基于时间戳和垂直格式的关联规则挖掘算法

王 斌 马俊杰 房新秀 魏天佑
(青岛理工大学信息与控制工程学院 山东 青岛 266520)

摘 要 基于时间戳的关联规则挖掘算法(SLMCM)主要用于解决新增项的问题,但效率较低,难以适应大数据挖掘。针对这个问题,文中提出了改进算法 E-SLMCM 和 DE-SLMCM。E-SLMCM 算法采用垂直结构,仅需遍历数据库两次,在将数据库转化为垂直格式时,可直接记录各项的时间戳,且不需要将每条事务的各项按时间戳进行排序;另外,提出了新的求项集时间戳的方法,在求更高项集的时间戳时不用多次遍历数据库。E-SLMCM 算法适合应用于稀疏数据库,为了提高在密集数据库上的运行效率,在 E-SLMCM 算法的基础上采用差集思想提出了 DE-SLMCM 算法。所列举的 4 个基于公共数据集的仿真实验中,在不同最小支持度条件下,E-SLMCM 和 DE-SLMCM 分别在稀疏和密集数据集上运行的时间效率是 SLMCM 的 10～1000 倍。

关键词 关联规则,时间戳,新增项,差集
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/jsjkx.190100223

Association Rule Mining Algorithm Based on Timestamp and Vertical Format

WANG Bin MA Jun-jie FANG Xin-xiu WEI Tian-you
(School of Information and Control Engineering, Qingdao University of Technology, Qingdao, Shandong 266520, China)

Abstract The SLMCM algorithm (Specific Later-marketed Consequent Mining) is mainly used to solve the problem of later item, but it is inefficient and difficult to adapt to big data mining. For this problem, this paper proposed the improved algorithms E-SLMCM and DE-SLMCM . E-SLMCM algorithm is based on vertical structure, so it only traversal the database twice. Furthermore, the timestamp of each item can be directly calculated when the format is converted to vertical, and each transaction does not need to be sorted by the timestamp of the item. In addition, a new method for finding the itemset timestamp was proposed, which dose not need to traverse the database to find the timestamp of itemset. In order to adapt to dense database, DE-SLMCM algorithm was proposed based on E-SLMCM algorithm and diffset, which improves the execution efficiency on dense database. In the listed four simulation experiments based on common data sets, the time efficiency of E-SLMCM and DE-SLMCM running on sparse and dense data sets is 10-1000 times higher than that of SLMCM.

Keywords Association rules, Timestamp, Later item, Diffset

1 引言

数据库的更新是研究关联规则挖掘算法时需要考虑的重要问题。针对这个问题,目前已有众多以 Apriori[1],FP-tree[2]和 Eclat[3]等经典算法为基础的算法。

在关联规则挖掘问题中,数据库的更新可以分为事务的更新和项的更新。文献[4-11]描述的算法主要是针对数据库事务更新进行的研究,没有涉及数据库项的更新。在实际应用中,项的更新是经常需要面对的问题,准确地挖掘新增项的关联规则面临的主要问题是新增项的支持度太低。文献[12-15]描述的算法都能在不同程度上解决这个问题,但其被用来挖掘新增项的关联规则时还有很多欠缺之处。

针对新增项问题,Weng[16]提出在 Apriori 算法中添加时间戳的算法——SLMCM。该算法在遍历数据库时,设置了一个时间戳来记录各项集首次出现的时间,并从这个时间开始计算这个项集对应的事务总量,以此为基础来计算支持度,使项集的支持度增大。但 SLMCM 算法是以 Aprior 算法为基础的,运行效率较低。为了提高效率,本文提出了 E-SLMCM 算法,此算法的数据库改用垂直结构,采用集合枚举树结构[17]和降序思维[18],并对求时间戳的方法进行改进。由于 E-SLMCM 算法在密集型数据库上的运行效率不高,本文又结合差集 diffset 思想[19]提出了 DE-SLMCM 算法。在解决新增项问题时,若数据库是稀疏型的,则采用 E-SLMCM 算法;若数据库是密集型的,则采用 DE-SLMCM 算法。

