负增量式关联规则更新算法*)

张师超[1,2] 张继连[2] 陈 峰[2] 倪艾玲[2]

(悉尼科技大学信息技术学院 澳大利亚悉尼)[1] (广西师范大学计算机系 桂林 541004)[2]

摘 要 模式维护是数据挖掘中一个具有挑战性的任务。现有的增量式关联规则更新算法主要解决两种情况下的维护问题:一是最小支持度不变,而数据量增加;二是数据量不变,而改变最小支持度。本文提出了一种负增量关联规则更新算法。实验表明,该算法是有效的。

关键词 数据挖掘,关联规则,增量更新算法

Negative Incremental Updating Algorithm for Maintaining Association Rules

ZHANG Shi-Chao[1,2] ZHANG Ji-Lian[2] CHEN Feng[2] NI Ai-Ling[2]

(Faculty of Information Technology, University of Technology Sydney, Australia)[1]

(Department of Computer Science, GuangXi Normal University, Guilin 541004)[2]

Abstract Pattern maintenance is a challenging task in data mining. Existing incremental updating algorithms are designed for maintaining association rules in two ways. One is to maintain association rules with a constant minimal support while new data is added to original database. Another is to maintain association rules with different minimal supports and a constant database size. In this paper, a new algorithm, called Negative Incremental Updating Algorithm (NIUA), is proposed for maintaining association rules. We experimentally evaluate the proposed approach, and illustrate that our algorithm is efficient and promising.

Keywords Data mining, Association rules, Incremental updating algorithm

1 引言

数据挖掘(Data Mining),又名数据库中的知识发现(Knowledge Discovery in Database KDD),是从大量的数据中发现有效的、新颖的、潜在有用的知识[1]。关联规则的挖掘是数据挖掘中一个非常重要的任务。Agrawal 等提出了用于挖掘关联规则的 Apriori 算法[2]。人们利用挖掘出的关联规则来进行销售分析,市场预测,交叉销售等等,取得了明显的经济效益。

当数据库非常庞大,从其中挖掘关联规则时,挖掘算法的效率往往是人们主要考虑的问题。同时,大型数据库当中的数据是不断更新着的,人们要及时地对已挖掘出来的关联规则进行更新维护。关联规则的更新问题主要集中在两个方面:1,保持最小支持度不变而把新数据增加到原有的数据库当中;2,数据库的容量不变,而改变最小支持度。针对这些问题,许多学者提出了相应的增量更新算法。例如,Cheung 提出了 FUP 算法[3],Zhang 等提出了增量式维护算法[8],冯玉才,冯剑琳提出了 IUA 算法[5],它们解决了上面两种情况下的关联规则的更新问题。

相对增加而言,删除也是一种重要且频繁的数据库操作。显然,从数据库中减少一部分数据时的关联规则维护问题是值得研究的。例如,一个大型数据库中的某些数据过时了要删除,那么要相应地更新原来挖掘的关联规则,使得这些关联规则能反应出这种数据库改变。又比如,某公司有连续若干年的业务数据且挖掘出一些关联规则。根据需要,现在不考虑其中某几年数据的影响,而只分析其余年份的数据。为此要把不需要考虑的数据暂时从业务数据库中删除掉,单独挖掘其余年份数据,以获得新的知识。针对上述实际存在的问题,本文分析了数据量减少而支持度不变时的关联规则更新问题,提出了一种负增量更新算法(Negative Incremental Updating Algorithm,NIUA)。详细的问题在第 2 节中描述,算法的主要思想及其描述在第 3 节给出,第 4 节给出实验结果及结论。

2 问题描述

2.1 关联规则挖掘

假设 $I=\{i_1,i_2,\cdots,i_m\}$ 是 m 个不同项目的一个集合。其中的 i 称为项(*item*)。给定一个事务数据库 $D\{T_1,T_2,\cdots,T_n\}$,其中的每一条交易 T(transaction)是 I 的一个子集。每一条交易都有一个唯一的标识与之对应,记作 TID。设 A 是 I 中一个或多个项的集合,称作项集,包含 k 个数据项的项集称为 k-项集。当且仅当 $A\subseteq T$ 时,称交易 T 包含 A。

