一种基于压缩矩阵的Apriori算法改进研究

罗 丹 李陶深
(广西大学计算机与电子信息学院 南宁 530004)

摘 要 针对已有基于矩阵的Apriori算法存在的问题,提出了一种改进的基于压缩矩阵的Apriori算法。算法进行了以下方面的改进:增加了两个数组,分别用于记录矩阵行与列中1的个数,使得算法在压缩矩阵时减少了扫描矩阵的次数;在压缩矩阵中,通过增加删除不能连接的项集和非频繁的项集的操作,使得矩阵压缩得更小,提高了空间效率;改变了删除事务列的条件和算法结束的条件,以减少挖掘结果的误差和算法循环的次数。算法性能分析和实验分析证明,改进后的算法能有效地挖掘频繁项集,并且比现有的算法具有更高的计算效率。

关键词 数据挖掘,频繁项集,Apriori算法,压缩矩阵
中图法分类号 TP311 **文献标识码** A

Research on Improved Apriori Algorithm Based on Compressed Matrix

LUO Dan LI Tao-shen
(School of Computer, Electronics and Information, Guangxi University, Nanning 530004, China)

Abstract Aiming at the deficiency of the existing Apriori algorithm, an improved Apriori algorithm based on compressed matrix called NCM_Apriori_1 was proposed. The improvements of this algorithm are as follows: (1) adding two arrays to record the counts of 1 in the row and column, so that the number of scanning the matrix can be reduced during compressing, (2) deleting the unnecessary itemsets which can't be connected as well as the infrequent ones in compressing matrix to minify the scale of matrix and improve space utilization, (3) changing the condition of deleting the unnecessary transactions to reduce the errors of the mine result, and changing the stopping condition to make the number of cycle decreased. Algorithm performance analysis and experiments results prove that the improved algorithm can mine frequent itemsets effectively and has better efficiency of computing than existing Apriori algorithms based on compressed matrix.

Keywords Data mining, Frequent itemsets, Apriori algorithm, Compressed matrix

1 引言

关联规则挖掘是数据挖掘领域中一个重要的研究方向,旨在挖掘事务数据库中有趣的关联。Agrawal等人提出的Apriori算法[1]是关联规则挖掘中最为经典的算法。然而,Apriori算法存在两个瓶颈:1)多次扫描事务数据库,需要很大的I/O负载;2)产生庞大的候选集,这是对运行时间和主存空间的挑战。

针对Apriori算法存在的产生候选项目集效率低和频繁扫描数据等缺点,人们提出了一些改进的算法。如基于模式增长的FP-Growth算法[2]、基于垂直型数据格式的Eclat算法[3]、基于Hash方法的DHP算法[4]、基于交集的Inter-Apriori算法[5]等等。一些研究人员提出了基于矩阵的Apriori算法[6-9],这类算法首先将事务数据库用矩阵表示出来,用"与运算"的方法代替了到数据库中去查找的算法,节省了计算频繁项集的时间,提高了计算的效率。文献[6]通过先找出最大频繁K项集,再找出其余的频繁项集来减少扫描候选项集的次数;文献[7]改变频繁1项集的排列从而减少了候选频繁项集的产生;文献[8]去除了大量冗余的非频繁项集,有效压缩了矩阵;文献[9]提出了一种基于压缩矩阵的Apriori算法CM_Apriori_1,算法只需扫描一遍数据库,压缩事务列,省去了生成候选集的步骤,直接通过矩阵的"与运算"产生频繁项集。但是基于矩阵的Apriori算法仍然存在以下问题:1)在运算过程中多次扫描矩阵,增加了运算的时间;2)压缩矩阵过程中,只压缩了事务或只压缩了项集,存储了很多与生成频繁项集无关的元素,压缩后的矩阵仍过于庞大,对算法空间造成较大压力;3)压缩事务列的条件删除了过多的边缘事务,使得生成的频繁项集有遗漏。

针对基于矩阵的Apriori算法存在的问题,本文提出了一种改进的基于压缩矩阵的Apriori算法。算法从准确而高效地挖掘出事务数据库中所有频繁项集的角度出发,通过减少扫描矩阵的次数、删除不能连接的项集和非频繁的项集、改变删除事务列条件和算法结束条件等措施,提高算法的计算效率。

到稿日期:2013-05-21 返修日期:2013-07-18 本文受国家自然科学基金项目(60973074)资助。
罗 丹(1988—),女,硕士生,主要研究方向为数据挖掘、云计算,E-mail:360081437@qq.com;李陶深(1957—),男,博士,教授,主要研究方向为分布式数据库、无线Mesh网络、云计算、网络与信息安全。

2 关联规则概念和算法原理

2.1 基本概念

关联规则挖掘问题的形式化描述如下：设 $I=\{I_1, I_2, \cdots, I_m\}$是项的集合，事务数据库是 $D=\{T_1, T_2, \cdots, T_n\}$，其中每个事务 T 是项的集合，使得 $T \subseteq I$，关联规则是形如 $A \Rightarrow B$ 的蕴含式，其中 $A \subset I, B \subset I$，并且 $A \cap B=\Phi$，定义支持度(*support*)为 D 中包含 $A \cup B$ 的百分比，则 $support(A \Rightarrow B)=P(A \cup B)$；置信度(*confidence*)为 D 中既包含 A 的事务也包含 B 的百分比，则 $confidence(A \Rightarrow B)=P(B|A)$。