本文第 2 节详细描述了算法需要解决的新增项的问题,并介绍了关联规则和时间戳的基本概念;第 3 节简单介绍了

到稿日期:2019-01-26 返修日期:2019-05-05 本文受国家自然科学基金项目(61502262)资助。
王 斌(1963－),男,博士,教授,主要研究方向为知识发现、博弈论及应用等,E-mail:wb769@sina.com(通信作者);马俊杰(1992－),男,硕士,主要研究方向为数据挖掘等;房新秀(1994－),女,硕士,主要研究方向为数据挖掘等;魏天佑(1994－),男,硕士,主要研究方向为数据挖掘等。

SLMCM 算法；第 4 节和第 5 节分别介绍了提出的 E-SLMCM 算法和 DE-SLMCM 算法；第 6 节对两种算法的时间复杂度进行了分析；第 7 节基于公共数据库进行实验并对实验结果进行了分析；最后总结全文与展望未来。

2 基本概念和定义

2.1 关联规则

设 $I=\{i_1,i_2,i_3,\cdots,i_m\}$ 是数据库中所有项的集合，数据库 D 是数据库中所有事务的集合，D 中每个事务 T 都是非空集，且 $T\subseteq I$，每个事务都有一个标识号，用 TID 表示。若 X 是 k 个项的集合，则称 X 为 k 项集；当且仅当 $X\subseteq T$ 时，称事务 T 包含 X 项集。

在整个数据库 D 中，包含项集 X 的事务数称为项集 X 的支持度计数，用 $|X|$ 表示。项集 X 的支持度计数和 D 中事务总数的比称为项集 X 的支持度，用 $sup(X)$ 表示。若 D 中事务的总数用 $|D|$ 表示，则求项集支持度的方法如式(1)所示：

$$sup(X)=\frac{|X|}{|D|} \tag{1}$$

若项集 X 的支持度满足设定的最小支持度阈值，则 X 是频繁项集。最小支持度阈值用 min_sup 表示，频繁 k 项集的集合通常记为 L_k。

先验性质的定义为：如果项集 X 不是频繁项集，则项集 $Y(X\subseteq Y)$ 也不是频繁项集；如果项集 Y 是频繁项集，则任意非空项集 $X(X\subseteq Y)$ 都是频繁项集。先验性质是众多关联规则算法进行剪枝优化的基础，利用先验性质可以避免大量不必要的计算。

2.2 新增项

新增项是指新加入数据库的项，如超市交易记录数据库中新增加的商品等。与新增项有关的关联规则往往是决策者关注的重点。由于新增项出现在数据库中的时间较晚，因此用传统方法计算得出的与新增项有关的项集的支持度会偏低，导致与新增项有关的项集被过滤掉，从而丢失一些有趣的信息。

在用传统的方法进行关联规则挖掘时，项集的支持度等于项集的支持度计数与数据库事务总数的比，所有项集的数据库事务总数都是一样的。对于新增项来说，数据库事务总数中包含了大量的新增项首次出现之前就已经出现的事务，不能正确地反映新增项的支持度，使得新增项的支持度偏小，这是不合理的。

下面通过例 1 说明新增项的问题。

例 1 表 1 列出了一个超市中 3 种商品 x,y,z 在 5－9 月的销售情况。

表 1 新增商品销售记录示例

Table 1 Example for later product

	5 月	6 月	7 月	8 月	9 月
x	100	100	100	100	100
xy		100	100	100	100
xyz				100	100

由表 1 可知：

(1)5－9 月份的交易总数为 500；

(2)出现 xyz 组合的事务数为 200；

(3)出现 xy 组合的事务数为 400。

从不同月份开始计算商品组合 xyz 和 xy 的支持度。

(1)从 5 月份开始计算：$sup(xyz)=200/500=40\%$；

从 8 月份开始计算：$sup(xyz)=200/200=100\%$。

(2)从 5 月份开始计算：$sup(xy)=400/500=80\%$；

从 6 月份开始计算：$sup(xy)=400/400=100\%$。

由计算结果可以看出，从不同月份开始计算同一商品组合的支持度会得出不同的结果。例如，对于商品组合 xyz，从 8 月份开始计算的支持度比从 5 月份开始计算的支持度高。传统计算方法是从 5 月份开始计算的，但是在 8 月份之前，y 商品没有上架，传统计算方法对 y 商品显然是不公平的，从 8 月份开始计算才较为合理。在实际应用中的数据库，特别是商品交易数据库中，项的数量巨大，各个项首次出现在数据库中的时间不统一，为了合理地计算项的支持度与置信度，Weng[16] 提出了时间戳的概念。

