基于 FC-tree 的频繁闭项目集挖掘算法*)

任永功[1] 张 亮[1] 付 玉[1] 吕君义[2]

(辽宁师范大学计算机与信息技术学院 大连 116029)[1] (辽河油田锦州工程技术处 凌海 121209)[2]

摘 要 目前提出的频繁项目集挖掘算法大多基于 Apriori 算法思想,但这类算法会产生巨大的候选集并且重复扫描数据库。本文针对这一问题,给出了一种基于 FC-tree 的频繁闭项目集挖掘算法 Max-FCIA,该算法将频繁项目集存储在哈希表中,节省了程序的搜索时间。此外,利用广度优先搜索和有效的剪枝策略,大大限制了候选项目集的生成,缩小了搜索空间从而提高了程序的性能。实验结果表明该算法是快速有效的。

关键词 频繁项目集,频繁闭项目集,最小频繁闭项目集,最大频繁闭项目集,FC-tree (频繁闭模式树)

Algorithm Based on FC-tree for Mining Frequent Closed Itemsets

REN Yong-gong[1] ZHANG Liang[1] FU Yu[1] LU Jun-yi[2]

(School of Computer and Information Technology, Liaohe Normal University, Dalian 116029, China)[1]

(Jinzhou Engineering & Technology Department of Liaoning Oilfield, Linhai 121209, China)[2]

Abstract Most mining algorithms of frequent Itemsets are based on Apriori. However, these algorithms make huge candidate itemsets and scan large database again and again. In order to solve this problem, an efficient algorithm called Max-FCIA based on FC-tree for mining frequent closed itemsets is proposed. The algorithm stores frequent Itemsets in hash table, which reduce the searching time. By breadth first search strategy and efficient pruning methods, making the frequent closed itemsets restrains the number of candidate, which saves the space and improves the efficiency of algorithm.

Keywords Frequent itemsets, Frequent closed itemsets, Minimum frequent closed itemsets, Maximal frequent closed itemsets, FC-tree (Frequent Pattern Tree)

1 引言

频繁项目集的挖掘在许多数据挖掘任务中起着重要作用,因此一直受到学术界的广泛关注,目前仍是国内外的一个研究热点。在众多频繁项目集的挖掘算法中,以 Agrawal 等人提出的 Apriori 算法最为著名,其后的算法大多数在 Apriori 算法基础之上进行改进,但该类算法存在如下不足:(1)算法需耗费大量的时间处理规模巨大的候选项目集。(2)算法需要重复扫描数据库 D。因此,针对 Apriori 算法产生大量频繁项目集这一问题,Pasquier 等人提出了频繁闭合模式的概念[1]。它唯一地决定所有频繁模式的准确支持率,并且尺寸比频繁模式集小几个数量级,其数量介于最大频繁项目集与频繁项目集之间,同时记录了所有频繁项目集的支持度。目前已经提出的频繁闭项目集挖掘算法有:A-Close,CLOSET,MAFIA,CHARM,CLOSE+,DCI-CLOSED 等。这些算法可以分为两类:一类是基于 Apriori 算法思想的频繁闭项目集挖掘算法,它们具有和 Apriori 算法同样的不足;另一类是不产生候选集而直接生成频繁闭项目集的挖掘算法,它们首先构造一棵树,然后在树上进行相关操作来发现数据库中的频繁闭项目集。

Pasquier 等人设计的算法 A-Close[1],以 Apriori[2] 为基础,采用自底向上、宽度优先的搜索策略,通过构造"生成子"集合,逐层求出所有频繁闭合模式。通过剪裁,减少了搜索范围,但没有解决宽度优先搜索算法内在的模式匹配的高 CPU 开销和重复扫描数据库的高 I/O 开销的问题。Pei 等人提出的算法 CLOSET[3] 以 FP-Growth 为基础,采用 FP-Tree 来表示模式支持集,通过深度优先搜索来挖掘频繁闭合模式。其困难是,递归构造"条件 FP-Tree"的 CPU 开销和存储开销很大。同时,CLOSET 采用分割法处理大型数据集,检查局部频繁闭合模式的全局闭合性和频繁性的代价非常高。Burdick 等人提出的深度优先搜索算法 MAFIA[4] 采用垂直二进制位图表示模式支持集,当项目平均支持率低于 1/32 时,存储效率远低于数组表示法,并且重建、计数和预处理开销很高。Zaki 等人提出的 CHARM[5] 是现有算法中性能最好的,它采用垂直格式的事务标识集来表示模式支持集,对项目集与事务标识集进行双向搜索,剪裁效率很高。CHARM 的困难在于事务标识集无论采用 tidset 还是 diffset 表示,存储开销都非常大,并且投影操作效率不高,对于密集型或存在特别长模式的数据集,CHARM 效率与可伸缩性较差。

