文章编号: 1001-0920(2004)08-0867-05

一种快速的自适应频繁模式挖掘方法

叶飞跃[1,2], 王建东[1], 庄　毅[1]
(1. 南京航空航天大学 信息科学与技术学院, 江苏 南京 210016;
2. 江苏技术师范学院 计算机科学与技术系, 江苏 常州 213001)

摘　要: 提出一种自适应的频繁模式挖掘算法: AD-Mine 算法. 该算法采用超结构, 根据计算机可用内存自动确定一次性产生超结构的大小, 能够自动适应各类不同特性的数据, 进行高效率的频繁模式挖掘工作. 同时提出了一种能够有效地减少扫描记录数的新颖的数据库划分方法.

关键词: 数据挖掘; 频繁模式; 划分数据库; 自适应
中图分类号: TP311.133　　**文献标识码**: A

Adaptive fast algorithm for mining frequent itemset

YE Fei-yue[1,2], *WANG Jian-dong*[1], *ZHUANG Yi*[1]
(1. College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China; 2. Department of Computer Science and Technology, Jiangsu Teachers College of Technology, Changzhou 213001, China. Correspondent: WANG Jian-dong, E-mail: jdwang@jlonline.com)

Abstract: An adaptive algorithm, AD-Mine, is put forward to mine frequent patterns. The algorithm can fit different conditions and achieve good performance. A alterable hyper-structure is used in AD-Mine. The algorithm can automatically adjust the size of the hyper-structure according to the memory available for accommodating different cases of data and mine frequent patterns effectively. A method of partitioning database is brought forward to reduce the counts of scanning records.

Key words: data mining; frequent pattern; partition database; adaptive

1 引　　言

对于集中式数据库中频繁模式的挖掘方法, 目前主要分为以下两大类:

第 1 类是 Apriori[1] 类算法, 包括 Apriori, AprioriHybrid[1], DHP[2] 和 DCP[3] 等算法. 这些算法均基于产生候选项集, 其缺点是需要不断产生候选项集, 并不断地进行数据库的扫描, 挖掘 k-项频繁项集需扫描 k 遍数据库, 而对数据库的扫描次数是影响挖掘速度的主要因素.

第 2 类方法是不需要产生候选项集的挖掘算法, 该类算法的代表有 FP-Tree 算法[4] 和 H-Mine[5] 算法等. 这类算法的优点是不需要产生候选项集, 在内存能满足要求的情况下只需 2 次扫描数据库, 与 Apriori 类算法相比, 具有较快的挖掘速度. 但如果内存无法满足要求, 这类算法便变得相对复杂, 而且 FP-Tree 算法在短模式或数据库密度较低的情况下, 其挖掘效率相对较低[5]. H-Mine 算法是基于某种超结构的算法, 数据库中的每条纪录作为超结构中的一个条目, 数据库越大超结构也越大, 因此 H-Mine 对于大数据库挖掘是比较复杂的.

本文提出一种基于超结构的频繁模式挖掘算法, 即AD-Mine算法. 该算法能够针对不同数据库

收稿日期: 2003-08-26; **修回日期**: 2003-11-12
基金项目: 江苏省自然科学基金资助项目(BK2002091); 江苏省高校自然科学研究计划项目(03KJD110089).
作者简介: 叶飞跃(1960—), 男, 浙江东阳人, 副教授, 博士生, 从事数据挖掘、电子商务等研究; 王建东(1945—), 男, 江苏沭阳人, 教授, 博士生导师, 从事数据挖掘、机器学习与知识工程等研究.

的数据特性, 自适应快速地挖掘频繁模式, 而且对内存的需求上限只与频繁 1-项集中的模式长度有关, 而与数据库中交易数的多少无关. 该算法能够自动适应数据库的特性, 调整计算过程, 具有很好的适应性和效率.

2 问题定义

定义 1 设 $I = \{i_1, i_2, \cdots, i_m\}$ 是项的集合, 项集 X 是 I 的子集, 即 $X \subseteq I, X = \{i_k, i_{k+j}, \cdots, i_n\}$. 这里 $k, k+j, n$ 等均为项编号 $(k \geq 1, n \leq m, 1 \leq j < n-k)$, 且从小到大排列. 为了简单, 一个项集定义为 $X = i_k i_{k+j} \cdots i_n$.