2.3 时间戳

定义 1[15] 项或者项集在数据库中首次出现的事务的编号称为项或者项集的时间戳。

定义 2[15] 在计算项集 X 的支持度时，若从 X 的时间戳标记处计数其对应的数据库事务总数，则将项集 X 对应的数据库事务总集记为 $D^t(X)$，对应的数据库事务总数记为 $|D^t(X)|$。

定义 3[16] 若从项集 X 的时间戳标记处计数 X 的数据库事务总数，则将项集 X 的支持度记为 $sup^t(X)$，称为项集 X 的带时间戳的支持度。式(2)给出了 $sup^t(X)$ 的计算方法。

$$sup^t(X)=\frac{|X|}{|D^t(X)|} \tag{2}$$

性质 1[16] 在计算项集 X 的支持度时，$sup(X)\leqslant sup^t(X)$。

定义 4[16] 将数据库中每条事务按项的时间戳排序后得到的数据库称为按时间戳排序的数据库，记为 D_t。

下面通过例 2 来说明时间戳的定义与作用。

例 2 表 2 列出了一个有 a,b,c 3 个项，以及 10 个事务的水平格式数据库 D。表 3 列出了数据库 D 中的各项以及各项对应的时间戳。表 4 列出了将表 2 中的每条事务按表 3 中各项的时间戳排序得到的数据库 D_t。表 5 列出了所有项和项集的时间戳。表 6 列出了以表 1 的数据为基础，对传统关联规则算法和带时间戳的关联规则算法中项集的数据库事务总数和支持度进行对比的结果。

表 2 示例事务集数据库

Table 2 Example transaction date sets

D_{t1}		D_{t2}	
TID	Itemset	TID	Itemset
1	a	6	ac
2	a	7	c,a,b
3	a,c	8	a,b,c
4	c,a	9	a,b,c
5	a,c	10	a,c,b

表 3　数据库中的项及项的时间戳

Table 3　Item and time stamp in database

Item	Time stamp
a	T_1
c	T_4
b	T_7

表 4　按时间戳排序的数据库 D_t

Table 4　Constructed timestamp transaction set D_t

D_{t1}		D_{t2}	
TID	Itemset	TID	Itemset
1	a	6	ac
2	a	7	a,c,b
3	a	8	a,c,b
4	a,c	9	a,c,b
5	a,c	10	a,c,b

表 5　项与项集的时间戳

Table 5　Time stamp of item and itemset

Item	First time	Itemset	Time stamp
a	T_1	ac	T_4
c	T_4	ab	T_7
b	T_7	cb	T_7
		acb	T_7

表 6　项集支持度与带时间戳支持度的比较结果

Table 6　Comparison of itemset support and time stamp support

Item set	Time stamp	Count	$\|D\|$	$\|D^t\|$	$sup/\%$	$sup^t/\%$
ac	T_4	8	10	8	80	100
ab	T_7	4	10	4	40	100
cb	T_7	4	10	4	40	100
acb	T_7	4	10	4	40	100

3　SLMCM 算法

3.1　SLMCM 算法的原理

SLMCM 算法在 Apriori 算法中加入了时间戳,避免了在计算项或项集时将其首次出现之前的事务计入其数据库事务总数的情况。在计算支持度时,项或项集对应的数据库事务总数减少,使得项集的支持度计算结果升高,原本会被过滤掉的项集被保留。

3.2　SLMCM 算法的步骤

SLMCM 算法的具体步骤如下:

(1)遍历数据库,记录项的首次出现时间与支持度计数;建立添加时间戳的事务数据库 D_t。

(2)由 $k-1$ 项频繁项集产生 k 项候选项集;遍历数据库,记录 k 项候选项集的时间戳,并对 k 项候选项集进行计数,计算候选项集带时间戳的支持度,判断候选项集是否为频繁项集;将 k 项频繁项集并入频繁项集总集。

(3)由频繁项集输出关联规则,并计算关联规则的置信度。

SLMCM 算法有效地解决了新增项支持度偏小的问题,但由于其是基于 Apriori 算法的,运行效率很差,因此下文提出基于垂直结构的 E-SLMCM 算法和 DE-SLMCM 算法,两种算法分别适用于稀疏型数据库和密集型数据库。

4　E-SLMCM 算法

4.1　E-SLMCM 算法的原理及相关定义

SLMCM 算法采用的是水平数据格式(即{Tid:itemset}格式),而 E-SLMCM 算法采用的是垂直数据格式(即{itemset:Tidset}格式)。以 Eclat 算法为基础,通过两个前缀相同的 $k-1$ 项集相交得出 k 项候选项集,再由这两个 $k-1$ 项集的事务集相交得出 k 项集的事务集。