项集 A 在数据库 D 中的支持度定义为:$\mathrm{supp}(A)=|T(A)|/|D|$,其中 $T(A)$ 是数据库 D 中包含项集 A 的交易条数,即 $T(A)=\{T\in D|A\in T\}$,$|D|$ 为数据库 D 中事务总条数。对于用户给定的最小支持度阈值 minsupp,若 $\mathrm{supp}(A)\geqslant\mathrm{minsupp}$,则称 A 为频繁项集。

一条关联规则是形如"$A\Rightarrow B$"的蕴涵式,其中 $A\subseteq I$,$B\subseteq I$,并且 $A\cap B=\varnothing$。关联规则 $A\Rightarrow B$ 的置信度定义为:$\mathrm{conf}(A\Rightarrow B)=\mathrm{supp}(A\cup B)/\mathrm{supp}(A)$。关联规则的挖掘问题就是要从数据库 D 中找出所有满足以下两个条件的规则:

$\mathrm{supp}(A\cup B)\mathrm{minsupp}$

$\mathrm{conf}(A\Rightarrow B)\mathrm{minconf}$

其中 minconf 是用户指定的最小置信度阈值。关联规则的挖掘可以分解为两个子问题:

(1)从数据库 D 中找出所有满足用户指定最小支持度阈

*)澳大利亚 ARC 项目(DP0559536);国家自然科学基金重大项目(60496321);国家自然科学基金项目(60463003)。张师超 教授,博士,研究方向为数据挖掘,人工智能等;张继连 硕士研究生,研究方向为数据挖掘,数据库;陈 峰 硕士研究生;倪艾玲 硕士研究生。

值 minsupp 的项集，即找出所有的频繁项集。

(2)利用(1)找出的频繁项集来生成关联规则。对于每一个频繁项集 A，找出 A 的所有非空子集 a，如果比值 supp(A)/supp(a)大于等于 minconf，就生成关联规则 a⇒(A−a)。

相对而言，第 2 个子问题是较为容易解决的。挖掘关联规则的工作主要集中在第 1 个子问题上。

2.2 关联规则的更新

关联规则的更新主要由以下两个方面引起：

(1) 随着时间的推移，事务数据库中的某些数据有可能过时了，要删除掉；另一方面，也可能有新的数据要不断地添加进来。当数据库进行更新以后，以前挖掘出来的关联规则也需要相应地进行更新，以保证结果的正确性和有效性。目前较多的论文研究的是往数据库中添加新数据时的关联规则更新问题。

(2)在一次挖掘任务过程中，用户可能要不断地调整最小支持度和最小置信度这两个参数，找出这两个参数的最佳组合，从而发现其真正感兴趣的关联规则。这个过程要求响应时间要短。因此，需要用高效的更新算法来对已经生成了的关联规则进行更新，生成新的关联规则，以满足用户的要求。

本文要研究的是最小支持度不变而数据库容量减少时的关联规则更新问题。已有的文献[3~5,8]称数据库容量增加时的更新算法为增量更新算法，我们把数据库容量减小时的更新算法称为负增量更新算法。

3 负增量关联规则更新算法(NIUA)