本文系统地给出了频繁闭项目集及其挖掘算法,并依据频繁闭项目集这个概念构造 FC-tree,提出了一种基于 FC-tree 的频繁闭项目集挖掘的新方法。此方法在建树之前对事务数据库 D 进行预处理,减小了节点存储空间的浪费,同时将第一次扫描事务数据库的结果用树进行存储,减少了扫描

*)国家自然科学基金项目(60603047; 60703068),辽宁省自然科学基金,辽宁省教育厅高等学校科研基金,大连市优秀青年科技人才基金。

任永功 教授,博士,CCF 高级会员,研究方向为数据挖掘、图像处理技术等;**张 亮** 硕士研究生,研究方向为 Web 数据挖掘;**付 玉** 硕士研究生,研究方向为 Web 数据挖掘;**吕君义** 工程师,硕士,主要研究方向为石油地质开发、油田井下作业等。

事务数据库的次数，提高了程序的运行时间，并且以 FC-tree 为存储结构来组织频繁闭项目集，通过广度优先搜索和有效的剪枝策略，大大减少了候选项目集的生成，缩小了搜索空间，提高了程序的运行效率。实验证明此方法是可行有效的。

2 基本概念和问题描述

2.1 基本概念

设 $I=\{i_1,i_2,\cdots,i_m\}$ 表示 m 个不同项目的一个集合，事务数据库 D 包含 n 个事务，即 $D=\{T_1,T_2,\cdots,T_n\}$。项目集 $X\subseteq I$，X 在 D 中的支持数是指 D 中包含 X 的事务数，记为 count(X)。X 在 D 中的支持度是指 D 中包含 X 事务的百分比，记为 sup(X)。如果 X 的支持度不小于用户给定的最小支持度阈值 *minsup*，则称 X 为 D 中的频繁项目集。项目集中项目的个数称为项目集的维数或长度，频繁 1-项目集简称频繁项目。

定义 1 项目集 $X\subseteq I$，事务集 $T\subseteq D$，定义映射关系：

$t:2^I\to 2^T$，$t(X)=\{t\in T \mid$ 事务 t 支持项目集 $X\}$。

$i:2^T\to 2^I$，$i(T)=\{c\in I \mid T$ 中的任何事务均支持项目 $c\}$。

定义 2 设项目集 $X\subseteq I$，如果满足 $i(t(X))=X$，且 X 为频繁项目集，则称 X 为最大频繁项目集，简称频繁闭项目集。

定义 3 设频繁项目集 $X\subseteq I$，对于 X 的任意真子集 Y，均有 sup(X)< sup(Y)成立，则称 X 为最小频繁闭项目集。

例 1 设 $I=\{1,2,3,4,5,6,7,8\}$，事务数据库 D 如表 1 所示。

表 1 事务数据库 D

Tid	Itemset
T_1	1,2,3,6,8
T_2	2,4,5,7
T_3	1,3,5,6,8
T_4	1,2,3,5,6
T_5	1,2,5,6,7
T_6	2,3,4,5,6

如项目集 $X=\{1,2,3\}$，则 $t(X)=\{T_1,T_4\}$，因为 T_1，T_4 均支持项目集 X，且其他项目集均不支持 X。如事务集 $T=\{T_1,T_5\}$，则 $i(T)=\{1,2,6\}$，因为 T_1 和 T_5 同时支持项目集 $\{1,2,6\}$，且不同时支持其超集。

2.2 问题描述

由于最大频繁闭项目集(即频繁闭项目集)是频繁项目集的子集，因此可以把发现所有频繁项目集的问题转化成为发现所有最大频繁闭项目集的问题，且频繁闭项目集包含了所有频繁项目集的支持度。现将发现频繁闭项目集的步骤分为两步：第 1 步是求最小频繁闭项目集；第 2 步是根据最小频繁闭项目集确定最大频繁闭项目集(即频繁闭项目集)。但在进行第 1 步之前，需要对数据库进行扫描，将项目信息(数字、频繁度、对应的事务)用结构体数组存储，然后用满足条件的项目(1-频繁项目集)建立频繁闭模式树 FC-tree，利用它来存储和组织频繁闭项目集，进而寻找最小频繁闭项目集，最终确定最大频繁闭项目集。

3 频繁闭模式树(FC-tree)