相对于 SLMCM 算法,E-SLMCM 从以下 3 个方面提升了算法的时间效率。

(1)E-SLMCM 算法在产生候选项集的同时,即可通过项集的事务集进行交集运算,计算出候选项集的支持度,不用多次遍历数据库。

(2)E-SLMCM 算法采用的是垂直数据格式,在由水平数据格式向垂直数据格式转换时,需要遍历一次数据库以产生各项的事务集。由于遍历是按照事务逐条进行的,因此在数据库中出现早的项将会先建立事务集,最终的结果是项按照出现的时间从早到晚排序的,在创建项的事务集时可以记录项的时间戳。同时,由于采用垂直格式挖掘,E-SLMCM 算法不需要每条事务按项的时间戳排序来产生 D_t,节省了时间。

(3)SLMCM 算法在给一项集添加时间戳时需要遍历数据库,在给更高项集添加时间戳时仍需遍历数据库。而在 E-SLMCM 算法中,计算出频繁一项集的时间戳后即可通过公式直接计算出更高项集的时间戳,不需要再多次遍历数据库,如性质 2 和性质 3 所述。

性质 2　设项集 $X=(i_1,i_2,i_3,\cdots,i_n)$,则:

$$sup^t(X)=\frac{|X|}{\min(D^t(i_1),D^t(i_2),D^t(i_3),\cdots,D^t(i_n))} \tag{3}$$

证明:略。

性质 3　设有 $k-1$ 项集 X_i 与 X_j 的前 $k-2$ 项相同,若 k 项集 $Y=X_i\cup X_j$,则:

$$sup^t(Y)=\frac{|Y|}{\min(|D^t(X_i)|,|D^t(X_j)|)} \tag{4}$$

证明:设

$$\begin{aligned}|D^t(X_i)|&=\min(|D^t(i_1)|,|D^t(i_2)|,\cdots,|D^t(i_{k-2})|,\\&\quad|D^t(i_{k-1})|)\\&=|D^t(i_p)|\\|D^t(X_j)|&=\min(|D^t(i_1)|,\\&\quad|D^t(i_2)|,\cdots,|D^t(i_{k-2})|,|D^t(i_k)|)\\&=|D^t(i_q)|\end{aligned}$$

则可得出:

$$\begin{aligned}|D^t(Y)|&=|D^t(X_i\cup X_j)|\\&=\min(|D^t(i_1)|,|D^t(i_2)|,\cdots,|D^t(i_{k-2})|,\\&\quad|D^t(i_{k-1})|,|D^t(i_k)|)\\&=\min(|D^t(i_p)|,|D^t(i_q)|)\\&=\min(|D^t(X_i)|,|D^t(X_j)|)\end{aligned}$$

因此:

$$sup^t(Y)=\frac{|Y|}{|D^t(Y)|}=\frac{|Y|}{\min(|D^t(X_i)|,|D^t(X_j)|)}$$

4.2　E-SLMCM 算法的步骤

E-SLMCM 算法的具体步骤如算法 1 所示。

算法 1 E-SLMCM 非递归算法

输入:事务集数据库 D,min_sup

输出:带时间戳的频繁项集的集合 L^t

1. T_1=find_1-frequent_tidset(D);$L^t=L^t\cup L_1^t$;
2. T_1=uporder_by_item_suport(T_1);
3. for(k=2;$L_{k-1}^t\neq\emptyset$;k++){
4. for all $X_i,X_j\in T_{k-1}$,j>I,do{
5. if X_i and X_j have the same prefix,do{
6. create R=$X_i\cup X_j$;
7. Tidset(R)=Tidset(X_i)∩Tidset(X_j);
8. sup^t(R)=|R|/ min(|D^t(X_i)|,|D^t(X_j)|);
9. if sup^t(R)>=min_sup,do{
10. insert R and Tidset(R) to T_k;
11. insert R to L_k^t;}}
12. $L^t=L^t\cup L_k^t$;}