定义 2 设 $\mathrm{ItemNum}(i_k) = k$, 则最小项编号 $\mathrm{MinNum}(X) = \mathrm{MinNum}(k, k+j, \cdots, n) = k$, 最大项编号 $\mathrm{MaxNum}(X) = \mathrm{MaxNum}(k, k+j, \cdots, n) = n$.

引理 1 设项集的集合 $\{X\} = \{X_1 = \{i_{k_1}, i_{(k+j)_1}, \cdots, i_{n_1}\}, \cdots, X_p = \{i_{k_p}, i_{(k+j)_p}, \cdots, i_{n_p}\}\}$, 则根据定义 2, 项集的集合 $\{X\}$ 的最小编号 $\mathrm{MinNum}(\{X\}) = \mathrm{MinNum}(k_1, k_2, \cdots, k_p)$, 项集的集合 $\{X\}$ 的最大项编号 $\mathrm{MaxNum}(\{X\}) = \mathrm{MaxNum}(n_1, n_2, \cdots, n_p)$.

定义 3 $|X|$ 表示项集 X 包含的项数, $|\{X\}|$ 表示项集的集合 $\{X\}$ 中的项集数.

引理 2 集合 $\{X\}$ 中可能出现的项集的最大项数 $m = (\mathrm{MaxNum}(\{X\}) - \mathrm{MinNum}(\{X\})) + 1$.

3 超结构的构建

3.1 超结构头表的构造

超结构的头表包含 2 个域, 即项数域和指针域, 其指针将指向具有该项数的一个哈希链结构. 超结构如图 1 所示. 其头表的构造采用动态方式, 并根据计算机可用内存情况自动调整构建头表的大小.

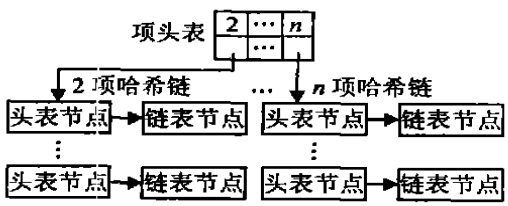

图 1 超结构示意图

3.2 哈希链的构造

3.2.1 链地址函数的构造

不难求得理论上最多可能出现的 $|X| \geq 2$ 的项集数为 $p = 2^m - (m+1)$, 而在实际数据库中容易得到最大项数 $\mathrm{Max}(|X|)$. 假设 $q = \mathrm{Max}(|X|)$, 则理论上计算的最多可能出现的 $|X| \geq 2$ 的项集数为 $p_q = 2^q - (q+1)$. 由于真正出现的项数与频繁 1-项集中项的密度有关, 实际出现的项数将远小于上述计算的 p_q 值, 则根据频繁 1-项集中项的密度情况对其调整后的总项数为 p'_q.

设某 1-项集 $X = i_k i_{k+1} \cdots i_n$, 项编号集合 $B = \{k, k+j, \cdots, n\}$, 其某一子集 $X' = i_{k'} i_{k'+j'} \cdots i_{n'}$, 项编号集合 $B' = \{k', k'+j', \cdots, n'\}$, 则采用除留余数法的哈希函数为

$$h(k', k'+j', \cdots, n') = \left(\sum_{i=k'}^{n'} (2i-1) z_i\right) \bmod p'' \tag{1}$$

式中: 当 $i \in B'$ 时, $z_i = i$, 否则 $z_i = 0$; p'' 可视情况分别取 p, p_q 或 p'_q.

3.2.2 哈希链结构

哈希链结构由哈希链头表和头表节点所指向的链表节点组成, 哈希链头表及链表节点结构如图 2 和图 3 所示. 图中: "链址"由哈希函数 $h(k', k'+j', \cdots, n')$ 计算得到, 头表中的"计数"为该头表节点所连结的所有链表节点中的计数之和, "指针 1"指向下一头表节点, "指针 2"指向相关的链表节点; 链表节点中的"计数"为该链表节点中模式累计出现的次数; 链表节点中的模式即为交易项的集合或项编号的集合.