定义 4(频繁闭模式树) 频繁闭模式树(FC-tree)是按频繁项目集的支持度构筑的多层结构树，树中节点的形式为 $h\{t\}$，其中 h 和 t 都是频繁项目集，即树上每个节点代表一个频繁项目集，同时节点还记录了用户通过同样的前缀路径到达该点的次数，即支持度。

FC-tree 的结构可描述为：

```
TYPE Nodeptr = ^Nodetype
Nodetype = RECORD
    CurrentItem_gat[ITEM_NUM]:Integer;
    SpareItem[ITEM_NUM]:Integer;
    ChildNum:Integer;
    ChildNode[ITEM_NUM],FatherNode:Nodeptr;
    CurrentNode_child:Integer;
    FrequentNum:Integer;
```

这里：CurrentItem_gat[*ITEM_NUM*]和 SpareItem[*ITEM_NUM*]分别用来保存当前项目集和候选项目。*ChildNum* 表示孩子个数。ChildNode[*ITEM_NUM*]和 FatherNode 分别表示指向其孩子节点和父亲节点的指针。*CurrentNode_child* 表示当前节点为其父亲节点的第几个孩子。*FrequentNum* 表示该项目集的支持度。*ITEM_NUM* 为频繁事务数据库 D 中的项目个数。

3.1 预处理用户事务数据库

由于事务数据库 D 是一个包含所有频繁项目和非频繁项目事务的集合，而非频繁项目在挖掘过程中对最终结果影响不大，因此我们可以在预处理事务数据库 D 时对非频繁项目进行删除，从而减少程序在运行时间和存储空间的浪费。

在对事务数据库 D 处理时，仅需扫描数据库 D 一次，产生频繁 1-项目集 E 及其支持度，按其支持度降序排列 E，生成频繁项目列表 $L_E=\{2,5,6,1,3\}$，同时对事务数据库 D 进行预处理，将非频繁项目从 D 中删除，从而得到频繁事务数据库 D'，如表 2 所示

表 2 事务数据库 D'

Tid	Itemset
T_1	1,2,3,6
T_2	2,5
T_3	1,3,5,6
T_4	1,2,3,5,6
T_5	1,2,5,6
T_6	2,3,5,6

3.2 生成 FC-tree

创建根节点记为{}的树，树中节点的形式为 $h\{t\}$，其中 h 和 t 均为频繁项目集。在建树算法中，采用先判断后建树的方式，避免生成不必要节点。当不对树做判断处理时，在树的第 i 层中，频繁项目集 h 中的项目的个数为 i(根节点为第 0 层)。但为了节省存储空间，避免生成不必要节点，在建树之前做一次支持度的判断，从父节点的候选频繁项目(t)中产生的频繁项目集(h)的支持度，只将支持度满足条件的节点建树，即选择支持度大于 50%的节点建树。例如：频繁项目集 $\{1,2,3\}$的支持数为 2，即支持度小于 50%，所以不满足建树条件，而频繁项目集$\{1,2,6\}$的支持数为 3，即支持度为 50%，因此为 FC-tree 的一个节点。

综上所述，建构 FC-tree 算法 1 如下。

算法 1 建构 FC-tree

输入：事务数据库 D′
输出：FC-tree.

```
1. 创建一个新节点 R 为根节点,其支持度为 0,SpareItem[ITEM_
   NUM]中用来保存事务数据库 D'中的成员
2. 目前节点指向 R
3. 频繁 n-项目集节点个数 SameKind_Num 为 1
4. CurrentChild = 0;
5. FOR i = 1 to Num_Item // Num_Item 为 D'中事务集中的事务个
   数
   1)While (1)
       a. FOR j = 1 to Num_Cand_Item //Num_ Cand_ Item 为目
          前节点(也称父亲节点)的候选项目个数
       a. 1 N = 从父亲节点的候选项目中产生的候选项目集的支
       持度。
       a. 2 if N >50% (此支持度可由用户自己定义)
            a. 2. 1. 创建一个新子节点 ChildeNode,将刚才计算的
               候选项目集保存到 CurrentItem_gat[ITEM_
               NUM],将该项目集的候选项目保存到 SpareItem
               [ITEM_NUM]
            a. 2. 2. FrequentNum = N。
            a. 2. 3. 新子节点的个数 CurrentChild + 1。
            Else
              a. 2. 4. Continue;
            EndIf
       EndFor
       b. SameKind_Num 减 1
       c. If SameKind_Num > 0
            c1. 目前节点指向下一个频繁 n-项目集节点
       Else
            c2. Break;
       EndIf
   EndWhile
   2) If CurrentChild > 0
        目前节点指向频繁 n+1-项目集的第一个节点
        SameKind_Num = CurrentChild;
      Else
        CurrentChild = 0;
      Break;
        EndIf
```