项的集合称为项集，包含 k 个项的项集称为 k 项集。如果项集的出现频率大于或等于最小支持度与 D 中事务总数的乘积，则该项集满足最小支持度 *min_support*。若其满足最小支持度，则称它为频繁项集，频繁 k 项集的集合称为项集 L_k[9]。

同时满足最小支持度和最小置信度的规则称为强关联规则。关联规则挖掘问题就是在一个事务数据库中寻找强关联规则的过程，划分为两个子问题：(1)找出所有频繁项集；(2)由频繁项集产生强关联规则。本文重点研究的问题就是如何快速而高效地挖掘出事务数据库中的所有频繁项集。

2.2 相关性质与定理

性质 1 频繁项集的所有非空子集都必须也是频繁项集。

推论 1[10] X_k 是数据集 D 的频繁 k 项集，则 L_{k-1} 中包括 X_k 的 $(k-1)$项子集的个数一定是 k。

推论 2[11] 如果频繁 k 项集还能产生频繁 $(k+1)$项集，则频繁 k 项集中的个数必大于 k。

性质 2 非频繁项集的任一超集必定也是非频繁项集。

性质 3[12] 不包含任何频繁 k 项集的事务不可能包含任何频繁 $(k+1)$项集。

定理 1 如果数据库中某条事务的长度为 k，那么这条事务就不可能包含任何项数大于 k 的频繁项集。

定理 2[13] 在由 $(k-1)$项集生成 k 项集时，当 L_{k-1} 作自身连接时，若两个项集的前 $(k-2)$项不同，则放弃该两个项集的连接运算，因为产生的项集不是重复的就是非频繁项集。

推论 3[14] 将每个事务及事务中的项目集按照字典顺序排序。对于两个 $(k-1)$频繁项目集 I_x 和 I_y，如果 I_x 和 I_y 不能连接，则 I_x 和 I_y 之后的所有项目集都不需要进行连接判断。

3 改进算法 NCM_Apriori_1

3.1 算法的改进思想

(1)增加数组 m：记录每列 1 的个数；增加数组 n：统计每行 1 的加权个数，即每行按位与各列对应权值相乘后的和值。

(2)压缩矩阵：①扫描矩阵，根据定理 2 和推论 3，若一个项集不能与它相邻的项集进行连接运算，则删除该项集对应的行向量，对数组 m 的值进行相应的修改。②扫描数组 m，根据性质 3 和定理 1，若其值小于等于 1，则删除该列向量，对数组 n 的值进行相应的修改。③扫描数组 n，根据性质 2，若其值小于最小支持度计数，则删除该行向量，对数组 m 的值进行相应的修改。重复步骤②、③，直到矩阵已无变化则停止处理。剩下的行向量和列向量按照顺序组合成新的矩阵。

(3)生成频繁项集：清空数组 m 和 n，对可连接的项集对应的行按位采用“与运算”，并与 w 中对应权值相乘，其加权和为项集的支持度计数，其值若大于等于最小支持度计数，则保存该行向量，并将该向量按位累加到数组 m 中，对应的支持度计数存入 n 数组中，否则舍去。此时，保存下来的行向量所对应的项集为所求的频繁项集。

(4)根据推论 2，直到频繁 $(k-1)$项集个数，即矩阵的行数小于 k 时，可不用再求 k 频繁项集，则算法终止。

3.2 算法描述

算法 NCM_Apriori_1

```
输入：事务数据库 D，最小支持度 min_support
输出：所有频繁项集
方法：
/*扫描数据库，构建布尔矩阵 D，列代表“事务”，行代表“项”，矩阵中
 的每列压缩存储一条或多条事务信息，且无重复列。*/
for(i=0;i<M;i++){ //项数为 M
  for(j=0;j<N;j++){ //不重复事务数为 N
    if(Ii in Tj) then D[i][j]=1;
    else D[i][j]=0;}
}
for (i=0;i<N;i++)//对相同的事务计数，计数存入权值数组 w
  if(Ti 有重复的事务) then w[i]++;
L1=find_frequent_1_itemsets(D,min_support);
D1←D 删除不频繁的项所在行向量;
for(i=0;i<M1;i++){   //M1 为 D1 的行数
  for(j=i+1;j<M1;j++){
    for(q=0;q<N1;q++){   // N1 为 D1 的列数
      support_counts+=(D1[i][q]AND D1[j][q])×w[q];}
    if(support_counts≥min_support×事务数) then{
      add 该项集 to L2;
      将该项集对应行向量按位累加到 m;
      add support_counts to n;}
}}
D2←L2 所对应的行向量;
for(k=3;|Lk-1|≥k;k++){
  for each 项集 l in Lk-1
    if(l 不能与相邻的项集进行连接运算)then {
      按位修改 m 的值;
      delete l 所在行向量;
    }
  do{ for(i=0;i<Nk-1;i++)// Nk-1为 Dk-1的列数
      if(m[i]≤1) then{
        第 i 列 1 所在位置对应 n 的位置上的值-1;
        delete 第 i 列和 w[i];
    }
    for(i=0;i<Mk-1;i++)// Mk-1为 Dk-1的行数
      if(n[i]<min_support×事务数) then{
        按位修改 m 的值;
        delete 第 i 行;
      }
  }while(Dk-1变化了)
  D'k-1←Dk-1剩下的行和列按顺序排列;
  clear(m);clear(n);//清空数组 m 和 n
  for(i=0;i<M'k-1;i++){//M'k-1为 D'k-1的行数
    for(j=i+1;j<M'k-1;j++){
      if(项集 li 与项集 lj 可以连接) then{
```

```
        for(q=0;q<N'k-1;q++){
          support_counts+=(D'k-1[i][q] and D'k-1[j][q])×w
[q];}
        if(support_counts≥min_support×事务数) then{
          add 该项集 to Lk;
          将该项集对应行向量按位累加到 m;
          将 support_counts 存入 n;
    } } } }
  Dk←Lk 所对应的行向量;
}
```

