序列模式挖掘算法研究

夏明波[1],王晓川[1],孙永强[2],金士尧[1]

(1.国防科学技术大学 计算机学院 并行与分布式国家重点实验室,湖南 长沙 410073;
2.国防科学技术大学 计算机学院,湖南 长沙 410073)

摘 要:数据挖掘领域一个活跃的研究分支就是序列模式的发现,即在序列数据库中找出所有的频繁子序列。目前的序列模式挖掘方法主要分为两类,一类是候选集生成-测试方法;另一类是模式扩展方法。先介绍序列模式挖掘中的基本概念,然后描述几个重要算法,最后给出性能分析。

关键词:序列模式挖掘;候选集生成-测试;模式扩展;算法分析

中图分类号:TP301.6 **文献标识码**:A **文章编号**:1005-3751(2006)04-0004-03

Research on Sequential Pattern Mining Algorithms

XIA Ming-bo[1], WANG Xiao-chuan[1], SUN Yong-qiang[2], JIN Shi-yao[1]

(1. National Key Lab. for Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China;
2. College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: An active research in data mining area is the discovery of sequential patterns, which finds all frequent sub-sequences in a sequence database. Recent studies can be divided into two major classes of sequential pattern mining methods: a candidate generation-and-test approach; a pattern-growth method. This paper firstly introduces the basic concept of sequential pattern mining, then describes the main algorithms and finally analyzes their performance.

Key words: sequential pattern mining; candidate generation-and-test; pattern-growth; algorithm analysis

0 前 言

序列模式挖掘或称序列挖掘,是从序列数据库中发现相对时间或者其他顺序所出现的高频率子序列。其最初动机是想通过在带有交易时间属性的交易数据库中发现频繁项目序列以发现一时间段客户的购买活动规律。近年来序列模式挖掘已经成为数据挖掘的一个重要方向,其应用范围不局限于交易数据库,在DNA分析等尖端科学研究领域、Web访问等新型应用数据源等众多方面得到针对性研究。

文章以AprioriAll,GSP和PrefixSpan三个典型算法为例,对两类算法进行介绍、分析和总结。

1 序列模式挖掘的基本概念

项目集或称项集(Itemset)是各种项目组成的集合。设 $I=\{i_1,i_2,\cdots,i_m\}$ 是一个项目集合,事务数据库 $D=\{t_1,t_2,\cdots,t_n\}$ 是由一系列具有惟一标识TID的事务组成,每个事务 $t_i(i=1,2,\cdots,n)$ 都对应 I 上的一个子集。设 $I_1\subseteq I$,项目集 I_1 在 D 上的支持度(support)是包含 I_1 的事务在 D 中所占的百分比,即

$$\text{support}(I_1)=\|\{t\in D\mid I_1\subseteq t\}\|/\|D\|$$

对项目集 I 和事务数据库 D,I 中所有满足用户指定的最小支持度(Minisupport)的项目集,即大于或等于Minisupport的 I 的非空子集,称为频繁项目集或者大项目集。在频繁项目集中挑选出所有不被其他元素包含的频繁项目集称为最大频繁项目集(Maximum Frequent Itemsets)或最大大项目集(Maximum Large Itemsets)[1]。

Agrawal等人建立了用于事务数据库挖掘的项目集格空间理论。这个理论的核心思想是:频繁项目集的子集是频繁项目集;非频繁项目集的超集是非频繁项目集。这个理论是经典的数据挖掘理论,也是下面将讨论的算法的理论依据。

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是项集,$i_k(1\leqslant k\leqslant m)$ 是一个项,序列 S 记为 $S=<s_1,s_2,\cdots,s_n>$,其中 $s_j(1\leqslant j\leqslant n)$ 为项集(也是 S 的元素)。每个元素由不同项组成。序列的元素可表示为 $(i_1,i_2,\cdots,i_k)$。序列包含的项的个数称为序列的长度,长度为 k 的序列记为 k-序列。