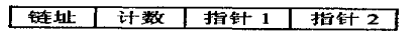

图 2 头表节点结构

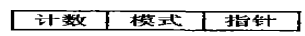

图 3 链表节点结构

3.3 哈希链的构建

为了在数据挖掘中节省空间开销, 本文采用动态方式构造链地址, 即对扫描中发现的模式构造链地址结构, 所有具有相同项数的哈希链连接到具有同一项数超链头表指针上. 在构建哈希链时, 频繁1-项集的数据库的每条交易, 一次性调入内存, 在内存中产生其所有的子集, 并将其保存到具有相同 $|X|$ 值的哈希链中, 如该项集已存在, 则只需对其相应的"计数"加 1 即可.

3.4 超结构内存需求计算

设 b_1 为超结构中每个项头表节点所占的字节数, b_2 为哈希链中每一头表节点的字节数, k-项模式 X^k 的链表节点的字节数为 $b_3 = a + k \times d$. 这里 a 和 d 是常数, a 是每个链表节点除存储模式 X^k 外的字节数, d 表示项模式 X^k 中每一项的字节数. 再设 i-项 $(i = 1, \cdots, m)$ 的项密度调整系数为 β_i, 设最大项数

为 y, 则构建项集超结构的字节数计算如下:

1) 设 b'_{1j} 是包含项数为 $2 \leq |X| \leq j$ 的项集的项头表的总字节数, 则有 $b'_{1j} = b_1 \times (j - 1)$.

2) 设 b'_{2j} 是包含项数为 $2 \leq |X| \leq j$ 的项集哈希链头表的总字节数, 则有

$$b'_{2j} = b_2 \times \left(\left[\beta_2 \times \left[\frac{y}{2} \right] \right] + \left[\beta_3 \times \left[\frac{y}{3} \right] \right] + \cdots + \left[\beta_j \times \left[\frac{y}{j} \right] \right] \right).$$

3) 包含项数为 $2 \leq |X| \leq j$ 的项集的哈希链链表节点的总字节数为

$$b'_{3j} = (a + 2 \times d) \times \left[\beta_2 \times \left[\frac{y}{2} \right] \right] + (a + 3 \times d) \times \left[\beta_3 \times \left[\frac{y}{3} \right] \right] + \cdots + (a + j \times d) \times \left[\beta_j \times \left[\frac{y}{j} \right] \right].$$

4) 求得前 j- 项的字节数之和为 $b_{yj} = b'_{1j} + b'_{2j} + b'_{3j}$. 这里: 当 y 和 j 均取 m 时, 即为最大项数为 m 的全部 m - 项字节数之和, 以 b_{mm} 表示; 当 y 和 j 分别取 m 和 i 时, 即为最大项数为 m 的前 i 项字节数之和, 以 b_{mi} 表示; 当 y 和 j 均取 q 时, 即为最大项为 q 的全部 q 项字节数之和, 以 b_{qq} 表示

4　划分数据库

设频繁项投影数据库为 TDB, 投影数据库中的每一投影为一个项集 X, 有 $1 \leq |X| \leq m'$, 则将投影数据库 TDB 按 $|X|$ 的值划分成 m' 部分, 即划分成 $D_1, \cdots, D_i, \cdots, D_{m'}$, 这里 $D_i (1 \leq i \leq m')$ 中只有 $|X| = i$ 的项集. 如果内存不能满足构建超结构的需要, 则将每一个 D_i 划分成 n 部分, 即对于具有相同 $|X| = i$ 的项集再划分成 n 部分, 形成具有 m' 行 n 列逻辑上不重叠的子数据库矩阵 Y, 即

$$Y = \begin{bmatrix} D_{11} & D_{12} & \cdots & D_{1n} \\ D_{21} & D_{22} & \cdots & D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ D_{m'1} & D_{m'2} & \cdots & D_{m'n} \end{bmatrix}.$$