3.1 算法主要思想

负增量关联规则更新问题的形式化描述如下：设原事务数据库为 DB，其包含的事务总条数为 $|DB|$，S_0 为用户给定的最小支持度，L 为数据库 DB 的频繁项集集合，db 是从 DB 中减去的那一小部分数据，其所包含的事务总条数为 $|db|$，L' 为 db 的频繁项集集合；DB-db 为 DB 减去 db 后剩下的数据库，L''为 DB-db 的频繁项集集合。解决数据库减少数据时的关联规则更新问题最简单的方法是直接把剩下的数据库 DB-db 挖掘一遍。如果有 $|db|\geqslant|DB|/2$，那么就可以直接挖掘数据库 DB-db 从而得出更新数据库后的关联规则；如果有 $|db|\leqslant|DB|/2$，直接挖掘 DB-db 的代价就比较大，特别是被删除的数据量远小于原数据库容量的一半时，直接挖掘的时间代价会更大。不失一般性，在本文中假定 $|db|<\frac{1}{2}|DB|$，算法的目的就是从节省时间开销入手，直接挖掘 db，通过分析挖掘 db 得出的项集和 DB 的频繁项集，得出数据库 DB-db 中的频繁项集。

如果项集 A 原来在 DB 中是频繁的，将数据集 db 从大数据库 DB 中减掉以后，A 在 DB-db 中有可能变为不频繁的；同样地，若 A 在 DB 中是不频繁的，在 DB-db 中也有可能变为频繁的。下面就项集在各个数据库中的频繁情况进行分析。

对于一个项集 A，设 S_1，S_2，S_3 是其在 DB 中的实际支持度，S_2 是在 db 中的实际支持度，$S3$ 是在 DB-db 中的实际支持度。则由支持度的定义有：

$$S_3=\frac{S_1\times|DB|-S_2\times|db|}{|DB|-|db|} \tag{1}$$

为了便于讨论项集 A 在 DB-db 中的频繁与否，令 $f=S_3-S_0$，并设 $\alpha=\frac{|db|}{|DB|}$，有：

$$f=\frac{S_1\times|DB|-S_2\times|db|}{|DB|-|db|}-S_0=$$

$$\frac{(S_1-S_0)\times|DB|-(S_2-S_0)\times|db|}{|DB|-|db|}=$$

$$\frac{(S_1-S_0)-(S_2-S_0)\times\alpha}{1-\alpha}$$

由上式可以看出，分母 $1-\alpha$ 恒为正，所以 f 的正负由分子来决定。分子大于等于零时，A 在 DB-db 中是频繁的，分子小于零时，A 是不频繁的。项集在各个数据库中的频繁与否，用判定公式 $(S_1-S_0)-(S_2-S_0)\times\alpha$ 来判断，有以下六种情况。

表 1 最小支持度为 S_0 时项集在各数据库中的情况

	在 DB 中	在 db 中	在 DB-db 中	说明
1	频繁	频繁	不确定	需要用实际的支持度由判定式计算决定
2	频繁	不频繁	频繁	由判定式可以直接得出结果，不用计算
3	频繁	不出现	频繁	直接得出结果，不用计算
4	不频繁	频繁	不频繁	由判定式可以直接得出结果，不用计算
5	不频繁	不频繁	不确定	需要用实际的支持度由判定式计算决定
6	不频繁	不出现	不确定	需要用实际的支持度由判定式计算决定

从表 1 中第 2,3,4 种情况可以得出以下几个性质：

性质 1 设 DB 和 db 使用相同的最小支持度 S_0，若一个项集在 DB 中是频繁的，而在 db 中没有出现，则该项集在 DB-db 中是频繁的。

证明：根据判定公式 $(S_1-S_0)-(S_2-S_0)\times\alpha$，已知项集在 db 中不出现，即该项集在 db 中的实际支持度为零，有 $S_2=0$。而该项集在 DB 中是频繁的，即 $S_1\geqslant S_0$，由此得判定公式恒为正，所以，该项集在 DB-db 中是频繁的。证毕。

性质 2 设 DB 和 db 使用相同的最小支持度 S_0，若一个项集在 DB 中是频繁的，而在 db 中是不频繁的，则该项集在 DB-db 中是频繁的。

证明：根据判定公式 $(S_1-S_0)-(S_2-S_0)\times\alpha$，因为已知项集在 DB 中是频繁的，即 $S_1\geqslant S_0$，所以，判定公式第一项恒大于等于零。又因为项集在 db 中是不频繁的，即 $S_2<S_0$，且有 $\alpha>0$，所以，第二项恒为负，最后得出判定公式恒为正。于是，该项集在 DB-db 中为频繁的。证毕。