3.3 算法复杂度

NCM_Apriori_1 算法复杂度用到的参数变量如表 1 所列。

表 1 参数列表

参数	参数说明	参数	参数说明
N	事务数	M	项数
K	最大频繁项数	T	平均事务长度
D_k	L_k 组成的矩阵	$N_{删}$	删除的列数
N_k	D_k 压缩后列数 (CM_Apriori_1)	N_k'	D_k 压缩后列数 (NCM_Apriori_1)
$L_{删1}$	不能连接的项集	$L_{删2}$	小于 min_support 的项集
L_k	频繁 k 项集	L_k'	压缩行后频繁 k 项集

扫描事务数据库 D,生成布尔矩阵及 w 数组,扫描时间为:$O(T\times N)$;生成 L_1,时间复杂度为:$O((M+1)\times N_1)$;生成 L_2,时间复杂度为:$O(\frac{|L_1|\times(|L_1|-1)}{2}\times N_1)$;当 $2\leqslant k\leqslant K-1$ 时,压缩矩阵,时间复杂度为:$O(|L_k|)+O(|L_{删1}|\times N_{k-1}')+O(N_{k-1}')+O(N_{删})+O(|L_k|-|L_{删1}|)+O(|L_{删2}|\times N_k')$,生成 L_{k+1} 后,时间复杂度为:$O(\frac{|L_k'|\times(|L_k'|-1)}{2}\times N_k')$。

(1)当频繁 K 项集个数小于 $(K+1)$ 时,算法总的时间复杂度为:

$$\begin{aligned}T_{NCMA1}=&O(T\times N)+O((M+1)\times N_1)+O\\&(\frac{|L_1|\times(|L_1|-1)}{2}\times N_1)+\sum_{2\leqslant k\leqslant K-1}[O(|L_k|)\\&+O(|L_{删1}|\times N_{k-1}')+O(N_{k-1}')+O(N_{删})+\\&O(|L_k|-|L_{删1}|)+O(|L_{删2}|\times N_k')+\\&O(\frac{|L_k'|\times(|L_k'|-1)}{2}\times N_k')]\end{aligned}\tag{1}$$

(2)当频繁 K 项集个数大于 K 时,算法总的时间复杂度为:

$$\begin{aligned}T_{NCMA2}=&O(T\times N)+O((M+1)\times N_1)+O\\&(\frac{|L_1|\times(|L_1|-1)}{2}\times N_1)+\sum_{2\leqslant k\leqslant K}[O(|L_k|)+\\&O(|L_{删1}|\times N_{k-1}')+O(N_{k-1}')+O(N_{删})+O\\&(|L_k|-|L_{删1}|)+O(|L_{删2}|\times N_k')+O\\&(\frac{|L_k'|\times(|L_k'|-1)}{2}\times N_k')]\end{aligned}\tag{2}$$

总的空间复杂度为:$S=O((|L_k|+2)\times(N_{k-1}'+1))$,当且仅当 $|L_k|$ 为最大时。

4 算法分析与对比实验

4.1 实例分析

某事务数据库中有 10 个事务,$D=\{T_1,T_2,\cdots,T_{10}\}$,对应的项目集 $I=\{I_1,I_2,I_3,I_4,I_5,I_6\}$,如表 2 所列。假设最小支持度 $min\_support=20\%$,则最小支持度计数为:$min\_support\times|D|=20\%\times10=2$。

表 2 事务数据库

TID	项 ID 的列表	TID	项 ID 的列表
T_1	I_1,I_2,I_4,I_5,I_6	T_6	I_2,I_3,I_5,I_6
T_2	I_2,I_3,I_5,I_6	T_7	I_3
T_3	I_2,I_3	T_8	I_2,I_4,I_5,I_6
T_4	I_3,I_4,I_5	T_9	I_3,I_4,I_5
T_5	I_3,I_5,I_6	T_{10}	I_5,I_6