5 DE-SLMCM 算法

5.1 DE-SLMCM 算法的原理

E-SLMCM 算法以{itemset:Tidset}结构为基础,在密集数据库中,每个项集的事务集数量过于庞大,求交集运算所花费时间的过长,运行效率较低。

对此,DE-SLMCM 算法加入了差集思想,采用{itemset:diffset}结构。将 k 项集的前 $k-1$ 项组成的集合称为 k 项集的前缀项集,若两个项集的前缀项集相同,则称这两个项集为同一个类。一项集的事务差集为事务总集中不包含这个项的事务的集合。项集 X 的事务差集用 $d(X)$ 表示。若项集 $Y=X_i\cup X_j$,则 Y 的前缀项集是 X_i。若 $(a,b,c)=(a,b)\cup(a,c)$,则项集 (a,b,c) 的前缀项集是 (a,b)。

设 X_i 与 X_j 是同一个类中的项集,$Y=X_i\cup X_j$,且 Y 是以 X_i 为前缀项的项集,则 Y 的事务差集和 Y 的支持度计数的计算公式[19]为:

$$d(Y)=d(X_j)-d(X_i) \tag{5}$$

$$|Y|=|X_i|-|d(Y)| \tag{6}$$

5.2 DE-SLMCM 算法的步骤

DE-SLMCM 算法的具体步骤如算法 2 所示。

算法 2 DE-SLMCM 非递归算法

输入:事务集数据库 D,min_sup

输出:带时间戳的频繁项集的集合 L^t

1. T_{diff_1}=find_1-frequent_diffset(D);$L^t=L^t\cup L_1^t$;
2. T_1=uporder_by_item_suport(T_{diff_1});
3. for(k=2;$L_{k-1}^t\neq\emptyset$;k++){
4. for all $X_i,X_j\in T_{diif\_k-1}$,j>I,do{
5. if X_i and X_j are at the same class,do{
6. create R=$X_i\cup X_j$;
7. d(R)=d(X_j)-d(X_i);
8. |(Y)|=|(X_i)|-|(d(Y))|;
9. sup^t=|(Y)|/min(|D^t(X^1)|,|Dt(X^2)|);
10. if sup^t(R)≥min_sup,do{
11. insert R and d(R) to T_{diff_k};
12. insert R to L_k^t;}}
13. $L^t=L^t\cup L_k^t$;}

6 时间复杂度分析

设数据集含 m 个事务和 n 个项,事务的平均长度为 L,k ($k\geqslant2$)项频繁项集的总数为 $|F_k|$,最长项集的项数为 q。

6.1 SLMCM 算法的时间复杂度

第一次遍历数据集,计算各项的支持度并记录各项的时间戳,对每个事务的各项逐个进行遍历。因为总共有 m 个事务,每个事务有 L 项,所以此步骤的语句频度为 mL。

将每个事务按项的时间戳进行排序,若排序算法采用最优的哈希排序,则时间复杂度与事务的平均长度有关,为 $L\log_2L$。对每个事务排一次序,由于共有 m 个事务,因此此步骤的语句频度为 $mL\log_2L$。

生成 k 项候选项集时,需要将前缀相同的 $k-1$ 项频繁项集两两结合。由于在实际运行数据集之前无法确定哪些项集有相同的前缀,且 3 种算法在此步骤的时间复杂度相同,因此暂时忽略此条件。将 $|F_{k-1}|$ 个 $k-1$ 项频繁项集两两组合并取并集,生成 k 项候选项集,语句频度为 $C_{|F_{k-1}|}^2$。

扫描数据集,计算 k 项集的支持度计数,每个事务平均有 L 个项,从 L 个项中取 k 个项进行组合,产生项集,并将产生的项集计入相应的候选项集,同时将候选项集第一次出现时的事务号作为时间戳。每个事务可以产生 C_L^k 个 k 项集,由于共有 m 个事务,因此此步骤的语句频度为 mC_L^k。

综上,从 2 项集到 q 项集累加,得出 SLMCM 的语句频度为 $mL+mL\log_2L+\sum_{k=2}^{q}(C_{|F_{k-1}|}^2+mC_L^k)$。

6.2 E-SLMCM 算法的时间复杂度

不同数据集有不同的特征,项集的平均事务集长度不能用统一的公式表示,但是可知 k 项集的平均事务集长度与 L 正相关。设项集的平均事务集长度为 $f_k(L)$,L 越大,$f_k(L)$ 就越大。

