关联规则算法的研究

文 拯,梁建武,陈 英
(中南大学 软件学院,湖南 长沙 410005)

摘 要:Apriori算法是发现频繁项目集的经典算法,但是该算法需反复扫描数据库,因此效率较低。文中针对传统的Apriori算法需要多次扫描数据库,由此导致的性能瓶颈及效率问题,提出了一种改进的关联规则挖掘算法(AAC算法)。该算法只需一次扫描数据库即可完成所有频繁项集的搜索,极大地提高了算法的效率。

关键词:AAC算法;关联规则;数据挖掘;Apriori算法;一次扫描数据库的Apriori
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2009)05-0056-03

Research of Association Rules Algorithm

WEN Zheng, LIANG Jian-wu, CHEN Ying
(Software College, Central South University, Changsha 410005, China)

Abstract: Apriori algorithm is the classic method which used to detect frequent item sets. But due to the algorithm need to be repeated scanning the database, it has less efficiency. The article for the traditional Apriori algorithm necessary to scan the database many times and the resulting efficiency and performance bottlenecks, raising an improved method of mining association rules (AAC algorithm), it only need to scan the database one time to finish all the frequent item sets detecting, greatly improve the efficiency of the algorithm.

Key words: AAC algorithm; multi- association rule; data mining; Apriori algorithm; one time scan database's Apriori algorithm

0 引 言

关联规则挖掘就是在海量的数据中发现数据项之间的关系,是数据挖掘领域中研究的热点问题。1993年Agrawal等人首先提出了交易数据库中不同商品之间的关联规则挖掘,并逐渐引起了专家、学者的重视。关联规则挖掘问题可以分为:发现频繁项目集和生成关联规则两个子问题,其中发现所有的频繁项目集是生成关联规则的基础[1]。近年来,发现频繁项目集成为了关联规则挖掘算法研究的重点,在经典的Apriori算法的基础上提出了大量的改进算法。但在这些改时算法里面,都不可避免地要多次扫描数据库,只是在量的方面提升了算法的效率,却没有得到质的改进[2]。

文中首先介绍Apriori算法,然后提出一种新的关于关联规则的灵活算法,该算法只需一次扫描数据库,并且在连接时可以同时剪枝,极大地提高了算法的效率。

收稿日期:2008-09-01
基金项目:国家自然科学基金(60173041)
作者简介:文 拯(1985-),男,江西萍乡人,硕士研究生,研究方向为数据库,数据挖掘,网络安全与认证等;梁建武,高级工程师,研究方向为信号处理,网络安全与认证;陈 英,硕士,副教授,研究方向为网络通信与安全技术。

1 关联规则算法

1.1 基本概念

定义1:事务(transaction):某个客户在一次交易中发生的所有项目的集合,每个事务都有一个唯一的标识-tid。

定义2:事务数据库(transaction database)

$D=\{t_1,t_2,\cdots,t_n\}$;由一系列具有唯一标识tid的事务组成,且tid是有序的(tid有序为新算法的要求)。

定义3:项(item):事务数据库中的一个属性字段,每个字段有一定的取值范围。对超市数据来讲,项是指交易中的特定商品。

定义4:项集(itemset):包含若干个项的集合。

定义5:项集维数:把一个项集所包含的项的个数称为此项集的维数或项集的长度。长度为k的项集,称为k-项集。

定义6:支持度(support):假定X是一个项集,D是一个事务集合或事务数据库,称D中包含X的交易的个数与D中总的交易个数之比为X在D中的支持度,记作Support(X),即

$$\text{Support}(X)=\frac{\|\{d\in D\mid X\subseteq d\}\|}{\|D\|}$$

定义7:最小支持度(minimum support):由用户定义的衡量项集频繁程序的一个阈值,记作minsup。

定义 8:频繁项集(frequent itemset):对一个项集 X,如果 X 的支持度不小于最小支持度,即 Support(X) $\geq$ minsup,称 X 为频繁项集。

定义 9:非频繁项集(non - frequent itemset):对一个项集 X,如果 X 的支持度小于最小支持度,即 Support(X) $\leqslant$ minsup,称 X 为非频繁项集。

定义 10:置信度(confidence):对形如 $X \rightarrow Y$ 的关联规则,其中 X 和 Y 都是项集,定义规则的置信度为事务集合 D 中既包含 X 也包含 Y 的事务个数与 D 中包含 X 的事务个数之比,或者说是项集 $X \cup Y$ 的支持度与 X 的支持度之比,记作 Confidence($X \rightarrow Y$),即