(1)扫描事务数据库 D,生成布尔矩阵及 w 数组。因为事务 T_2 和 T_6 以及 T_4 和 T_9 含有相同项目,所以第 2、4 列的权值为 2,其余的权值均为 1。

$w=[1\ \ 2\ \ 1\ \ 2\ \ 1\ \ 1\ \ 1\ \ 1]$

$$D=\begin{matrix}I_1\\I_2\\I_3\\I_4\\I_5\\I_6\end{matrix}\begin{bmatrix}1&0&0&0&0&0&0&0\\1&1&1&0&0&0&1&0\\0&1&1&1&1&1&0&0\\1&0&0&1&0&0&1&0\\1&1&0&1&1&0&1&1\\1&1&0&0&1&0&1&1\end{bmatrix}$$

(2)生成 L_1。扫描矩阵 D 并计算所有项目的支持度计数,若 I_1 的支持度计数小于 2,则删除该行向量。将保存下来的行向量按顺序重新组合得到初始矩阵 D_1。

$w=[1\ \ 2\ \ 1\ \ 2\ \ 1\ \ 1\ \ 1\ \ 1]$

$$D_1=\begin{matrix}I_2\\I_3\\I_4\\I_5\\I_6\end{matrix}\begin{bmatrix}1&1&1&0&0&0&1&0\\0&1&1&1&1&1&0&0\\1&0&0&1&0&0&1&0\\1&1&0&1&1&0&1&1\\1&1&0&0&1&0&1&1\end{bmatrix}$$

(3)生成 L_2。对矩阵各行按位采用“与运算”,$I_2\wedge I_3=01100000$,$support\_count(I_2,I_3)=0\times1+1\times2+1\times1+0\times2+0\times1+0\times1+0\times1+0\times1=3$。项集$\{I_2,I_3\}$的支持度计数大于最小支持度计数,应保存该行向量。同理,项集$\{I_2I_4,I_2I_5,I_2I_6,I_3I_4,I_3I_5,I_3I_6,I_4I_5,I_4I_6,I_5I_6\}$对应的行向量应当保存,同时将行向量按位累加到数组 m,将支持度计数存入数组 n。所保存的行向量对应的项集即为 L_2,按顺序重新组合得到矩阵 D_2。

$w=[1\ \ 2\ \ 1\ \ 2\ \ 1\ \ 1\ \ 1\ \ 1]n$

$$D_2=\begin{matrix}I_2I_3\\I_2I_4\\I_2I_5\\I_2I_6\\I_3I_4\\I_3I_5\\I_3I_6\\I_4I_5\\I_4I_6\\I_5I_6\end{matrix}\begin{bmatrix}0&1&1&0&0&0&0&0\\1&0&0&0&0&0&1&0\\1&1&0&0&0&0&1&0\\1&1&0&0&0&0&1&0\\0&0&0&1&0&0&0&0\\0&1&0&1&1&0&0&0\\0&1&0&0&1&0&0&0\\1&0&0&1&0&0&1&0\\1&0&0&0&0&0&1&0\\1&1&0&0&1&0&1&1\end{bmatrix}\begin{matrix}3\\2\\4\\4\\2\\5\\3\\4\\2\\6\end{matrix}$$

$m=[6\ \ 6\ \ 1\ \ 3\ \ 3\ \ 0\ \ 6\ \ 1]$

(4)压缩矩阵。①扫描矩阵 D_2,删除不能与相邻项集连接的项集对应的行向量,删除项集 I_5I_6 所在的行,对数组 m 的值进行相应的修改。②扫描数组 m,删除结果小于等于 1 的列向量,删除了第 3、6、8 列,对数组 n 的值进行相应的修

改。③扫描数组 n，删除其值小于最小支持度计数的行向量，由此，无删除行。剩下的行向量和列向量按照顺序组合成矩阵 D_2'。

$w=[1\quad 2\quad 2\quad 1\quad 1]n$

$$D_2'=\begin{array}{c} \\ I_2I_3 \\ I_2I_4 \\ I_2I_5 \\ I_2I_6 \\ I_3I_4 \\ I_3I_5 \\ I_3I_6 \\ I_4I_5 \\ I_4I_6 \end{array}\begin{array}{|ccccc|c} \multicolumn{5}{c}{} & \\ 0 & 1 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 & 1 & 4 \\ 1 & 1 & 0 & 0 & 1 & 4 \\ 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 & 5 \\ 0 & 1 & 0 & 1 & 0 & 3 \\ 1 & 0 & 1 & 0 & 1 & 4 \\ 1 & 0 & 0 & 0 & 1 & 2 \end{array}$$

$m=[5\quad 5\quad 3\quad 2\quad 5]$

(5)生成 L_3。清空数组 m 和 n，对 D_2' 各行按位采用“与运算”，$I_2\wedge I_3\wedge I_4=00000$，$support\_count(I_2,I_3,I_4)=0\times1+0\times2+0\times2+0\times1+0\times1=0$。项集$\{I_2,I_3,I_4\}$的支持度计数小于最小支持度计数，应舍去该行向量。同理，项集$\{I_2I_3I_5,\ I_2I_3I_6,\ I_2I_4I_5,\ I_2I_4I_6,\ I_2I_5I_6,\ I_3I_4I_5,\ I_3I_5I_6,\ I_4I_5I_6\}$对应的行向量应当保存，同时将行向量按位累加到数组 m，将支持度计数放入数组 n，而项集$\{I_3,I_4,I_6\}$对应的行向量应舍去。所保存的行向量对应的项集即为 L_3，按顺序重新组合得到矩阵 D_3。