5　自适应快速挖掘频繁模式算法

从第 4 节超结构对计算机内存的需求可以看出, 在计算机内存一定的情况下, 一次性构建的超结构的规模便确定了, 起决定性作用的是 b_j 值. 而 b_j 值与 m, q 及 β 紧密相关; 各项的 β 值与频繁 1- 项集投影数据库中项的密度有关, 也与数据库划分的 n 值有关, n 越大, β 值越小, 其取值范围为 0～ 1, 具体取值可通过取样得到. 一般而言, 算法中应尽量减少扫描数据库的次数, 而对数据库的扫描次数与一次性能构建的超结构的项数多少(或 i 值的大小) 有关. 因此, 在内存许可的情况下, 应尽可能构建多的项数的超结构. 基于上述思想, 假设系统能提供给构造超结构的构建过程的内存空间为 M, 频繁项投影数据库为 TDB, 投影数据库中的一条交易为 t, 则 AD-Mine 算法如下:

主模块结构:

```
1) 产生频繁 1- 项集投影数据库 TDB
2) i = 2, j = m
3) if b_ji > M then {
4)    call x1
5)    if b_ji > M then {
6)       k = ij
7)       call x2}}
8) else {β = 1, j = m, y = m }
9) if b_yj < M then {
10)    构建完全频繁项集超结构;
11)    输出满足支持度阈值的完全频繁项集 }
12) else {call x3}
13) 挖掘结束
proedure x1
1) 对投影数据库 TDB 进行按项值划分
2) 获得最大项值 q = Max(|X|)
3) 取样并计算 β_i, ⋯, β_q
4) j = q, y = q
procedure x2
1) 构造 m'× n 阶子数据库矩阵 Y
2) forall k ≤ m' do begin
3)    forall p ≤ n do begin
4)       扫描矩阵 Y 中的子数据库;
5)       if k > 2 then {
6)          if 所有 k- 项集的 (k - 1)- 项子集在超结构中存在 then {
7)             构建局部 k- 项超结构;
8)             剪枝获得局部 k- 项频繁项集 }
9)       else {
10)         构建局部 k- 项超结构;
11)         剪枝获得局部 k- 项频繁项集 }}
12)   end
13)   if k > 2 then {删除 (k - 1)- 项超结构 }
14)   合并得到全局 k- 项集候选项;
15)   扫描 TDB 中的 |X| ≥ k 部分, 得出 k- 项
```

集候选项支持度;

16) 删除超结构中小于支持度阈值的节点;

17) 得出全局频繁 k- 项集 B^k;

18) if $B^k \neq \varnothing$ then {

19) 输出频繁 k- 项集 B^k;

20) k + + }

21) else {挖掘过程结束}

22) end

23) 挖掘过程结束

procedure x3

1) $b_j^{(1)} = 0$

2) forall $i < m$ do begin

3) if $b_j^{(i)} + b_j^{(i-1)} < M$ then{

4) $k = i$;

5) forall $b_{ji} < M$ do begin

6) i + + ;

7) $b_{ji} = b_{j(i-1)} + b_j^{(i)}$;

8) end

9) 构造 k- 项到(i - 1)- 项超结构(只构造其子集(k - 1)- 项是频繁的项集, 并对交易 t 的所有(k - 1)- 项均是非频繁的交易进行剪枝;

10) if 存在 X^{i-1} 频繁项 then

11) {输出频繁项集;

12) if $b_j^{(i)} + b_j^{(i-1)} < M$

13) 删除超结构;

14) else{

15) 删除超结构中除频繁的(i - 1)- 项以外的相关节点}

16) $b_{ji} = b_j^{(i)}$}

17) else{挖掘过程结束}

18) else {

19) 由频繁(i - 1)- 项预测 $b_j^{(i)} = b_j^{\prime(i)}$;

20) if $b_j^{(i)} < M$ then {

21) forall $t \in$ TDB do begin

22) forall X^i = subset(t, i) do begin

23) if 每个 $X^{i-1} \in X^i$ 均在超结构中 then{构建 i- 项超结构}

24) end

25) if 每个 X^i 的所有 X^{i-1} 均不在超结构中 then{剪去交易 t}

26) end

27) if 存在 $|X| = i$ 的频繁项 X then{

28) 输出频繁项集;

29) 删除除频繁 i- 项以外的项及相关节点

30) i + + }

31) else{结束挖掘过程}}

32) else {

33) call x1

34) if $b_{qq} - b_{qi} < M$ then {

构建 $|X| \geq i$ 完全频繁项集超结构

35) 输出满足支持度阈值的完全频繁项集}

36) else {

37) $k = i$;