第一次遍历数据集,将数据集转化为垂直格式并记录时间戳,需遍历所有事务的每个项。由于总共有 m 个事务,每个事务有 L 项,因此语句频度为 mL。

生成 k 项候选项集并求其事务集时,生成 k 项候选项集的时间复杂度与 SLMCM 相同,为 $C_{|F_{k-1}|}^2$。将两个 $k-1$ 项集结合的同时,对两个 $k-1$ 项集对应的事务集取交集,得到 k 项候选项集的事务集,同时可计算时间戳。设交集算法采用最优的交集算法,时间复杂度为两个集合长度的较大值,由于已设两个集合的长度均等于 $f_k(L)$,每个候选项集都需要求事务集,因此求 k 项频繁项集及其事务集的语句频度为 $f_k(L)C_{|F_{k-1}|}^2$。

综上,从 2 项集到 q 项集进行累加,E-SLMCM 的语句频度为 $mL+\sum_{k=2}^{q}f_k(L)C_{|F_{k-1}|}^2$。

6.3 DE-SLMCM 算法的时间复杂度

不同数据集有不同特征,项集的事务差集的平均长度不能用统一的公式表示,但是可知 k 项集的事务差集的平均长度与 $m-L$ 正相关。设其为 $g_k(m-L)$,$m-L$ 越大,$g_k(m-L)$ 就越大。

第一次遍历数据集时,将数据集转化为垂直格式并记录时间戳,语句频度为 mL。

生成 k 项候选项集并求其事务差集时,生成 k 项候选项集的语句频度为 $C^2_{|F_{k-1}|}$。将两个 $k-1$ 项集结合的同时,对两个 $k-1$ 项集对应的事务差集取差集,从而得到 k 项候选项集的事务差集,同时计算时间戳。设差集算法采用最优的差集算法,其时间复杂度为两个集合长度的较大值,由于已设两个集合的长度均等于 $g_k(m-L)$,每个候选项集都需要求事务差集,则求 k 项频繁项集及其事务差集的语句频度为 $g_k(m-L)C^2_{|F_{k-1}|}$。

综上,从 2 项集到 q 项集累加,得 DE-SLMCM 的语句频度为 $mL+\sum_{k=2}^{q} g_k(m-L)C^2_{|F_{k-1}|}$。

6.4　3 种算法的时间复杂度比较

SLMCM 算法与 E-SLMCM 算法的语句频度之差为:$mL\log_2 L+\sum_{k=2}^{q}((1-f_k(L))C^2_{|F_{k-1}|}+mC^k_L)$;SLMCM 算法与 DE-SLMCM 算法的语句频度之差为 $mL\log_2 L+\sum_{k=2}^{q}((1-g_k(m-L))C^2_{|F_{k-1}|}+mC^k_L)$。

在真实数据集中,m 远大于$|1-f_k(L)|$和$|1-g_k(m-L)|$,又由于算法根据支持度进行剪枝,两个项集需要有相同的前缀才能结合产生更高项集的候选项集,且 $C^2_{|F_k|}$ 和 C^k_L 的值相差不大,因此 $\sum_{k=2}^{q}((1-f_k(L))C^2_{|F_{k-1}|}+mC^k_L)$ 与 $\sum_{k=2}^{q}((1-g_k(m-L))C^2_{|F_{k-1}|}+mC^k_L)$ 的值均大于 0,故 SLMCM 算法与 E-SLMCM 算法和 DE-SLMCM 算法的语句频度之差较大,SLMCM 算法的耗时远超 E-SLMCM 算法和 DE-SLMCM 算法。

E-SLMCM 算法与 DE-SLMCM 算法的语句频度之差为:$\sum_{k=2}^{q}(f_k(L)-g_k(m-L))C^2_{|F_{k-1}|}$。

在稀疏数据集中 L 较小,此时 $f_k(L)-g_k(m-L)<0$,则 E-SLMCM 算法在稀疏数据集中的运行效率优于 DE-SLMCM 算法的效率;在密集数据集中 L 较大,此时 $f_k(L)-g_k(m-L)>0$,则 DE-SLMCM 算法在密集数据集中的运行效率优于 E-SLMCM 算法的效率。

7　仿真实验与分析

为了验证所提算法的高效性和可行性,从挖掘出的频繁项集数量、运行时间两个方面对 SLMCM 算法、E-SLMCM 算法和 DE-SLMCM 算法 3 种算法进行对比。

实验环境为 CPU Intel Core i5-3210 2.50 GHz,内存为 DDR3 8 GB,操作系统为 Window10。算法使用的编程语言为 Java 1.8.0,开发工具为 Eclipse2018。实验数据来自于 SPMF[20] 文库。