性质 3 设 DB 和 db 使用相同的最小支持度 S_0，若一个项集在 DB 中是不频繁的，而在 db 中是频繁的，则该项集在 DB-db 中是不频繁的。

证明：根据判定公式 $(S_1-S_0)-(S_2-S_0)\times\alpha$，已知项集在 DB 中是不频繁的，即 $S_1<S_0$，得判定公式第一项为负。又已知该项集在 db 中是频繁的，即 $S_2\geqslant S_0$，得判定公式第二项为正，最后得判定公式为负。所以，该项集在 DB-db 中是不频繁的。证毕。

讨论表 1 中其它几种情况。第 1 种情况下，因为项集 A 在 DB 中是频繁的，其实际的支持度可以获得，由判定式计算结果来决定 A 在 DB-db 中是否频繁。对于第 5,6 种情况，项集 A 在 DB 中是不频繁的，由于一般的关联规则挖掘算法在产生频繁项集的过程中会把非频繁项集剪掉，因此非频繁项集 A 的实际支持度就无从得知，无法用判定式来计算。最简单的解决办法是把传统的关联规则挖掘算法进行扩展，使其不仅能够产生频繁项集，同时还能够产生非频繁项集，如文

[6]。有了非频繁项集的实际支持度，就可以用判定式进行判断了。但是这种用空间来换取时间的办法并不总是有效的，当 *DB* 非常大时，其产生的非频繁项集往往非常多，需要用巨大的存贮空间来存贮它们。同时，维护和扫描计算这些非频繁项集也要花费一定的时间。

本文不采用产生并记录非频繁项集的方法，而采用我们称之为负增量关联规则更新的算法。首先，我们以最小支持度 S' 挖掘 *db*，得出 *db* 的项集集合 L'，其中 $S'=\frac{3}{5}S_0$，这样能够保证从 *db* 中挖掘出更多的项集，而这些项集在支持度为 S_0 时是非频繁的，用传统的挖掘算法得不到它们。对于 *DB* 的频繁项集集合 L 中的每个项集 $L.c$，扫描一遍 L'，如果存在某个项集 $L'.c$，使得 $L'.c=L.c$ 且 $\frac{3}{5}S_0 \leqslant \mathrm{supp}(L'.c)<S_0$，由性质 2 知项集 $L'.c$ 在 *DB-db* 中是频繁的；若 $\mathrm{supp}(L'.c)\geqslant S_0$，则需要用式(1)来计算其在 *DB-db* 中的实际支持度，从而判断其是否频繁；如果不存在某个 $L'.c$，使得 $L'.c=L.c$，则由性质 1 得出 $L.c$ 在 *DB-db* 中是频繁的，只要直接把 $L.c$ 并入 L'' 就行了。L 中的项集都处理完后，对于 L' 中剩下的项集，根据性质 3 可推得所有支持度大于等于 S_0 的项集在 *DB-db* 中是不频繁的，可以将它们从 L' 中删掉。这样 L' 中最后剩下的项集，相对于支持度 S_0 而言它们在 *db* 和 *DB* 中都是不频繁的了，对应于表 1 中第 5 种情况，为了获得它们在 *DB* 中的实际支持度，需要扫描一遍 *DB*，然后再用式(1)来计算，决定它们在 *DB-db* 中是否成为频繁的。L' 中剩下的项集也处理完后，此时只剩下表 1 中的第 6 种情况，即如何处理在 *db* 中不出现而在 *DB* 中是不频繁的项集。注意到我们对 *db* 是用较小的支持度 $S'=\frac{3}{5}S_0$ 进行挖掘的，因此 *db* 中肯定存在一定数量的非频繁项集，它们在 *db* 中的实际支持度是得不到的。因此我们把这些非频繁项集视为在 *db* 中不出现，以对应于第 6 种情况进行处理。为了获得在 *db* 中不出现而在 *DB* 中为非频繁的项集，需要在 *DB-db* 中随机抽取一小部分数据，然后用较小的支持度挖出它的频繁项集 L_1，再从 L_1 中删掉在 *DB* 和 *db* 的频繁项集集合中出现过的元素，那么 L_1 中剩下的项集便是在 *DB* 中非频繁而在 *db* 中不出现的一部分项集，对于 L_1 中剩下的所有项集，扫描一遍 *DB*，得到这些项集在 *DB* 中的实际支持度，再用式(1)计算它们在 *DB-db* 的实际支持度，最后得出它们在 *DB-db* 中是否频繁。在实验结果中我们发现，当 *db* 是从 *DB* 中随机删减掉且 *db* 的数据量中等时，*db* 中项集的分布与 *DB* 中项集的分布相近，因此用 *NIUA* 算法只分析 *db* 和 *DB* 的项集便能基本上得到 *DB-db* 的频繁项集，在一定的准确率容许范围内，可以不用考虑从 *DB-db* 中抽取数据来分析表 1 中第 6 种情况，以保证算法较高的执行效率。