38) call x2}}

39) end

算法说明:

1) 当 $b_{mm} > M$ 时, 直接构造完全频繁项集超结构;

2) 当 $b_{mm} > M$ 且 $b_{m2} < M$ (或 $b_{qq} > M$ 且 $b_{q2} < M$) 时, 调用过程 x3; $b_{mm} > M$ 且 $b_{m2} < M$ 且 $b_{qq} < M$ 时, 直接构造完全频繁项集超结构;

3) 当 $b_{q2} > M$ 时, 调用过程 x2, 采用划分技术后进行挖掘

6 算法性能分析

目前比较有效的Apriori类算法是DHP 算法和DCP 算法, 而DCP 算法的有效性与操作系统的性能具有非常大的相关性[3]. 因此, 在此主要将本文算法与DHP 法进行比较分析, 而对于构造2- 项集候选项的超结构时超过可用内存的情况, 则与文献[6] 中的算法进行比较分析.

6.1 扫描数据库记录数量分析

1) 当 $b_{mm} < M$ 时, AD-Mine 算法一次性构建超结构, 扫描数据库 2 遍, 且第 2 遍扫描一次完成; 而DHP 算法扫描数据库的次数需多于 2 遍, 且第 2 遍开始每遍的扫描次数与候选项的数量成正比. 因而AD-Mine 算法对数据库的扫描次数更少.

2) 当 $b_{mm} > M$ 但 $b_{m2} < M$ 时, 2 种算法采用相同的交易剪枝技术, 当 $b_j^{(i)} + b_j^{(i-1)} < M$ (i = 3, 4, …, m) 时, AD-Mine 算法扫描数据库的次数必少于DHP 算法; 当 $b_j^{(i)} + b_j^{(i-1)} > M$ 且 $b_j^{\prime(i)} < M$ (这时的 $b_j^{\prime(i)}$ 通过预测得到) 时, 采用逐项构造超结构, 且只构造其所有(i - 1)- 项子集均频繁的 i- 项超结构, 此时AD-Mine 算法构建全部 i- 项超结构只需扫描数据库一次, 而DHP 算法是 l_i (l_i 为 i- 项候选项数) 次; 当 $b_j^{\prime(i)} > M$ 时, 由于AD-Mine 算法的划分技术和DHP 的候选项问题, 扫描记录数前者必然更少. 因而, 整个挖掘过程的总体扫描次数低于DHP 算法

3) 当 $b_{m2} > M$ 时,AD-Mine 算法首先用采样技术获取各项 β_i 和 q 值. 当 $b_{qq} < M$ 时, 一次性构建完全频繁项集超结构, 只需扫描 2 次数据库; 当 $b_{qq} > M$ 且 $b_{q2} < M$ 时, 采用分段构建超结构, 总的扫描次数必然小于DHP 算法的扫描次数; 当 $b_{q2} > M$ 时, 对于Apriori类算法中较为有效的处理方法是文献[6]中提出的方法, 需用采样和划分方法进行处理.

AD-Mine 算法与文献[6]算法相比具有以下优势: ① 新的划分方法降低了对数据库的扫描次数; ② 产生更少的局部候选项和全局候选项; ③ 按 $|X|$ 值逐项产生频繁项集, 逐项输出, 需要空间较少; ④ 采用按 $|X|$ 值逐项产生全局频繁项集, 一旦第 i 项不再有频繁项集, 则挖掘过程结束, 这样可以减少一些不必要的挖掘操作. 因此, 在此种情况下, AD-Mine 算法与Apriori 类算法相比仍具有更少的数据库扫描次数.