例 2　事务数据库 D' 如表 2 所示,建树过程如下:

1)由上面频繁项目列表 L_E={2,5,6,1,3}可知,最小频繁闭 1-项目集 $L_{c1}=C_1$={{1},{2},{3},{5},{6}};

2)利用 L_{c1} 生成 D' 中的候选最小频繁闭 2-项目集 C_2={{1,2},{1,3},…,{3,6},{5,6}};

3)扫描 D' 一次,计算 C_2 中各项目集的支持数(度);

4)删除 C_2 中支持度小于 50%的项目集,得 C_2={{1,2},{1,3},{1,5},{1,6},{2,3},{2,5},{2,6},{3,5},{3,6},{5,6}};

5)利用 C_2 生成 D' 中的候选最小频繁闭 3-项目集 C_3={{1,2,6},{1,3,6},{1,5,6},{2,3,6},{2,5,6},{3,5,6}};

6) 利用 C_3 生成 D' 中的候选最小频繁闭 4-项目集 C_4=Φ;

7) 由于 C_4=Φ,本次过程结束,用得到的候选最小频繁闭项目集 C_1,C_2,C_3 建树。

由此不难得到 FC-tree,如图 1 所示。

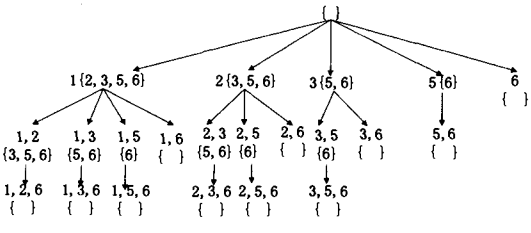

图 1　FC-tree

4　利用 FC-tree 挖掘频繁闭项目集

根据最小频繁闭项目集和最大频繁闭项目集的定义,可以得到最小频繁闭项目集和最大频繁闭项目集各自的约束条件,将这些约束条件作为上面得到的 FC-tree 的剪枝条件,那么经过广度优先遍历进行剪枝的 FC-tree,就将是需要挖掘的频繁闭项目集。

性质 1　设项目集 $X=\{x_1,x_2,\cdots,x_k\}$ 为最小频繁 k-项目集,则其任意$(k-1)$-项目子集均为最小频繁$(k-1)$-项目集。

推论　设 FC-tree 为包含所有最小频繁项目集的树,如果树中有子节点,则其父节点 Freque_father[i]也必为非频繁项目集。

证明:反证法。假设父节点 Freque_father[i]为频繁项目集,则根据性质 1 可知,其任意子集 Freque_child 也必为频繁项目集,并且 Freque_child[k]属于集合 *Freque_child*,所以 Freque_child[k]也必为频繁项目集,这与 Freque_child[k]为非频繁项目集相矛盾,所以假设不成立。结论得证。

由此,我们可以得到最小频繁闭项目集挖掘算法。

4.1　最小频繁闭项目集挖掘算法

算法思想:由最小频繁闭项目集的定义可知,树中各个频繁项目集的支持度要小于其任意子集的支持度,如果当前节点的支持度大于等于其父亲节点的支持度,则删除当前节点和其孩子节点。由于最小频繁闭项目集的任意子集也必为最小频繁闭项目集,因此利用 FC-tree 很容易对应其子集之间的关系,删除相关节点。例如:节点{1,6}的支持度等于其父节点{1}的支持度,所以删除节点{1,6}。由此可以得到最小频繁闭项目集的挖掘算法,具体描述见算法 2。

算法 2　最小频繁闭项目集挖掘算法

```
输入:频繁 n-项目集的个数和频繁 n-项目集的地址
输出:保留的频繁 n+1 项目集的地址
1. k = 0
2. For i = 1 to Frequent Num
   //Frequent Num 为 FC-tree 中频繁 n-项目集的个数
     For j = 1 to Children Num
       // Children Num 为频繁 n-项目集 Freque_father[i]中的孩子
   个数
       a. 用 Freque_child[k](频繁 n+1-项目集)保存频繁 n-项目集
          Freque_father[i]中的孩子结点
       b. k++
     EndFor
   EndFor
3. For i = 1 to k
   //k 为频繁 n+1-项目集的个数
     If Freque_child[i]的频繁度 = frequent_father;
     //frequent_father 为父亲结点的频繁度
       Delete_node(Freque_child[i]);
     Else
         For j = 其父亲结点的位置 to 最后一个频繁 n-项目集的位
             置
             If Freque_child[i]项目集所有成员 < Freque_father
               [j]项目集中的最小项目成员
                 Break;
             Else
                 If Freque_child[i]项目集最大成员 > Freque_fa-
                   ther[j] 项目集的最大成员 &&Freque_father
                   [j] 项目集⊆Freque_child[i]项目集 &&
                   Freque_child[i]的频繁度 = Freque_father[j]
                   的频繁度
                     a. Delete_node(Freque_child[i]);
                     b. Break;
                 EndIf
             EndIf
```