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$$

定义 11:最小置信度(minimum confidence):用户定义的一个置信度阈值,表示规则的最低可靠性,记作 minconf[3]。

1.2 关联规则挖掘基本思想

关联规则数据挖掘的挖掘过程基本分为两步:

(1)挖掘频繁项项集。

通过用户给定的最小支持度 minsup,寻找所有频繁项集,即支持度 Support 不小于最小支持度 minsup 的所有项集。

(2)生成关联规则。

使用频繁项集生成置信度大于预先给定的最小置信度阈值的关联规则。

挖掘关联规则的整个性能主要是由第一步(挖掘频繁项集)决定的。因此文中只讨论如何有效地计算频繁项集。

2 Apriori 算法

2.1 Apriori 的基本思想

Apriori 算法使用宽度优先的迭代搜索方法,首先找出频繁 1 - 项集集合 F_1,用 F_1 找频繁 2 - 项集合 F_2,用 F_2 找 F_3,依次循环,直到不能找到频繁 k - 项集为止。

它通过如下两个步骤来完成[4]:

(1) 连接:通过 L_{k-1} 与自身进行连接

$\text{TID}(X_K) = \{\forall X_{k-1} \in X_k, \text{TIDS}(X_k) = \bigcap_{i=1}^{k-1} \text{TIDS}(X_{k-1}[i])\}$

($L_{k-1} \otimes L_{k-1}$) 产生候选 k - 项集 C_k;

(2)剪枝:C_k 是 L_k 的超集,即它的成员可能是也可能不是频繁的,但所有的频繁 k - 项集都包含在 C_k 中。扫描数据库,确定 C_k 中每个候选项集的计数,计数值不小于最小支持度阈值的所有候选项集都是频繁的,从而属于 L_k。

2.2 Apriori 的算法分析

Apriori 算法在执行"连接 - 剪枝"的迭代过程中,需要多次扫描数据库,如果生成的频繁项目集中含有 20 - 项集,则需要扫描 20 遍数据库,增大了 I/O 负载。并且在迭代过程中,候选项目集合 C_k 是以指数速度增长的,L_{k-1} 自连接会产生大量的候选 k - 项目集,例如有 10^3 个 1 - 项集,自连接后就可以产生大约 10^6 个候选 2 - 项集。这些都严重影响了 Apriori 算法的效率[5]。

3 Apriori 改进算法(AAC 算法)

3.1 AAC 的基本思想

Apriori 算法在迭代过程中多次扫描数据库和产生大量的候选项目集形成了算法的性能瓶颈。为了提高算法的效率文中进行如下改进:

数据库 D 中每个交易 T 都有一个唯一有序的编号 TID。

定义 12:新算法使用的 k - 项集结构如下:{k - item - set, tids},其中 tids 为事务数据库 D 所有包含 k - item - set 的索引集合。

算法描述如下:

C_1 = findFrequentOneItemSets(D);

$L_1 = \{X_1 \in C_1 \cap \text{TIDS}(X_1) \geq \text{minsup}\}$

连接与剪枝:通过 L_{k-1} 与自身进行连接($L_{k-1} \otimes L_{k-1}$) 产生候选 k - 项集 C_k,并且在连接时,直接通过 L_{k-1} 元素的 tids 相交得到 C_k 元素的 tids;即:$C_k = L_{k-1} \otimes L_{k-1}$,其中

剪枝:直接判断 C_k 中成员的 tids 的元素个数是否大于 minsup;如小于则直接剪枝。即:$L_k = \{X_k \in C_k \cap \text{TIDS}(X_k) > \text{minsup}\}$

3.2 AAC 算法描述(程序)

●Procedure main()

1)L_1 = find_ frequent_ 1 - itemsets(D);

//遍历数据库 D,产生频繁 1 项集;

2)For($k = 2; L_{k-1} \neq \emptyset; k++$){

3)C_k = Apriori_ gen(L_{k-1}, min_ sup);//新的候选项集

4)$L_k = \{C \in C_k \mid \text{C.tids.count} \geqslant \text{min\_ sup}\}$

5)}

6)return$L = U_k L_k$;

●procedure Apriori_ gen(L_{k-1}:frequent(k - 1) - itemsets;min_ sup:minimu support threshold)

1)for each itemset $P_1 \in L_{k-1}$

2)for each itemset $P_2 \in L_{k-1}$

3)if($P_1[1] = P_2[1] \wedge P_1[2] = P_2[2] \wedge \cdots \wedge$

$P_1[K-2] = P_2[K-2] \wedge P_1[K-1] < P_2[K-1]$)then{

4)$C = P_1 \otimes P_2$//join step;generate candidates

5)if has_ infrequent_ subset(C, L_{k-1})then

6)delete C;//Prune step:remove unfruitful cadidate

7)else{

8)for each($K-1$) − subset S of C

9)C.tids $\cap$ = S.tids;

add C toC_k;