$w=[1\quad 2\quad 2\quad 1\quad 1]n$

$$D_3=\begin{array}{c} \\ I_2I_3I_5 \\ I_2I_3I_6 \\ I_2I_4I_5 \\ I_2I_4I_6 \\ I_2I_5I_6 \\ I_3I_4I_5 \\ I_3I_5I_6 \\ I_4I_5I_6 \end{array}\begin{array}{|ccccc|c} \multicolumn{5}{c}{} & \\ 0 & 1 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 & 1 & 4 \\ 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 & 0 & 3 \\ 1 & 0 & 0 & 0 & 1 & 2 \end{array}$$

$m=[4\quad 4\quad 1\quad 1\quad 4]$

(6)压缩矩阵。①扫描矩阵 D_3，删除不能与相邻项集连接的项集对应的行向量，删除项集$\{I_2I_5I_6,\ I_3I_4I_5,\ I_3I_5I_6,\ I_4I_5I_6\}$所在的行，对数组 m 的值进行相应的修改。②扫描数组 m，删除结果小于等于 1 的列向量，删除了第 3、4 列，对数组 n 的值进行相应的修改。③扫描数组 n，删除其值小于最小支持度计数的行向量，由此，无删除行。剩下的行向量和列向量按照顺序组合成矩阵 D_3'。

$w=[1\quad 2\quad 1]n$

$$D_3'=\begin{array}{c} \\ I_2I_3I_5 \\ I_2I_3I_6 \\ I_2I_4I_5 \\ I_2I_4I_6 \end{array}\begin{array}{|ccc|c} \multicolumn{3}{c}{} & \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 2 \end{array}$$

$m=[2\quad 2\quad 2]$

(7)生成 L_4。清空数组 m 和 n，对 D_3' 各行按位采用“与运算”，$I_2\wedge I_3\wedge I_5\wedge I_6=010$，$support\_count(I_2,I_3,I_5,I_6)=0\times1+1\times2+0\times1=2$。项集$\{I_2,I_3,I_5,I_6\}$的支持度计数等于最小支持度计数，应保存该行向量。同理，项集$\{I_2,I_4,I_5,I_6\}$对应的行向量应保存，同时将行向量按位累加到数组 m，将支持度计数放入数组 n。所保存的行向量所对应的项集即为 L_4，按顺序重新组合得到矩阵 D_4。

$w=[1\quad 2\quad 1]n$

$$D_4=\begin{array}{c} \\ I_2I_3I_5I_6 \\ I_2I_4I_5I_6 \end{array}\begin{array}{|ccc|c} \multicolumn{3}{c}{} & \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 2 \end{array}$$

$m=[1\quad 1\quad 1]$

(8)L_4 的个数为 2，小于 5，则不用再求 L_5，最大频繁项集为频繁 4 项集，算法终止。

4.2 性能分析

算法性能分析主要是与文献[9]提出的 CM_Apriori_1 算法进行比较。性能分析用到的参数变量如表 1 所列。其中 $N_1'=N_1$，当 $k\geqslant2$ 时，$N_k'\leqslant N_k$，$L_k'\leqslant L_k$，$N_{k-1}'-N_{删}=N_k'$，$|L_k'|+|L_{删1}|+|L_{删2}|=|L_k|$。

4.2.1 CM_Apriori_1 算法复杂度

从扫描事务数据库到生成 L_2 的时间复杂度与 NCM_Apriori_1 算法相同，此处不再赘述。当 $2\leqslant k\leqslant K-1$ 时，压缩矩阵，时间复杂度为：$O((|L_k|+1)\times N_{k-1})$；生成 L_{k+1} 后，时间复杂度为：$O(\frac{|L_k|\times(|L_k|-1)}{2}\times N_k)$。

(1)当频繁 K 项集个数等于 1 时，算法总的时间复杂度为：

$$T_{CMA1}=O(T\times N)+O((M+1)\times N_1)+O(\frac{|L_1|\times(|L_1|-1)}{2}\times N_1)+\sum_{2\leqslant k\leqslant K-1}[O((|L_k|+1)\times N_{k-1})+O(\frac{|L_k|\times(|L_k|-1)}{2}\times N_k)]\quad(3)$$

(2)当频繁 K 项集个数大于 1 时，算法总的时间复杂度为：

$$T_{CMA2}=O(T\times N)+O((M+1)\times N_1)+O(\frac{|L_1|\times(|L_1|-1)}{2}\times N_1)+\sum_{2\leqslant k\leqslant K}[O((|L_k|+1)\times N_{k-1})+O(\frac{|L_k|\times(|L_k|-1)}{2}\times N_k)]\quad(4)$$