3.2 算法描述

负增量关联规则更新算法的形式化描述如下：

输入：1)DB：原事务数据库
2)L：DB 的频繁项集集合
3)db：从 DB 中减去的事务数据库
4)S_0：最小支持度

输出：L"：DB-db 的频繁项集集合

1)用关联规则挖掘算法，以 S′为最小支持度，挖出 db 的频繁项集 L′
2)for all itemset c∈L do
2.1) 扫描一遍 L′中的所有元素
2.2) if 找到某个项集=c then
根据表 1 中第 1,2 种情况处理，若 c 是频繁的，则 L" = L"∪c
else 依第 3 种情况，直接有 L" = L"∪c
3)根据第 4 种情况，把 L′中支持度大于 S_0 的项集 c 做上标记，它们在 DB-db 中是不频繁的，不用再对它们进行处理。
4)for all T∈DB do
4.1) for remained itemset c ∈ L′ do
4.2) if c⊆T then 统计 c 在 DB 中的实际支持度
5)for remained itemset c∈L′ do
5.1) 根据 c 在 DB 中的统计结果和 c 原来在 db 中的支持度，用判定式对 c 进行判定，若 c 是频繁的，则 L"=L"∪c
6)在 DB-db 中随机抽样，得到抽样数据库 db_1
7)用关联规则挖掘算法，以 S'_1 为最小支持度，挖出 db_1 的频繁项集 L'_1
8)for all itemset c ∈ L'_1 do
8.1) if c appears in L∪L′ then delete c from L'_1
9)for all T∈DB do
9.1) for all itemset c∈L'_1 do
9.2) if c⊆T then c.count++;
10)for all itemset c∈L'_1 do
11) if c.count $\geqslant S_0\times(|DB|-|db|)$ then L"=L"∪c

实验结果及结论 算法用 VC++ 6.0 实现，采用文[7]的数据生成程序生成实验所需要的合成数据集，其中 $|DB|=100K$，$|T|=10$，$|I|=4$，$|L|=2000$，$N=1000$，分别从 *DB* 中随机抽取 1%，5%，10%，20%，30% 的数据作为 *db*，模拟从 *DB* 中被删除的数据集。在 *db* 取不同容量大小的情况下依次取最小支持度为 1.5%，1%，0.75%，0.5%，在内存为 2.0G，CPU 主频为 2.6G，操作系统为 WINDOWS 2000 的 Dell Workstation PWS650 计算机上进行实验，将 NIUA 算法与 Apriori 算法进行了比较。图 1 和图 2 是从 *DB* 中随机抽取 5% 的数据作为 *db*，使用不同的最小支持度时的实验结果。