表 7 列出了对稀疏型数据集 retail(88 162 条事务,sup=0.006)的挖掘结果,表 8 列出了对密集型数据集 mushroom(8124 条事务,sup=0.25)的挖掘结果。从中得出,带时间戳的算法比不带时间戳的算法挖掘出的各项频繁项集数都高。如表 7 中不带时间戳的频繁项集从五项集开始便没有挖掘结果,而带时间戳的频繁项集在五项集之后仍有大量的频繁项集,这些是用传统方法无法挖掘的。

表 7　retail 上项集数的比较

Table 7　Comparison of umber of item sets on retail

	二项集	三项集	四项集	五项集	六项集	七项集	八项集	高于八项集
带时间戳	414	468	374	293	263	210	120	56
不带时间戳	172	69	11	0	0	0	0	0

表 8　mushroom 上项集数的比较

Table 8　Comparison of number of item sets on mushroom

	二项集	三项集	四项集	五项集	六项集	七项集	八项集	高于八项集
带时间戳	296	1080	2320	3407	3907	3797	3098	3489
不带时间戳	241	749	1323	1433	1005	498	192	69

图 1 和图 2 分别给出了 SLMCM 算法、E-SLMCM 算法和 DE-SLMCM 算法 3 种算法在密集型数据集 pumsb 和 chess 上的运行时间对比结果。数据集 pumsb 中事务数较多,有 49 046 条事务;数据集 chess 中事务数较少,有 3 196 条事务。由此可推断:在密集型数据集上,无论事务的多少,3 种算法的运行速度由快到慢为:DE-SLMCM > E-SLMCM > SLMCM。

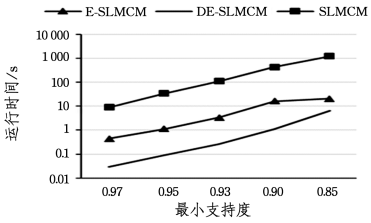

图 1　pusmb 数据集上运行时间的比较

Fig. 1　Comparison of run-time on pusmb

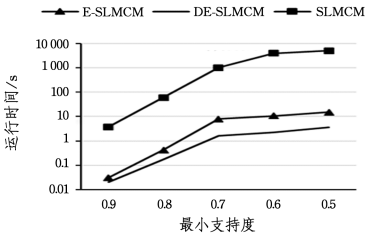

图 2　chess 数据集上运行时间的比较

Fig. 2　Comparison of run-time on chess

图 3 和图 4 分别给出了 SLMCM,E-SLMCM,DE-SLMCM 3 种算法在稀疏型数据集 retail 和 food-mart 上的运行时间的对比情况。数据集 retail 中事务数较多,有 88 163 条事务;数据集 food-mart 中事务数较少,有 4 141 条事务。由此可推断:在稀疏型数据集上,无论事务多少,3 种算法的运行速度由快到慢排列为:E-SLMCM > DE-SLMCM > SLMCM。

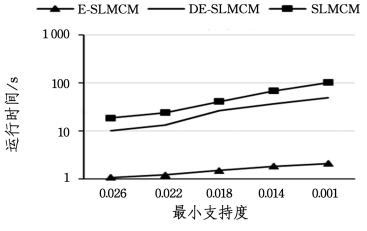

图 3　retail 数据集上运行时间的比较

Fig. 3　Comparison of run-time on retail

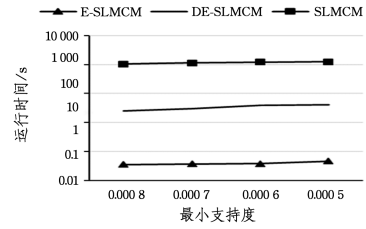

图 4 foodmart 数据集上运行时间的比较

Fig. 4 Comparison of run-time on foodmart

结束语 本文提出的 E-SLMCM 算法与 DE-SLMCM 算法不仅可有效地解决新增项的问题，在效率上也比 SLMCM 算法有显著提高。两种算法的数据库改用垂直结构，并且在记录项的时间戳和计算更高项集的时间戳时采用了更加省时的计算方法；在剪枝方面，采用了集合枚举树和降序思维，大大提高了算法效率；另外，在 E-SLMCM 基础上采用 diffset 思想提出的 DE-SLMCM 算法更是解决了密集数据库上运行效率的问题。实验也证明了 E-SLMCM 算法更适合稀疏型数据库，DE-SLMCM 算法更适合密集型数据库。在下一步的工作中，我们可以在并行处理和分布式处理方面对提出的两种算法进行研究，以提高对大数据的处理效率。

参 考 文 献

[1] AGRAWAL R, IMIELIŃSKI T, SWAMI A. Mining association rules between sets of items in large databases[J]. Acm sigmod record, 1993, 22(2): 207-216.