总的空间复杂度为：$S=O((|L_k|+1)\times N_{k-1})$，当且仅当 $|L_k|$ 为最大时。

4.2.2 算法的比较

(1)当频繁 K 项集个数大于 K 时：

$$\begin{aligned}T_{CMA2}-T_{NCMA2}&=式(4)-式(2)\\&\geqslant\sum_{2\leqslant k\leqslant K}[O(|L_k'|\times(N_k'-2))+O((|L_k'|+|L_{删2}|-1)\times N_{删})+O(\frac{(|L_{删1}|+|L_{删2}|)^2}{2}\times N_k')+O(\frac{((2|L_k'|-1)\times N_k'-4)(|L_{删1}|+|L_{删2}|)}{2})+O(|L_{删1}|)]>0\end{aligned}$$

(2)当频繁 K 项集个数小于$(K+1)$且大于 1 时：

$$\begin{aligned}T_{CMA2}-T_{NCMA1}&=式(4)-式(1)\\&\geqslant\sum_{2\leqslant k\leqslant K-1}[O(|L_k'|\times(N_k'-2))+O((|L_k'|+|L_{删2}|-1)\times N_{删})+O(\frac{(|L_{删1}|+|L_{删2}|)^2}{2}\times N_k')+O(\frac{((2|L_k'|-1)\times N_k'-4)(|L_{删1}|+|L_{删2}|)}{2})\end{aligned}$$

$+O(|L_{删1}|)]+O((|L_K|+1)\times N_{K-1})+O$

$(\frac{|L_K|\times(|L_K|-1)}{2}\times N_K)>0$

当频繁 K 项集个数等于 1 时：

$T_{CMA1}-T_{NCMA1}$

$=$式(3)$-$式(1)

$\geqslant\sum_{2\leqslant k\leqslant K-1}[O(|L_k'|\times(N_k'-2))+O((|L_k'|+|L_{删2}|-1)\times N_{删})+O(\frac{(|L_{删1}|+|L_{删2}|)^2}{2}\times N_k')+O(\frac{((2|L_k'|-1)\times N_k'-4)(|L_{删1}|+|L_{删2}|)}{2})+O(|L_{删1}|)]>0$

由上述 3 种情况下的算法时间复杂度比较可知，改进后的 NCM_Apriori_1 算法比 CM_Apriori_1 算法在运行时间上有所减少，特别是在(2)情况下，由于算法终止条件的改变，NCM_Apriori_1 算法少循环一次，时间用得更少。

从空间复杂度上的比较可知，NCM_Apriori_1 算法比 CM_Apriori_1 算法增加了两个数组，在空间上相当于多了一行一列，通过增加的数组记录所需的值，使得算法在压缩矩阵的步骤减少了扫描矩阵的次数，所以，即使在最差的条件下，即当 $|L_{删1}|=0$，$|L_{删2}|=0$，$L_k'=L_k$，$N_k'=N_k$ 时，也能保证 NCM_Apriori_1 算法时间复杂度减少的幅度是大于 0 的。NCM_Apriori_1 算法在压缩矩阵时删去了不能连接的项集，使得矩阵空间的压缩速度更快。随着算法的进行，空间复杂度逐渐减小为 0。

4.3 实验结果分析

为了比较 NCM_Apriori_1 算法和 CM_Apriori_1 算法的性能，将两种算法在 4 组不同数据库下进行算法运行时间的对比实验。算法采用 JAVA 编程实现，开发工具是 Eclipse SDK。实验环境是一台 PC 机：CPU Inter(R) Core(TM) i5-3210M，2.50GHz；4GB 内存；64 位 Windows 7 操作系统。采用 IBM Quest Market-Basket Synthetic Data Generator 数据生成器来生成 3 组不同的数据库。分别通过不同项数、不同密集度、不同事务数、不同支持度进行对比实验，每组实验除了进行对比的参数是变化的，其余参数都是相同的。图 1 为两种算法在不同项数数据库中的运行时间对比图，实验中的事务数为 2000、平均事务长度比例为 0.6、支持度为 45%。图 2 为两种算法在不同密集度数据库中的运行时间对比图，实验中的事务数为 1000，项目数为 50，支持度为 30%。图 3 为两种算法在不同事务数数据库中的运行时间对比图，实验中的项目数为 40，平均事务长度为 24，支持度为 30%。图 4 为两种算法在不同支持度同一数据库中的运行时间对比图，实验中的事务数为 2000，项目数为 30，平均事务长度为 18。