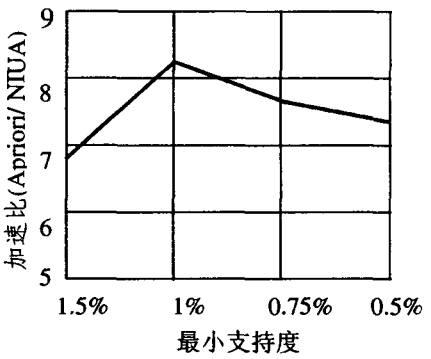

图 1 最小支持度改变时的性能加速比(T10 I4 D100K)

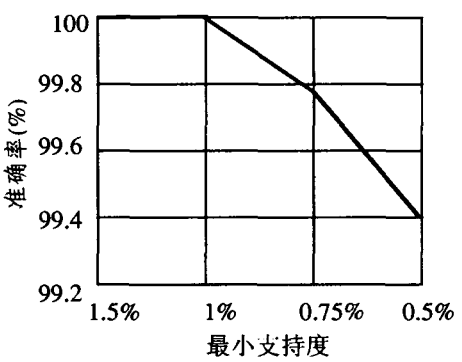

图 2 最小支持度改变时的准确率变化情况(T10 I4 D100K)

从上图可以看出，当 *DB* 数据量为 100K，*db* 数据量为 5K

(下转第 175 页)

文中还讨论了初始训练样本数与识别率之间的关系，指出对于数字语音识别，在达到一定的样本规模后继续增加初始训练样本所带来的附加利益不是太大，应用中需要从计算复杂度、存储容量与识别率等几个方面进行权衡。

整个研究过程并未对各个算法细节作过多的强调，例如采取某些降噪措施、确切地获取语音端点位置、精确地进行模式分类以得到最佳的中心向量等。虽然细细地挖掘各个处理环节仍可获得一点利益，却不是研究的重点，这里考察的是具有一定鲁棒性的模型的识别率。

实验发现，即使是采用基于数字语音时频信息整体结构的单特征向量识别模型，仍然存在着少数数字语音的判决模糊问题。进一步降低错识率的研究可以从两个方面进行：(1) 运用分层结构的模式，通过调整权系数突出不同数字语音的个性差异，减小特征空间的重叠区域。(2) 改进模型结构，扩大参数范围，通过挖掘和运用各种参数的互补信息降低错识率。

参 考 文 献

1 Kopec G E, Bush M A. Network-Based Isolated Digit Recognition Using Vector Quantization. IEEE Trans on Acoustics, Speech and Signal Processing, 1985, 33(4): 850～867

2 Loizon P C, Spanias A S. High-Performance Alphabet Recognition. IEEE Trans on Speech and Audio Processing, 1996, 4(6): 430～445

3 Hao Ying, Fang Ditang. Speech Recognition Using Speaker Adaptation by System Parameter Transformation. IEEE Trans on Speech and Audio Processing, 1994, 2(1): 63～68

4 Gales M J F. Cluster Adaptive Training of Hidden Markov Models, IEEE Trans on Speech and Audio Processing, 2000, 8(4): 417～428

5 Huo Qiang, Chan Chorkin, Lee Chin-Hu. Bayesian Adaptive Learning of the Parameters of Hidden Markov Model for Speech Recognition. IEEE Trans on Speech and Audio Processing, 1995, 3(4): 334～345

6 Karnjanadecha M, Zahorian S A. Signal Modeling for High-Performance Robust Isolated Word Recognition. IEEE Trans on Speech and Audio Processing, 2001, 9(6): 647～654

7 Young S, Evermann G, Kershaw D, etc. HTK Book (Version 3.2.1), Cambridge University Engineering Department, 2002

(上接第 155 页)