6.2　超结构内存需求算法有效性分析

超结构内存需求计算主要涉及到二项式的计算, 二项式的计算可通过累乘得到, 对于一个最大项数为 m 的超结构, 通过约简处理其 b_{mm} 的计算需要的循环累乘次数为 $m^2/2 + m - 2$ 次. 现假设AD-Mine方法与DHP 算法相比, 可减少扫描数据库的交易数为 N_1, 数据库 TDB 中的交易数为 N_2, T_{cpu} 表示每一循环的计算时间, T_{disk} 表示扫描每一条交易的时间, 而 $N_1 = \sum_{i=2}^{L}(l_i \times N_2)$. 这里: L 为最大频繁项集中的项数, l_i 是为获取 i-项频繁项集时的候选项数量. 一般说, $T_{disk}/T_{cpu} = 10^5 \sim 10^6$. 如 θ为虚拟内存等操作系统技术可能提高的扫描数据库的速度系数(θ > 1, 对于一个大数据而言, θ值是有限的), 则对于第1) 种情况, 构建完全超构时, 当 $(m^2/2 + m - 2) < (10^5/\theta)\sum_{i=2}^{L}(l_i \times N_2)$ 时是有效的.

可以看出, 对于具有正常 m 值的情况, 等式右边远远大于左边, N_2 越大AD-Mine 算法的效果越明显; 同时可以看出, 对于一个大数据库而言, 扫描遍数及候选项数的多少是影响效率的关键因素. 对于第 2) 种和第 3) 种情况, AD-Mine 算法仍然由于有较少的扫描次数, 而获得比Apriori类算法更高的效率.

6.3　数据库划分有效性分析

数据库划分的有效性反映在划分后减少的扫描交易数是否大于划分过程需要增加的对数据库交易的扫描次数上. 现假设有 D 个交易的投影数据库 TDB 按项值划分成 m' 部分, 这样 D 个交易划分成 l_1 个 1-项, l_2 个 2-项, …, $l_{m'}$ 个 m'-项, m'' 为最大频繁项集中的项数, 产生的 2-项, 3-项, …, m''-项候选项集的数量分别为 $c_2, c_3, \cdots, c_{m''}$ (这里 c_i ($i = 2, 3, \cdots, m''$) 是自然数); 又设 N' 为采用划分方法与不划分数据库两种情况的扫描数据库的交易总数之差, 则可通过推导得出:

当使用矩阵 Y 时, 有

$$N' = c_2\sum_{i=1}^{l} l_i + c_3\sum_{i=1}^{2} l_i + \cdots + c_{m''}\sum_{i=1}^{m''-1} l_i - \sum_{i=1}^{m'} l_i; \tag{2}$$

其他情况下为

$$N' \geq \sum_{i=1}^{1} l_i + \sum_{i=1}^{2} l_i + \cdots + \sum_{i=1}^{m''-1} l_i \tag{3}$$

式(3) 必定满足 $N' > 0$, 即划分必定是有效的; 式(2) 中最后一项是划分过程需要增加的扫描的交易数量, 其他项表示划分后可以减少的扫描的交易数量. 从AD-Mine 算法可以看出, 只在某项超结构足够大, 内存无法满足要求时, 才采用数据库的划分方法, 此时必定具有较大的 m'' 值, $N' > 0$ 的条件总是可以满足的, 即划分是有效的.

模拟研究也证实了以上理论推断. 设数据库中的交易数量为 $D = 100\,000$, m' 取值分别为 10 和 100, $l_1 = l_2 = \cdots = l_{m'} = D/m'$. 如图 4 所示, 当 $m' = 10$, $m'' \geq 4$ 时, 或 $m' = 100$, $m'' \geq 6$ 时, $N' \geq 0$, N' 随着 m'' 增加呈指数型增加趋势. 因此, 对于大的 m'' 值划分挖掘的效率是显著的.

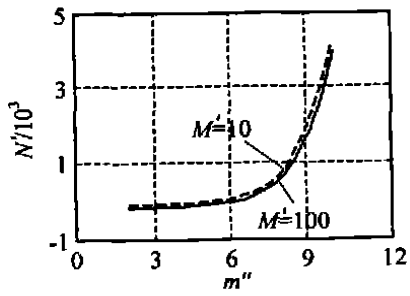

图 4　N′与 m″的关系图

综上所述, AD-Mine 算法比Apriori 类算法具有更高的效率.

(下转第 880 页)

到了 100%. 而文献[10]方法的学习结构的网络结构为: 4-8-4-3, 非零连接权个数为 128 个, 对所有样本的输出误差平方和为 3. 366 2(最好学习结果).