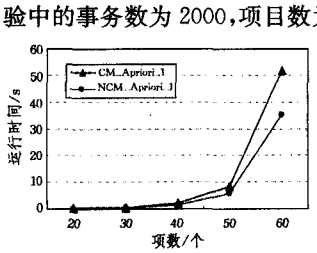

图 1 不同项数数据库运行时间对比图

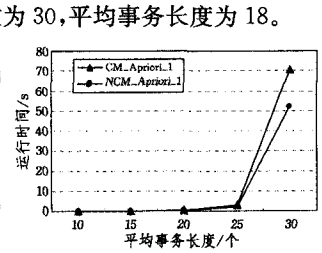

图 2 不同密集度数据库运行时间对比图

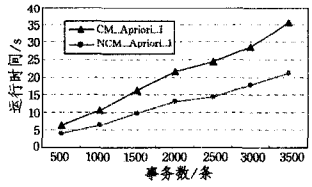

图 3 不同事务数数据库运行时间对比图

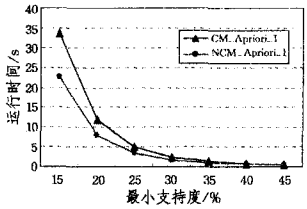

图 4 不同支持度同一数据库运行时间对比图

通过对 4 组实验结果进行对比分析可知：

(1)对于项数多、密集型的数据库，由于挖掘出的频繁项集的数目远大于事务数，形成的矩阵行数远超过列数，结合前面的时间复杂度对比分析可知，在这种情况下 NCM_Apriori_1 算法在压缩矩阵的步骤更凸显优势，因此，能比 CM_Apriori_1 算法表现出更好的性能。

(2)在事务数增多以及支持度下降的情况下，NCM_Apriori_1 算法也明显比 CM_Apriori_1 算法表现出更好的性能。

总的来说，实验结果与理论上算法复杂度的分析是一致的。由此可得出结论：NCM_Apriori_1 算法优于 CM_Apriori_1 算法，并且在项数多、事务数多的密集型数据库下，能表现出更好的执行效率。

结束语 本文提出了一种改进的基于压缩矩阵的频繁项集挖掘算法 NCM_Apriori_1，并通过算法性能分析和实验结果说明了该算法能有效地减少扫描矩阵的次数，压缩了矩阵规模，提高了生成频繁项集的效率，其性能优于文献[9]的 CM_Apriori_1 算法。但是 NCM_Apriori_1 算法仍然存在着不足，该算法是一个串行算法，对于海量数据库，由于产生的矩阵会给内存造成巨大的压力，因此不适用于处理海量数据。我们下一步的研究工作将考虑把 NCM_Apriori_1 算法的思想运用到并行 Apriori 算法中，使得新的算法能高效挖掘海量数据库。

参 考 文 献

[1] Agrawal R, Imielinaki T, Swami A. Mining association rules between sets items in large databases[C]// Proceedings of the ACM SIGMOD Conference on Management of Date. Washington, D. C, 1993: 207-216

[2] Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generations[C]// Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, USA, 2000: 1-12

[3] Zaki M J. Scalable Algorithms for Association Mining[J]. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(3): 372-390

[4] Park J S, Chen M-S, Yu P S. An effective hash-based algorithms for mining association rules[C]// Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. San Jose, California: ACM Press, 1995: 175-186

[5] 刘步中. 基于频繁项集挖掘算法的改进与研究[J]. 计算机应用研究, 2012, 29(2): 475-477

[6] Wang Feng, Li Yong-hua. An improved Apriori algorithm based on the matix[C]// 2008 International Seminat on Future Bio-Medical Information Engineering. Wuhan, 2008: 152-155

[7] 吕桃霞，刘培玉. 一种基于矩阵的强关联规则生成算法[J]. 计算机应用研究，2011，28(4)：1301-1303

[8] 张笑达，徐立臻. 一种改进的基于矩阵的频繁项集挖掘算法[J]. 计算机技术与发展，2010，20(4)：93-96

[9] 付沙，廖明华，宋丹. 基于压缩矩阵方式的 Apriori 改进算法[J]. 微电子学与计算机，2012，29(6)：28-32

[10] Khare N，Adlakha N，Pardasani K R. An Algorithm for Mining Multidimensional Association Rules using Boolean Matrix[C]// 2010 International Conference in Recent Trends in Information，Telecommunication and Computing. Kochi，Kerala，2010：95-99

[11] Liu Hui-zhen，Dai Shang-ping，Jiang Hong. Quantitative association rules mining algorithm based on matrix[C]// 2009 International Conference on Computational Intelligence and Software Engineering. Wuhan，2009：1-4

[12] 何丽. 基于规模压缩的关联规则数据挖掘算法研究[J]. 计算机科学，2007，34(9)：148-150

[13] 闫珍，皮德常，吴文昊. 高维稀疏数据频繁项集挖掘算法的研究[J]. 计算机科学，2011，38(6)：183-186

[14] 李瑞，康良玉，耿浩. 基于数组的关联规则算法的改进[J]. 科学技术与工程，2008，8(21)：5846-5849

(上接第 54 页)

表 5 UCI 数据集上实验参数

数据集	λ	SWLSSM (γ σ^2)	LSSVM(γ σ^2)	SVM(σ^2 C)
Credit	2^{-2}	(747.229 99.1677)	(8.6702 15.90)	(0.5 1000)
Cancer	2^{-5}	(747.229 99.1677)	(81.975 4.751)	(0.2 3000)
Ionosphere	2^{-5}	(14.876 7.4717)	(1.7253 21.893)	(0.5 3000)
Iris	2^{0}	(1.6396 1.5397)	(1.8295 1.4354)	(0.5 1000)
Liver	2^{-2}	(4627.87 339.238)	(278.8 242.25)	(5 3000)
Statolag	2^{10}	(0.801 12.4462)	(5.056 343.32)	(0.2 3000)
Balance	2^{2}	(3362.64 62.4015)	(331857 102)	(5 3000)
Wis	2^{-2}	(10.0714 104.853)	(180.0 412.07)	(0.2 3000)
Indata	2^{-2}	(0.4879 6.4259)	(0.32 13.835)	(0.3 5000)