设 $\alpha=<a_1a_2\cdots a_n>$,$\beta=<b_1b_2\cdots b_m>$,如果存在

收稿日期:2005-10-19

作者简介:夏明波(1980-),男,湖南武冈人,硕士研究生,研究方向为信息安全与数据挖掘;金士尧,教授,博导,研究方向为系统仿真与信息安全。

整数 $1 \leqslant j_1 < j_2 < \cdots < j_n \leqslant m$,使得 $a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, \cdots, a_n \subseteq b_{jn}$,则称序列 α 为序列 β 的子序列,又称序列 β 包含序列 α,记为 $\alpha \subseteq \beta$。序列 α 在序列数据库 S 中的支持数为序列数据库 S 中包含序列 α 的序列个数,记为 Support(α)。给定支持度阈值 ζ,如果序列 α 在序列数据库中的支持数不低于 ζ,则称序列 α 为序列模式,长度为 k 的序列模式记为 k - 模式。

下面是几个在模式扩展方法中用到的重要概念[2]:

前缀(Prefix):设每个元素中的所有项目按照字典序排列。给定序列 $\alpha = < e_1 e_2 \cdots e_n >$, $\beta = < e_1' e_2' \cdots e_m' >$ $(m \leqslant n)$,如果 $e_i' = e_i (i \leqslant m-1)$, $e_m' \subseteq e_m$,并且 $(e_m - e_m')$ 中的项目均在 e_m' 中项目的后面,则称 β 是 α 的前缀;

投影(Project):给定序列 α 和 β,如果 β 是 α 的子序列,则 α 关于 β 的投影 α' 必须满足:β 是 α' 的前缀,α' 是 α 的满足上述条件的最大子序列;

后缀(Suffix):序列 α 关于子序列 $\beta = < e_1 e_2 \cdots e_{m-1} e_m' >$ 的投影为 $\alpha' = < e_1 e_2 \cdots e_n > (n \geqslant m)$,则序列 α 关于子序列 β 的后缀为 $< e_m'' e_{m+1} \cdots e_n >$,其中 $e_m'' = (e_m - e_m')$;

投影数据库(Projected database):设 α 为序列数据库 S 中的一个序列模式,则 α 的投影数据库为 S 中所有以 α 为前缀的序列相对于 α 的后缀,记为 $S \mid \alpha$;

投影数据库中的支持数:设 α 为序列数据库 S 中的一个序列模式,序列 β 以 α 为前缀,则 β 在 α 的投影数据库 $S \mid \alpha$ 中的支持数为 $S \mid \alpha$ 中满足条件 $\beta \subseteq \alpha.\gamma$ 的序列 γ 的个数。

2 基于 Apriori 的算法研究

Apriori 算法是关联规则挖掘的经典算法。它的思想扩展到序列挖掘中,形成了以 AprioriAll 和 GSP 为代表的经典序列挖掘算法。从文献[3]和[4]被引用的次数达数百次之多,可以看出基于 Apriori 算法的研究不少,也正因为此,下面直接引入 AprioriAll 算法和 GSP 算法进行描述、分析和总结。

2.1 AprioriAll 算法描述和分析

2.1.1 算法描述

AprioriAll 算法[1,3]:

输入:大项集阶段转换后的序列数据库

输出:所有最长序列

(1) L_1 = {large 1 - seqence};//大项集阶段得到的结果

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

(3) C_k = Candidate - generate(L_{k-1})// C_k 是从 L_{k-1} 中产生的新的候选者

(4)For each customer - sequence c in the database do// 对数据库中的每一个顾客序列 c

(5)Increment the count of all candidates in C_k that are contained in c;//对包含于 c 中 C_k 内的所有候选者计数

(6) L_k = Candidates in C_k with minimum support;// L_k 为 C_k 中满足最小支持度的候选者

(7)end

(8)Answer = Maximal Sequences in $\bigcup_k L_k$

其中候选集生成算法 Candidate - generate(L_{k-1})如下:

输入:所有的大 ($k-1$) 序列的集合 L_{k-1}

输出:候选 C_k

(1)Insert into C_k//首先进行 L_{k-1} 与 L_{k-1} 的连接运算

select p.litemset$_1$, $\cdots$, p.litemset$_{k-1}$, q.litemset$_{k-1}$

from L_{k-1} p, L_{k-1} q//p 和 q 是 L_{k-1} 中两个不同的序列串

where p.litemset$_1$ = q.litemset$_1$, $\cdots$,

p.litemset$_{k-2}$ = q.litemset$_{k-2}$;