例 3　事务数据库 D' 如表 2 所示,则该事务数据库中的最小频繁闭项目集的挖掘过程如下:

1)从树根{}开始进行父子节点的支持度比较,由于 sup({1})= sup({1,6}),删除 C_2 中的频繁项目集{1,6},得最小频繁闭 2-项目集 L_{c2}={{1,2},{1,3},{1,5},{2,3},{2,5},{2,6},{3,5},{3,6},{5,6}};

2)同样删除 C_3 中的频繁项目集{1,2,6},{1,3,6},{1,5,6},{2,3,6},{3,5,6},可得 L_{c3}={{2,5,6}}。

3)D' 中最小频繁闭项目集为:{{1},{2},{3},{5},{6},{1,2},{1,3},{1,5},{2,3},{2,5},{2,6},{3,5},{3,6},{5,

6},{2,5,6}}。

由图1的FC-tree经过上面的过程,可以得到基于FC-tree的最小频繁闭项目集,如图2所示。

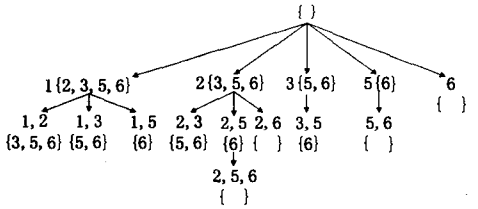

图2 基于FC-tree的最小频繁闭项目集

性质2 设 X 为最大频繁闭项目集,则存在最小频繁闭项目集 Y,使 $t(X)=t(Y)$成立。

性质3 设 X 为最大频繁闭项目集,则 $i(t(X))$为最大频繁闭项目集。

推论1 设 Y 为事务数据库 D 中的最小频繁闭项目集,则与 Y 对应的最大频繁闭项目集为 $i(t(Y))$。

推论2 设 MinFCI 为事务数据库 D 中最小频繁闭项目集的集合,则 Max$FCI=\{\ i(t(Y))\mid Y\in$ Min$FCI\ \}$。

性质4 设项目集 $Y\in$ MinFCI,$t(Y)=\{T_{Y1},T_{Y2},\cdots,T_{Ym}\}$,则 $i(t(Y))=\bigcap T_{Yj}$。

根据推论2和性质4,我们可以得到最大频繁闭项目集的挖掘算法。

4.2 频繁闭项目集挖掘算法 Max-FCIA

广度优先遍历上面得到的最小频繁闭项目集所对应的FC-tree,可以得到所有最小频繁闭项目集(MinFCI),由于 Max$FCI=\{\ i(t(Y))\mid Y\in$ Min$FCI\ \}$[7],即最小频繁闭项目集和最大频繁闭项目集存在对应关系,所以可以利用这些最小频繁闭项目集,进一步得到最大频繁闭项目集,即频繁项目集。最大频繁闭项目集的挖掘算法3描述如下:

算法3 最大频繁闭项目集挖掘算法 Max-FCIA(maximal frequent closed itemsets algorithm)

```
输入:树的根节点;事务数据库 D'
输出:D'中所有最大频繁闭项目集 MaxFCI
1. 通过树的根节点找到树种所有最小频繁闭项目集 Y,将其放到MinFCI中
2. Y ∈ MinFCI
3. For all Y ∈ MinFCI do
   a. MY = T;
   // MY用来存放最小频繁闭项目集 Y 所对应的最大频繁闭项目集,初始为全集 T
      //T 是树中频繁 1-项目集的项目成员集合
4. For all transaction t ∈ D' do
     For all Y ∈ MinFCI do
        a. If Y ⊆t then
             a. 1. MY = MY ∩ t;
5. MaxFCI = Φ ;
6. For all Y ∈ MinFCI do
a. MaxFCI= MaxFCI ∪ MY
```