表 6 UCI 数据集上 3 种方法的测试精度

数据集	SWLSSVM	LSSVM	SVM
Credit	89.156	85.627	88.888
Cancer	95.614	95.3216	95.609
Ionosphere	96.022	92.6136	87.73
Iris	100	98.7	100
Liver	69.364	66.438	68.076
Statolag	80.741	77.778	92.682
Balance	96.805	96.4856	93.617
Wis	97.193	96.1404	96.674
Indata	88	82	87.888

另外，本文从 ROC 曲线角度进一步分析，ROC 曲线是根据一系列不同的二分类方式，以真阳性率(灵敏度)为纵坐标、假阳性率(1-特异度)为横坐标绘制的曲线，ROC 曲线将灵敏度与特异性以图示方法结合在一起，可准确反映该方法特异性和敏感性的关系，曲线下的面积越大，判断价值越高；灵敏度：就是把实际为真值的判断为真值的概率。特异度：就是把实际为假值的判断为假值的概率。图 2 给出了 UCI 数据集中的 Ionosphere 数据的 ROC 曲线，从这个曲线中能看出 SWLSSVM 具有较高的精度。

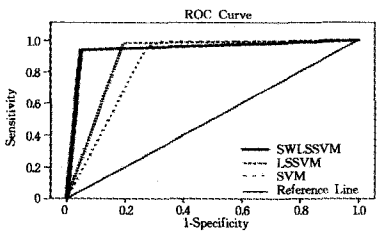

图 2 Ionosphere 数据集上的 ROC 曲线

SWLSSVM 的测试精度优于其他两种算法的原因如下：

(1)SVM 和 LSSVM 仅考虑了如何最大间隔地分离两类数据，而忽略了数据本身的结构信息。

(2)SWLSSVM 通过引入协方差矩阵考虑了样例的内部结构信息，在最大间隔的同时最小化类内紧实度；另外，又通过引入距离加权，降低了对异常点的敏感度。

结束语 为了提高分类器的泛化能力和减少异常点的影响，提出一种新的分类器——结构化加权最小二乘支持向量机(SWLSSVM)，其通过引入协方差矩阵将样例本身隐含的内部结构信息考虑到优化问题中，同时满足了最大化类间间隔和最小化类内紧实度，另外，通过加权减少了噪音点对分类器的影响，提高了分类器的泛化能力。在 UCI 数据库上的实验表明，SWLSSVM 能够更好地处理分类问题，具有较好的泛化能力。

参 考 文 献

[1] Suykens j AK，Vandewallej. Least squares support vector machines classifiers [J]. Neural Network Letters，1999，19(3)：293-300

[2] Liu Sheng，Yang Zhen. Real-time online forecasting model of ship rolling motion based on chaotic online LSSVM：Industrial Engineering and Engineering Management[C]// IEEE 18th Conference. Chicago：Illinois，2011：1732-1736

[3] Gao Song，Tang Yao-geng. LSSVM based missing data imputation in nuclear power plant's environmental radiation monitor sensor network：Advanced Computational Intelligence [C] // IEEE fifth Conference. USA：Alaska，2012：479-484

[4] 王熙照，崔芳芳，鲁淑霞. 密度加权近似支持向量机[J]. 计算机科学，2012，39(1)：182-184

[5] Belkin M，Niyogi P，Sindhwani V. Manifold regularization：A geometric framework for learning from examples [J]. Journal of Machine Learning Research，2006，7：2399-2434

[6] Yeung D，Wang D，Ng W，et al. Structured large margin machines：Sensitive to data distribution [J]. Mach. Learn.，2007，68：171-200

[7] Xue H，Chen S，Yang Q. Structural support vector machine [C]// Proc. 15th Int. Symp. Neural Netw. Beijing，2008，5263：501-511

[8] Wang Y，Chen S，Zhou Z. New semi-supervised classification method based on modified cluster assumption [J]. IEEE Trans. Neural Netw. Learn. Syst.，2012，23：689-702

[9] Huang K，Yang H，King I，et al. Learning large margin classifiers locally and globally[C]// Proc. 21st Int. Conf. Mach. Learn. Canada：Banff，2004：1-8

[10] Xue H，Chen S，Yang Q. Structural Regularized Support Vector Machine：A Framework for Structural Large Margin Classer[J]. Neural Networks，2011，22(4)：573-587

[11] Woodbury M A. Inverting modified matrices [M]. Princeton，NJ，Memo. Stat. Res. Group，Princeton Univ.，1950：42

[12] Chang C C，Lin C J. LIBSVM：a library for support vector machines[CP]. http：//www. csie. ntu. edu. tw/cjlin/libsvm，2013