时，用 NIUA 算法对关联规则进行更新的效率是直接运用 Apriori 算法进行更新的 6～8 倍；准确率方面，在数据库 *DB-db* 中，用 NIUA 算法得到的频繁项集数量为直接运用 Apriori 算法挖掘到的频繁项集数量的 99.4%～100%。当支持度变小时，准确率有所下降，原因是数据库中支持度较小的项集一般来说比较多，若最小支持度定得比较低，则实际支持度略小于最小支持度的很多非频繁项集在 *DB-db* 中将变为频繁的，而这些项集有可能不在 *db* 中出现，分析 *db* 时找不到这些项集，而实际上它们在 *DB-db* 中却是频繁的，因此降低了 NIUA 算法的准确率。

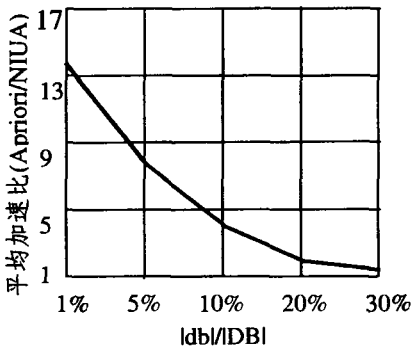

图 3 db 的数据量改变时的平均性能加速比(T10 I4 D100K)

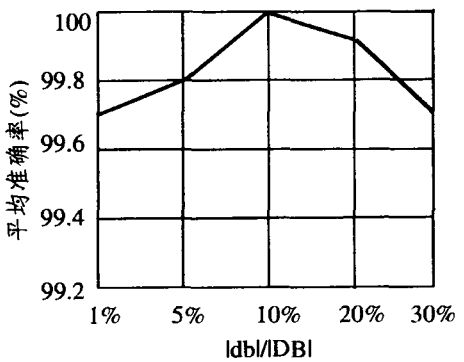

图 4 db 的数据量改变时的平均准确率变化情况(T10 I4 D100K)

图 3，图 4 分别为依次从 *DB* 中抽取 1%，5%，10%，20%，30%的数据作为 *db* 时，NIUA 算法与 Apriori 算法的平均执行时间之比和平均准确率变化情况。

从图 3 可以看出，当 *db* 的数据量为 *DB* 的 1%时，NIUA 算法的效率最高，是 Apriori 算法的 13 倍左右。随着 *db* 的增大，NIUA 算法的效率随之下降，当 *db* 增大到 *DB* 的 30%时，NIUA 算法与 Apriori 算法的效率相近。从图 4 可看出，当 *db* 数据量为 *DB* 的 10%～20%时，NIUA 算法的平均准确率较高，都在 99.8%以上。但是从准确率变化曲线可以预见，平均准确率有明显的下降趋势。

由以上分析及实验结果可看出，用本文提出的 NIUA 算法对数据库进行减量时的关联规则更新，效率比 Apriori 算法高。但是当最小支持度降低，或者原数据库很大而减掉的那部分数据又很少时，算法的效率会下降，并且准确率会降低。当减掉的数据量达到或者超过原数据库容量的 30%时，NIUA 算法的执行时间与 Apriori 算法相近。

参 考 文 献

1 Han Jiawei, Kamber M. Data mining: Concepts and techniques [M]. Beijing: Higher Education Press, 2001

2 Agrawal R. Mining Association Rules Between Sets of Items in Large Database[C]. In: Proc. of ACM SIGMOD Conf. on Management of Data, Washington, 1993. 207～216

3 Cheung D W. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In: Proc. of the 12th Intl. Conf. on Data Engineering, New Orleans, Louisiana, 1996. 106～114

4 陈劲松，施小英. 一种关联规则增量更新算法[J]. 计算机工程，2002, 28(7): 106～107

5 冯玉才，冯剑琳. 关联规则的增量式更新算法[J]. 软件学报，1998, 9(4): 301～306

6 孙浩，赵霁. 一种关联规则增量更新算法[J]. 系统工程与电子技术，2004, 26(5): 676～677

7 http://www.almaden.ibm.com/cs/quest/data/assoc.gen.tar.Z

8 Zhang Shichao, Zhang Chengqi, Yan Xiaowei. Post-mining: Maintenance of Association Rules by Weighting. Information Systems, 2003. 28(7): 691～707