[2] HAN J, PEI J, YIN Y. Mining Frequent Patterns Without Candidate Generation[C]// Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, USA: ACM, 2000.

[3] ZAKI M J. Scalable algorithms for association mining[J]. IEEE transactions on knowledge and data engineering, 2000, 12(3): 372-390.

[4] RASHID M M, GONDAL I, KAMRUZZAMAN J. Mining associated patterns from wireless sensor networks[J]. IEEE Transactions on Computers, 2015, 64(7): 1998-2011.

[5] WANG L, MENG J, XU P, et al. Mining temporal association rules with frequent itemsets tree[J]. Applied Soft Computing, 2018, 62: 817-829.

[6] CHEN C H, LAN G C, HONG T P, et al. Mining fuzzy temporal association rules by item lifespans[J]. Applied Soft Computing, 2016, 41: 265-274.

[7] ZOU C, DENG H, WAN J, et al. Mining and updating association rules based on fuzzy concept lattice[J]. Future Generation Computer Systems, 2018, 82: 698-706.

[8] MIHOLCA D L, CZIBULA G, CRIVEI L M. A new incremental relational association rules mining approach[J]. Procedia Computer Science, 2018, 126: 126-135.

[9] NGUYEN D, LUO W, PHUNG D, et al. LTARM: A novel temporal association rule mining method to understand toxicities in a routine cancer treatment[J]. Knowledge-Based Systems, 2018, 161: 313-328.

[10] ZHANG S C, ZHANG J L, CHEN F, et al. Negative Incremental Updating Algorithm for Maintaining Association rules[J]. Computer Science, 2005, 32(9): 153-155. (in Chinese)
张师超，张继连，陈峰，等. 负增量式关联规则更新算法[J]. 计算机科学，2005，32(9)：153-155.

[11] CAI J, XUE Y S, LIN L, et al. Updating Algorithm for Association Rules Based on Fully M ining Incremental Transactions[J]. Computer Science, 2007, 34(2): 220-222. (in Chinese)
蔡进，薛永生，林丽，等. 基于充分挖掘增量事务的关联规则更新算法[J]. 计算机科学，2007，34(2)：220-222.

[12] GAN W, LIN C W, FOURNIER-VIGER P, et al. Mining of frequent patterns with multiple minimum supports[J]. Engineering Applications of Artificial Intelligence, 2017, 60(C): 83-96.

[13] DARRAB S, ERGENC B. Vertical Pattern Mining Algorithm for Multiple Support Thresholds[J]. Procedia Computer Science, 2017, 112: 417-426.

[14] HU H W, CHANG H C, LIN W S, et al. An optimized frequent pattern mining algorithm with multiple minimum supports[C]// IEEE International Conference on Big Data. IEEE, 2017.

[15] DAHBI A, BALOUKI Y, GADI T. Using Multiple Minimum Support to Auto-adjust the Threshold of Support in Apriori Algorithm[C]// International Conference on Soft Computing and Pattern Recognition. Cham: Springer, 2017: 111-119.

[16] WENG C H. Identifying association rules of specific later-marketed products[J]. Applied Soft Computing, 2016, 38: 518-529.

[17] GAO J, SHI B L. Research on Fast Association Rule Mining Algorithm[J]. Computer Science, 2005, 32(3): 200-201. (in Chinese)
c高俊，施伯乐. 快速关联规则挖掘算法研究[J]. 计算机科学，2005，32(3)：200-201.

[18] LIU D, LIU Y, YIN Z. Fast algorithm for discovering maximum frequent itemsets of association rules[J]. Acta Scientiarium Naturalium Universitatis Jilinensis, 2004, 42(2): 212-215.

[19] ZAKI M J, GOUDA K. Fast vertical mining using diffsets[C]// Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2003: 326-335.

[20] FOURNIER-VIGER P, GOMARIZ A, SOLTANI A, et al. SPMF: OpenSource Data Mining Platform[EB/OL]. http://www. philippe-fournier-viger. com/spmf.