10)}}

11)return C_k;

●Procedurce has_ infrequent_ subset(c:candidate k − itemset;L_{k-1}:frequent($k-1$) − itemset)

1) for each($K-1$) − subset s of C

2)if $s \in L_{K-1}$then

3)return true;

4)return false;

3.3 AAC 算法举例分析

设数据库 D 如表 1 所示,最小支持数 minsup=4,运行改进的算法的过程如下所示:

具体步骤如下:

遍历整个事务数据库,找出 C_1,并通过剪枝得出 L_1;

通过 $L_1 \otimes L_1$ 得到 C_2,其中 TIDS(X_2) = TIDS(X_1) $\cap$ TIDS(X_1),并通过剪枝得出 L_2;

通过 $L_2 \otimes L_2$ 得到 C_3,并且 C_3 中的项集的 2 − 项集都 $\subseteq L_2$;

其中:TIDS(X_3) = {$\forall X_2 \in X_3$,TIDS(X_3) = $\cap_{i=1}^{3}$TIDS($X_2[i]$}

即:TIDS(t_1,t_2,t_3) = TIDS(t_1,t_2) $\cap$ TIDS(t_1,t_3) $\cap$ TIDS(t_2,t_3)

3.4 实 验

为了验证算法优化算法的性能,实验旨在比较算法的效率。针对同样的数据库,在相同的硬件(CPU P4 2.66,内存 1G,硬盘 160G)和软件环境下,采用 Apriori 算法和 AAC 算法进行对比测试,在相同的支持度(20%)下,两个算法的时间比较,结果如表 1 和图 1 所示。

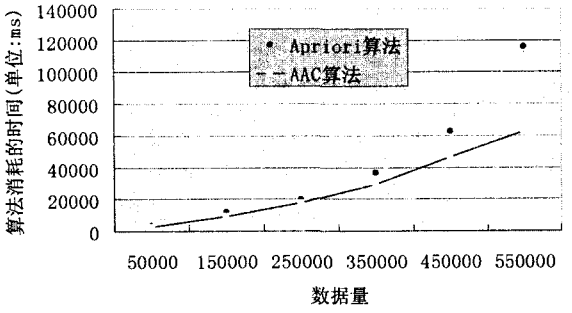

图 1 不同数据量的情况下运行时间的比较

表 1 实验数据表

时间(ms) \ 数据量	50000	150000	250000	350000	450000	550000
Apriori	3280	11896	19765	36170	52600	91560
AAC	2623	9211	15662	25117	39338	52663

4 结束语

传统的关联规则方法大都需要多次扫描数据库,极大地降低了算法的效率[6]。而 AAC 算法,只是在生成时进行了一次数据库扫描,在之后的迭代过程中不需要扫描数据库。Apriori 及其改进算法相比,使用 AAC 算法大大降低了 I/O 负载,使得频繁项目集的发现速度大大提高,尤其是在项目集长度较大的情况下。算法的迭代过程不需要复杂的计算,项目集连接仅仅使用集合的并、交运算即可完成,使得该算法易于实现,相信该算法具有一定的理论与实用价值。

但是该算法也有不足:为了减少 I/O 负载,要求在第一次扫描时把所有的信息装入内存,虽然本算法对数据库进行编码,以二元组的形式存储项集,但是数据挖掘都是基于海量数据的,因此,算法运行时需要大量内存,对此将在今后的研究中进行改进。

参考文献:

[1] Cai C H, Fu A, Cheng C H, et al. Mining Association Rules with Weighted Items[R]. IEEE DEAS. [s. l.]: [s. n.], 2002:68 − 77.

[2] Ballou D P, Tayi G K. Enhancing data quality in data warehouse environments[J]. Communication of ACM, 1999, 42:73 − 78.

[3] 胡可云,田凤占,黄厚宽. 数据挖掘理论与应用[M]. 北京:清华大学出版社,2008.

[4] 邵峰晶,于忠清. 数据挖掘原理与算法[M]. 北京:中国水利水电出版社,2003.

[5] 李敏强,寇纪淞,李 丹. 遗传算法的基本理论与应用[M]. 北京:科学出版社,2003.

[6] 徐 勇,周森鑫. 一种改进的关联规则挖掘方法研究[J]. 计算机技术与发展,2006,16(3):77 − 79.