例4 事务数据库 D' 如表2所示,最小频繁闭项目集为:{{1},{2},{3},{5},{6},{1,2},{1,3},{1,5},{2,3},{2,5},{2,6},{3,5},{5,6},{2,5,6}}。事务数据库中的最大频繁闭项目集的挖掘过程如下:

1)令 $M_{\{1\}}=M_{\{2\}}=M_{\{3\}}=M_{\{5\}}=M_{\{6\}}=M_{\{1,2\}}=M_{\{1,3\}}=M_{\{1,5\}}=M_{\{2,3\}}=M_{\{2,5\}}=M_{\{2,6\}}=M_{\{3,5\}}=M_{\{5,6\}}=M_{\{2,5,6\}}=\{1,2,3,5,6\}$;

2)扫描事务数据库 D' 一次;

3)对于 $T_1=\{1,2,3,6\}$,由于 T_1 支持最小频繁闭项目集中的{1},{2},{3},{6},{1,2},{1,3},{2,3}和{2,6},因此修改 $M_{\{1\}},M_{\{2\}},M_{\{3\}},M_{\{6\}},M_{\{1,2\}},M_{\{1,3\}},M_{\{2,3\}}$ 和 $M_{\{2,6\}}$,修改指令为 $M_Y=M_Y\cap\{1,2,3,6\}$,如 $M_{\{1\}}=\{1,2,3,5,6\}\cap\{1,2,3,6\}=\{1,2,3,6\}$,$M_{\{5\}},M_{\{1,5\}},M_{\{2,5\}},M_{\{3,5\}},M_{\{5,6\}}$ 和 $M_{\{2,5,6\}}$ 不变。

4)对于 $T_2=\{2,5\}$,由于 T_2 支持最小频繁闭项目集中的{2},{5}和{2,5},修改 $M_{\{2\}},M_{\{5\}}$ 和 $M_{\{2,5\}}$,其他不变。

5)对于 $T_3=\{1,3,5,6\}$,由于 T_3 支持最小频繁闭项目集中的{1},{3},{5},{6},{1,3},{1,5},{3,5}和{5,6},修改 $M_{\{1\}},M_{\{3\}},M_{\{5\}},M_{\{6\}},M_{\{1,3\}},M_{\{1,5\}},M_{\{3,5\}}$ 和 $M_{\{5,6\}}$,如 $M_{\{1\}}=\{1,2,3,6\}\cap\{1,3,5,6\}=\{1,3,6\}$,其他不变。

6)对于 $T_4=\{1,2,3,5,6\}$,由于 T_4 支持最小频繁闭项目集中的{1},{2},{3},…,{5,6}和{2,5,6},修改 $M_{\{1\}},M_{\{2\}},M_{\{3\}},\cdots,M_{\{5,6\}}$ 和 $M_{\{2,5,6\}}$,如 $M_{\{1\}}=\{1,3,6\}\cap\{1,2,3,5,6\}=\{1,3,6\}$,其他不变。

7)对于 $T_5=\{1,2,5,6\}$,由于 T_5 支持最小频繁闭项目集中{1},{2},{5},{6},{1,2},{1,5},{2,5},{2,6},{5,6}和{2,5,6},修改 $M_{\{1\}},M_{\{2\}},M_{\{5\}},M_{\{6\}},M_{\{1,2\}},M_{\{1,5\}},M_{\{2,5\}},M_{\{2,6\}},M_{\{5,6\}}$ 和 $M_{\{2,5,6\}}$,如 $M_{\{1\}}=\{1,3,6\}\cap\{1,2,5,6\}=\{1,6\}$,其他不变。

8)对于 $T_6=\{2,3,5,6\}$,由于 T_6 支持最小频繁闭项目集中的{2},{3},{5},{6},{2,3},{2,5},{2,6},{3,5},{5,6}和{2,5,6},修改 $M_{\{2\}},M_{\{3\}},M_{\{5\}},M_{\{6\}},M_{\{2,3\}},M_{\{2,5\}},M_{\{2,6\}},M_{\{3,5\}},M_{\{5,6\}}$ 和 $M_{\{2,5,6\}}$,其他不变。

9)因此事务数据库 D' 的最大频繁闭项目集:Max$FCI=M_{\{1\}}\cup M_{\{2\}}\cup M_{\{3\}}\cup M_{\{5\}}\cup M_{\{6\}}\cup M_{\{1,2\}}\cup M_{\{1,3\}}\cup M_{\{1,5\}}\cup M_{\{2,3\}}\cup M_{\{2,5\}}\cup M_{\{2,6\}}\cup M_{\{3,5\}}\cup M_{\{5,6\}}\cup M_{\{2,5,6\}}=$ {{2},{5},{6},{1,6},{2,5},{2,6},{5,6},{1,2,6},{1,3,6},{1,5,6},{2,3,6},{2,5,6},{3,5,6}}。

