 孙晓莹, 王勋, 等. 挖掘最大频繁模式的新方法[J]. 计算机学报, 2004, 27(10): 1329-1334.
(Liu J Q, Su X Y, Wang X, et al. A new algorithm for mining maximal frequent patterns[J]. Chinese J of Computers, 2004, 27(10): 1329-1334.)

(上接第 36 页)

[10] Fu M, Xie L. The sector bound approach to quantized feedback control[J]. IEEE Trans on Automatic Control, 2005, 50(11): 1698-1711.

[11] 唐功友, 张勇. 一类不确定性非线性网络控制系统的扰动抑制[J]. 控制与决策, 2008, 23(9): 1055-1064.
(Tang G Y, Zhang Y. Disturbance rejection for a class of uncertain nonlinear networked control systems[J]. Control and Decision, 2008, 23(9): 1055-1064.)