(2)delete all sequences $c \in C_k$ such that some //对候选者进行修剪(修剪的理论依据是频繁模式集的子集也是频繁的)

($k-1$) - subsequence of c is not in L_{k-1};

2.1.2 算法分析

AprioriAll 本质上是 Apriori 思想的扩张,只是在候选集产生和生成频繁序列方面考虑序列元素有序的特点进行处理(项目集中的元素或者序列集中的元素按顺序排列,如按字典顺序或者频率高低顺序,都将有助于减轻数据挖掘的任务。文献[1]的作者毛国君于 2002 年针对关联规则挖掘问题提出了项目序列集的概念,并指出使用项目序列集可以简化挖掘的过程)。

AprioriAll 算法存在以下问题[1]:

1)缺少时间限制:用户可能需要指定序列模式的相邻元素的时间距离。例如,一个序列模式可能会发现客户在购买物品 A 后的第三年购买物品 B,我们需要的却是给定时间间隔内用户的购买意向。

2) 事务的定义过于严格:一个事务中包含在客户的一次购买行为中所购买的所有物品。可能需要指定一个滑动的时间窗口,客户在滑动时间窗口的时间段内的所有的购买行为均作为一个事务。

3)缺少分类层次:只能在项目的原始级别上进行挖掘。

2.2 GSP 算法描述和分析

2.2.1 算法描述

GSP(Generalized Sequential Patterns)算法[1,4]:

输入:大项集阶段转换后的序列数据库

输出:最大序列

(1) L_1 = {large 1 - seqence};

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

(3) C_k = GSP - generate (L_{k-1})

(4) For each customer - sequence c in the database do

(5) Increment the count of all candidates in C_k that are contained in c;

(6) L_k = Candidates in C_k with minimum support;

(7)end

(8)Answer = Maximal Sequences in $\bigcup_k L_k$

其中,GSP - generate (L_{k-1}) 分为两步:

连接阶段(Join Phase):若序列 S_1 去掉第一个项目与序列 S_2 去掉最后一个项目的所得到的序列相同,则 S_1 可以与 S_2 相连,S_2 的最后一个项目附加到 S_1 上所得的结果即为 S_1 与 S_2 相连所生成的候选序列。

剪切阶段(Prune Phase):若长度为 k 的候选序列的某个长度为 $k-1$ 的子序列的支持度小于最小支持度,则将该候选序列删除。

2.2.2 算法分析

与 AprioriAll 算法相比,GSP 算法统计较少的候选集,并且在数据转化过程中不需要事先计算频繁集。GSP 算法的时间复杂度与序列中的元素个数成线性比例关系,执行时间随数据序列中字段的增加而增加,但是增加不明显。

GSP 算法存在以下几个问题[1]:1)如果序列数据库的规模比较大,则有可能产生大量的候选序列模式;2)需要对序列数据库进行循环扫描;3)对于序列模式的长度比较长的情况,算法很难处理。

2.3 基于 Apriori 的算法小结

从以上两个算法及 AprioriSome 等算法,可以发现基于 Apriori 的候选集生成 - 测试类方法通常的思路如下:

(1)确定初始种子集(通常的作法是扫描序列数据库,把得到的长度为 1 的序列模式 L_1 作为初始的种子集)。

(2)通过某种规则连接序列以得到更长的候选序列模式,并尽可能早地剪掉那些不是序列模式的候选序列模式;然后扫描序列数据库,删除不是序列模式的候选序列模式;把生成的序列模式作为新的种子集。

(3)重复步骤(2),当找不到新的序列模式,或者没有候选集产生的时候算法中止。

3 基于模式扩展的算法研究

Pei Jian 等人不再在 Apriori 的基础上做改进工作,而是另辟蹊径,提出模式扩展[5](patter - growth)的概念,并开发了 FreeSpan,PrefixSpan 和 gSpan 等算法。

3.1 类 Apriori 方法的不足

基于 Apriori 的序列模式挖掘方法虽然减少了检索空间,但不管采取何种优化措施或实现策略,始终存在以下 3 个与生俱来的问题[5]:

(1)在大的序列数据库中,可能会产生一个相当庞大的候选序列集:由于序列中元素所有可能的置换和项目的重复,即使是一个适中的种子集,也会产生一个相当大的候选序列集。

(2)数据库的重复扫描:每扫描一次数据库,候选序列的长度就加一。通常,要找一个长度为 l 的序列模式,至少扫描数据库 l 次,当存在很长的模式时,这个扫描开销就很大。

(3)挖掘长序列模式时,候选序列的数量呈爆炸式方式增长:候选序列的数量随需要挖掘的序列模式的长度指数级增长。例如,在一个只包含一个长度为 100 的序列,最小支持数为 1(即出现的就是频繁的),产生的候选序列的数量为 $\sum_{i=1}^{100}\binom{100}{i} = 2^{100} - 1 \approx 10^{30}$。

3.2 PrefixSpan 算法的提出和描述

在许多应用中,如 DNA 分析和股市序列分析等,数据库常包含大量的序列模式,而且许多模式很长,此时有必要重新审视序列模式挖掘问题,以探索一些更加有效、易于扩展的方法。通过观察发现,基于 Apriori 算法的瓶颈在于每一步的候选集生成和测试,能否找到一种方法,既能吸取 Apriori 的灵魂又能避免或者充分减少昂贵的候选集生成和测试。顺着这个思路,Pei Jian,Han Jiawei 及 Wang Jianyong 等人基于分治和模式扩展的原理提出了模式扩展方法,代表性的算法有 FreeSpan 和 PrefixSpan,其中 PrefixSpan 采用了伪投影技术,性能比 FreeSpan 好。下面描述并分析 PrefixSpan 算法。

PrefixSpan 算法[2,5]:

输入:序列数据库 S 及最小支持度阈值 min_sup

输出:所有的序列模式

方法:调用子程序 PrefixSpan(<>, 0, S)

其中子程序 PrefixSpan($\alpha, L, S|\alpha$) 描述如下:

(1)扫描 $S|\alpha$,找到满足下述要求的长度为 1 的序列模式 b:

1) b 可以添加到 α 的最后一个元素中并为序列模式;

2) <b>可以作为 α 的最后一个元素并为序列模式。

(2)对每个生成的序列模式 b,将 b 添加到 α 形成序列模式 α',并输出 α';

(3)对每个 α',构造 α' 的投影数据库 $S|\alpha'$,并调用子程序 PrefixSpan(α', $L+1$, $S|\alpha'$)。子程序参数说明:α 为一个序列模式;L 为序列模式 α 的长度;$S|\alpha$ 为($的投影数据库($当 α 为空时,$S|\alpha$ 就是 S)。

3.3 PrefixSpan 算法分析

PrefixSpan 算法分为 4 个步骤:a. 找出所有长度为 1 的序列模式;b. 根据前缀(prefix)分割检索空间;c. 找序列模式的子集;d. 步骤 3 递归进行,挖掘过程中产生的模式集合就是序列模式集。该算法不再产生候选序列,当然也就不要测试了,其主要开销集中在投影数据库的构建过程。

经过测试比较[5],PrefixSpan 算法性能比基于 Apriori 的算法(GSP 和 SPADE)明显要好,原因在于:

1) 模式扩展方式不生成候选序列。PrefixSpan 是一个基于模式扩展的方法,就象 FP - growth 一样。用传统

(下转第 10 页)

置过程[7]。在这个过程中,每个服务工厂拥有一个惟一的名称,并使用 OGSI 定义的 factory_service.wsdl。真实的实现工厂类使用 createClass 进行定义,接受类名称和程序集名称。服务工厂的处理程序列表在 messageHandler 属性中指定。服务实例的处理程序列表在 createMessageHandlers 属性中指定。并且,服务必须分别使用 createSchemaPath 和 createClass 参数提供它的 wsdl 和类名称。

```
< service name = samples/weather/WeatherForecastFactoryService
>
<parameter name = "schemaPath" value = "schema/core/factory/
factory_service.wsdl">
<parameter name = "class" value = "UVa.Grid.OGSISamples.Fac-
tory.BasicFactoryService,OGSISampleServices.dll">
<parameter name = "messageHandlers"
value = "UVa.Grid.SoapMessageHandlerLib.SoapMessageHandler,
SoapMessageHandlerLib.dll">
<parameter name = "createSchemaPath"
value = "schema/weather/WeatherForecastService.wsdl">
<parameter name = "createClass"
value = "UVa.Grid.SampleServices.WeatherForecastService,Sam-
pleServices.dll">
<parameter name = "createMessageHandlers"
value = "UVa.Grid.SoapMessageHandlerLib.SoapMessageHandler,
SoapMessageHandlerLib.dll">
</service>
```