5 算法分析与比较

将 Max-FCIA 算法与 Max-Mine[6] 算法比较,可以得到下面的结论:

1)Max-Mine 算法中,扫描事务数据库 D,而 Max-FCIA 算法扫描事务数据库 D',由于 $D'\in D$,因此在扫描用户事务数据库时,Max-FCIA 算法比 Max-Mine 算法有更小的搜索范围。

2)在 Max-FCIA 算法中采用先判断后建树的处理方法,大大减少了内存的存储空间。而 Max-Mine 算法以所有频繁项目节点建树,浪费了不必要的存储空间。以本文事务数据库为示例,对 Max-FCIA 算法和 Max-Mine 算法做存储空间的比较,具体见图3。

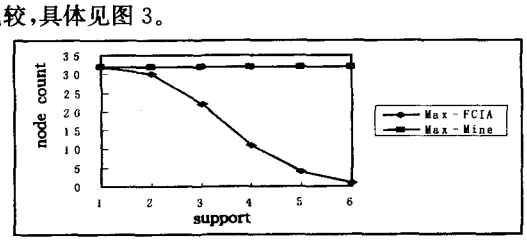

图3 存储空间对比

3)在 Max-FCIA 算法利用多叉树结构对候选项目集进行存储,好处在于可以以指数级减少候选项目集数量,可更快地得到最小频繁闭项目集。例如:假设 i 为树的层数(根节点的层数为0),则 Max-Mine 算法中候选项目集个数为 2^i,而 Max-FCIA 算法中候选项目集个数小于等于 2^i,并且如果某一节点从候选项目集中删除,则其子孙节点也必不满足条件(最小频繁闭项目集约束条件,见文献[7]定理1),会同时被

(下转第164页)

(2) 比较 DBCI 与 RBC 在 12 个数据集上的标准离差可以发现,在 7 个数据集上 DBC 的标准离差低于 RBC,在 1 个数据集上二者的标准离差相同,在其余 4 个数据集上,DBC 的标准离差高于 RBC,而 DBCI 在 12 个数据集上的标准离差的平均值也明显比 RBC 的要低。因此,总体来说 DBCI 的分类性能比 RBC 的更加稳定。

(3)从 DBCI 和 RBC 的运行时间来看,在所有实验数据集中的 9 个数据集上,DBCI 的运行时间少于 RBC 的运行时间,在 1 个数据集上二者的运行时间相同,只在其余 2 个数据集上 DBCI 的运行时间略多于 RBC 的运行时间。而 DBCI 在 12 个数据集上的平均运行时间也明显比 RBC 的要少。因此,从总体来看 DBCI 的运行效率明显高于 RBC。

结束语 分类是模式识别、机器学习以及数据挖掘中一个基本而又重要的问题。虽然大量有效的分类算法应运而生,但由于处理不完整数据的复杂性,以往的分类器大都是针对完整数据的。然而,实际中的数据通常是不完整的。因此,对不完整数据分类器的研究具有重要的意义。

本文通过分析已有的在分类过程中处理不完整数据的方法,提出了一种不完整数据分类器:DBCI。为验证 DBCI 的有效性,将它与分类效果显著的不完整数据分类器 RBC 进行了实验比较。在 12 个标准的不完整数据集上的实验结果显示,DBCI 的分类准确率与 RBC 的分类准确率大致相当。但 DBCI 的运行效率明显高于 RBC,而且其分类性能比 RBC 更加稳定。

参 考 文 献

[1] Quinlan J R. C4. 5:Programs for Machine Learning [M]. San Francisco:Morgan Kaufmann,1993

[2] Kohavi R, Becker B, Sommerfield D. Improving simple Bayes [C]// M. van Someren, G. Widmer, eds. Poster Papers of the ECML-97. Prague:Charles University,1997:78-87

[3] Friedman N,Geiger D,Goldszmidt M. Bayesian network classifiers [J]. Machine Learning,1997,29 (2/3):131-163

[4] Lauritzen S L. The EM algorithm for graphical association models with missing data [J]. Computational Statistics and Data Analysis,1995,19(2):191-201

[5] Russell S,Binder J,Koller D,et al. Local learning in probabilistic networks with hidden variables[C]// Proc. of IIJCAI-95. San Francisco:Morgan Kaufmann,1995:1146-1151