由上述试验可看出, 采用本文算法, 每个子神经网络的结构都减小很多, 并提高了整个神经网络群的泛化能力. 本文最终实验结果获得的 3 个子网络, 比文献[10]中的网络数目多([10]只有 1 个网络). 换言之, 本文将网络的规模大小与问题复杂性的矛盾转化为子网络数目多少与问题复杂性的矛盾. 后者比较容易由软件模拟或硬件解决, 而且基于进化规划的进化神经网络群更便于工程上并行实现.

4 结 语

本文提出了一种基于进化规划的神经网络群设计方法. 仿真实验表明, 由多个神经网络组成的神经网络群具有更为简洁的子网络结构和更好的算法性能. 同时还提出了一种用于网络权值训练的高斯变异与柯西变异相结合的进化规划算法. 该算法可以增强进化规划, 具有跳出局部极小的能力, 可改善算法的全局收敛性. 但是, 如何进一步发展神经网络群的设计方法及其相关理论尚有待于进一步研究.

参考文献(References):

[1] Liu Yong, Yao Xin, Tetsuya Higuchi. Evolutionary ensembles with negative correlation learning[J]. *IEEE Trans on Evolutionary Computation*, 2000, 4(4): 308-387.

[2] Liu Yong, Yao Xin. Simultaneous training of negatively correlated neural networks in an ensemble[J]. *IEEE Trans on Man and Cybernetics — Part B*, 1999, 29(6): 716-725.

[3] Baldi P. Learning in linear neural networks: A surve[J]. *IEEE Trans on Neural Networks*, 1995, 6(4): 837-858.

[4] Clemen R T, Winkler R L. Limits for the precision and value of information from dependent sources[J]. *Operartion Reasearch*, 1985, 33: 427-442.

[5] Perrone M, Cooper L N. *When Networks Disagree: Ensemble Methods for Hybrid Neural Networks*[M]. London: Chapman & Hall, 1993.

[6] 周明, 孙树栋. 遗传算法原理及应用[M]. 北京: 国防工业出版社, 1999. 74-77.

[7] 李人厚. 智能控制理论和方法[M]. 西安: 西安电子科技大学出版社, 1999. 178-179.

[8] 云庆夏. 进化算法[M]. 北京: 冶金工业出版社, 2000. 157-158.

[9] 李孝安. 进化神经网络理论及方法研究[D]. 西安: 西北工业大学, 2000. 92-93.

[10] 王磊, 戚飞虎. 进化计算在神经网络学习中的应用[J]. 计算机工程, 1999, 25(11): 41-43.
(Wang L, Qi F H. Applaction of evolutionary computation in neural network [J]. *Computer Engineer*, 1999, 25(11): 41-43.)

(上接第 871 页)

7 结 语

本文提出的AD-Mine 算法, 能够自动判别数据库中的数据特性, 确定挖掘策略, 最大限度地利用计算机的内存资源, 提高数据挖掘的效率, 与现有的算法相比具有更强的适应性, 与Apriori 类算法相比具有更高的效率. 同时, 本文提出的一种新的数据库划分方法, 是一种高效的方法, 可以显著提高数据挖掘的效率, 特别是对于挖掘较长频繁模式时, 优势更加明显.

参考文献(References):

[1] Agrawal R, Srikant R. Fast algorithms for mining association rules in large database[A]. *Proc of the 20th VLDB Conf*[C]. Santiago, 1994. 487-499.

[2] Park J S, Chen M S, Yu P S. An effective hash-based algorithm for mining association rules[A]. *Proc of the 1995 ACM SIGMOD*[C]. San Jose, 1995. 175-186.

[3] Orlando S, Palmerini P, Perego R. Enhancing the apriori algorithm for frequent set counting[A]. *Proc of 3rd Int Conf on DaWaK 2001*[C]. Munich: Spriger, 2001. 1-17.

[4] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[A]. *SIGMOD'00*[C]. Dallas, 2000. 1-12.

[5] Pei J, Han J, Lu H, et al. H-Mine: Hyper-structure mining of frequent in large database[A]. *Proc 2001 Int Conf on Data Mining (ICDM'01)*[C]. San Jose, 2001. 38-49.

[6] Savasers A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in larges dadabase[A]. *In 21st Int Conf on Very Large Databases (VLDB)*[C]. Ztrich, 1995. 432-443.