图 3 网格服务配置代码

6 结束语

文中涉及的在.NET 框架下基于 OGSI.NET 的网格计算技术,结合实例能灵活、高效地应用于网格计算环境中。为网格计算的研究提供了一个新的途径。同时仍然还有很多问题需要进一步说明,包括持久性、可伸缩性和 GT3 与 OGSI.NET 之间的互操作性等。

参考文献:

[1] 都志辉,陈 渝,刘 鹏.网格计算[M].北京:清华大学出版社,2002.

[2] Globus Toolkit 3.0.2. The Globus Alliance. Available[EB/OL]. http://www.globus.org/toolkit/gt3-factsheet.html, 2003.

[3] Microsoft Corporation. .NET Framework[EB/OL]. http://www.microsoft.com/net/,2003.

[4] OGSI.NET Programmer's Reference[EB/OL]. http://www.cs.virginia.edu/humphrey/GCG/ogsi.net.html,2004.

[5] Tuecke S, Czajkowski C, Foster I, et al. Grid Service Specification-Draft 11/4/02. OGSI Working Group, Global Grid Forum[EB/OL]. http://www.ggf.org/ogsi-wg.,2002.

[6] Joseph J, Fellensein C. 网格计算[M]. 战晓苏,张少华,译. 北京:清华大学出版社,2005.

[7] OGSI.NET: OGSI-compliance on the .NET Framework [EB/OL]. http://www.cs.virginia.edu/humphrey/GCG/ogsi.net.html,2004-12.

(上接第 6 页)

的候选集生成-测试方法来处理一个比较小的数据库(例如只有 10k 的序列),需要相当多的时间来生成和测试大量的候选序列模式。

2) 基于投影的分治是数据缩减(reduction)的有效方法。序列模式 α 的投影数据库包含且仅包含用来挖掘那些由 α 扩展得到的模式的必需信息,投影数据库的大小随着挖掘过程向更长的序列模式进行而迅速缩减。

3) PrefixSpan 需要的内存空间相对稳定。原因在于它采用分治的方法,不生成候选集。而 GSP 和 SPADE,当支持度阈值(support threshold)降低时,由于需要容纳大量候选序列,需要相当数量的内存。

基于模式扩展的方法,可以应用到多层次、多维数的序列模式中,也可以挖掘其他结构化的模式。

4 结论与展望

序列模式挖掘是当前数据挖掘领域中一个较新而且非常活跃的研究分支,有着广泛的应用价值。文章在介绍了序列模式挖掘的相关概念后,对两类序列模式挖掘的几个经典的算法进行了描述和分析,不难发现,基于模式扩展的方法是个前途很好的发展方向。模式扩展方法还有很多工作要做,如闭合集挖掘、在特定领域的针对性研究等等。

参考文献:

[1] 毛国君,段立娟,王 实,等.数据挖掘原理与算法[M].北京:清华大学出版社,2005.

[2] Han Jiawei, Pei Jian, Yan xifeng. From Sequential Pattern Mining to Structured Pattern Mining: A Pattern-Growth Approach[J]. Journal of Computer Science and Technology, 2004,19:257-279.

[3] Agrawal R, Srikant R. Mining sequential pattern[A]. Proc 1995 Int Conf Data Engineering (ICDE's95)[C]. Chinese Taipei:[s.n.],1995.3-14.

[4] Srikant R, Agrawal R. Mining sequential pattern: Generalizations and performance improvements[A]. Proc 5th Int Conf Extending Database Technology(EDBT's96)[C]. Avignon, France:[s.n.],1996.3-17.

[5] Pei Jian, Han Jia-Wei, Mortazavi-Asl B, et al. Mining sequential pattern by Pattern-Growth: The PrefixSpan Approach[J]. IEEE TKDE,2004,16(10):9-13.