[6] Geman S, Geman D. Stochastic relaxation, Gibbs distributions andthe Bayesian restoration of images [J]. IEEE Trans on Pattern Analysis and Machine Intelligence,1984,6(6):721-741

[7] Little R J A,Rubin D B. Statistical Analysis with Missing Data [M]. New York:Wiley,1987

[8] Spiegelhalter D J,Cowell R G. Learning in probabilistic expert systems[C]// J Bernardo, J Berger, A PDawid, eds. Bayesian Statistics 4,Oxford:Oxford University Press,1992:447- 466

[9] Ramoni M,Sebastiani P. Robust Bayes classifiers [J]. Artificial Intelligence,2001,125(1/2):209-226

[10] Blake C,Keogh E,Merz C J. UCI Repository of machine learning databases [OL]. Department of Information and Computer Sciences,University of California,Irvine. http:// www. ics. uci. edu/~mlearn /MLRepository. html],1998

[11] Weka: Data Mining Software in Java [OL]. http://www. cs. waikato. ac. nz/ ml/weak,2007

(上接第 152 页)

从 FC-tree 中删除,这样可以以 2 的指数级减少搜索节点。如本文图 1 所示,如果节点{2}从树中删除,则其后继节点{2,3},{2,5},{2,6},{2,3,5},{2,3,6},{2,5,6},{2,3,5,6}也必不满足最小频繁闭项目集约束条件,从而应被剪枝。

为了进一步验证算法 Max-FCIA 的优越性,我们用 VC++ 6.0 在内存 512M、操作系统为 Windows XP 的机器上实现了 Max-FCIA 算法和 Max-Mine 算法。Max-Mine 算法是一种基于树型结构和 Apriori 算法思想的最大频繁项目集挖掘算法,它比 Apriori 算法有很大优势。本实验使用了 chess 数据集中的 3196 条数据,实验结果如图 4 所示,图 4 表明 Max-FCIA 算法是有效的。

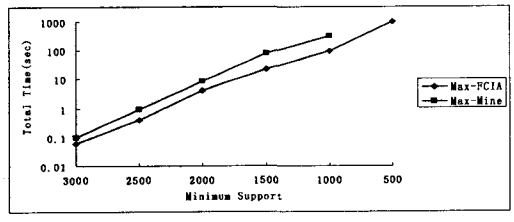

图 4 算法执行时间比较

结束语 频繁闭项目集提供了事务数据库的一个最小描述,其数量介于最大频繁项目集和频繁项目集之间,同时记录了所有频繁项目集的支持度,因此发现频繁闭项目集对数据挖掘具有十分重要的意义。本文系统地给出了频繁闭项目集及其存储结构 FC-tree 和挖掘算法 Max-FCIA,并对一些关键技术进行了改进,实验表明该算法是有效可行的。今后研究工作的重点将放在频繁闭项目集挖掘算法改进和增量式更新方面。

参 考 文 献

[1] Pasquier N, Bastide Y, Taouil R, et al. Discovering frequent closed itemsets for association rules// Beeri C, et al. , eds. Proc. of the 7th Int'l. Conf. on Database Theory. Jerusalem: Springer-Verlag,1999:398-416

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules// Beeri C, et al. , eds. Proc. of the 20th Int'l. Conf. on Very Large Databases. Santiago: Morgan Kaufmann Publishers, 1994:487-499

[3] Pei J, Han J, Mao R. CLOSET: An efficient algorithm for mining frequent closed itemsets// Gunopulos D, et al. , eds. Proc. of the 2000 ACM SIGMOD Int' l. Workshop on Data Mining and Knowledge Discovery. Dallas: ACM Press,2000:21-30

[4] Burdick D, Calimlim M, Gehrke J. MAFIA: A maximal frequent itemset algorithm for transactional databases// Georgakopoulos D, et al. , eds. Proc. of the 17th Int'l. Conf. on Data Engineering. Heidelberg: IEEE Press,2001:443-452

[5] Zaki M J, Hsiao C J. CHARM: An efficient algorithm for closed itemset mining// Grossman R, et al. , eds. Proc. of the 2nd SIAM Int'l. Conf. on Data Mining. Arlington: SIAM,2002:12-28

[6] 朱玉全,杨鹤标,孙蕾. 数据挖掘技术[M]. 南京:东南大学出版社,2006

[7] 朱玉全,宋余庆. 频繁闭项目集挖掘算法研究[J]. 计算机研究与发展,2007,44(7):1177-1183