

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法





■ 随机时间序列预测方法

- 通过建立随机模型，对随机时间序列进行分析，可以预测未来值。
- 若时间序列是平稳的，可以用自回归 (Auto Regressive, 简称AR) 模型、移动回归模型 (Moving Average, 简称MA) 或自回归移动平均 (Auto Regressive Moving Average, 简称ARMA) 模型进行分析预测。



基于ARMA模型的序列匹配方法

- ARMA模型（特别是其中的AR模型）是时序方法中最基本的、实际应用最广的时序模型。早在1927年，G. U. Yule就提出了AR模型，此后，AR模型逐步发展为ARMA模型、多维ARMA模型。ARMA通常被广泛用于预测。由于ARMA模型是一个**信息的凝聚器**，可将系统的特性与系统状态的所有信息凝聚在其中，因而它也可以用于时间序列的匹配。

■ 1. ARMA模型

对于平稳、正态、零均值的时序 $X = \{x_t | t = 0, 1, 2, \dots, n-1\}$ ，若 X 在 t 时刻的取值不仅与其前 n 步的各个值 $x_{t-1}, x_{t-2}, \dots, x_{t-n}$ 有关，而且还与前 m 步的各个干扰 $\alpha_{t-1}, \alpha_{t-2}, \dots, \alpha_{t-m}$ ($n, m=1, 2, \dots$)，则按多元线性回归的思想，可得到最一般的ARMA (n, m) 模型：

$$x_t = \sum_{i=1}^n \varphi_i x_{t-i} - \sum_{j=1}^m \theta_j \alpha_{t-j} + \alpha_t$$

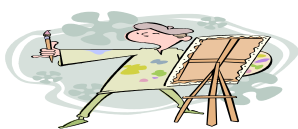
其中 $\alpha_t \sim NID(0, \delta_a^2)$

第六章 时间序列和序列模式挖掘

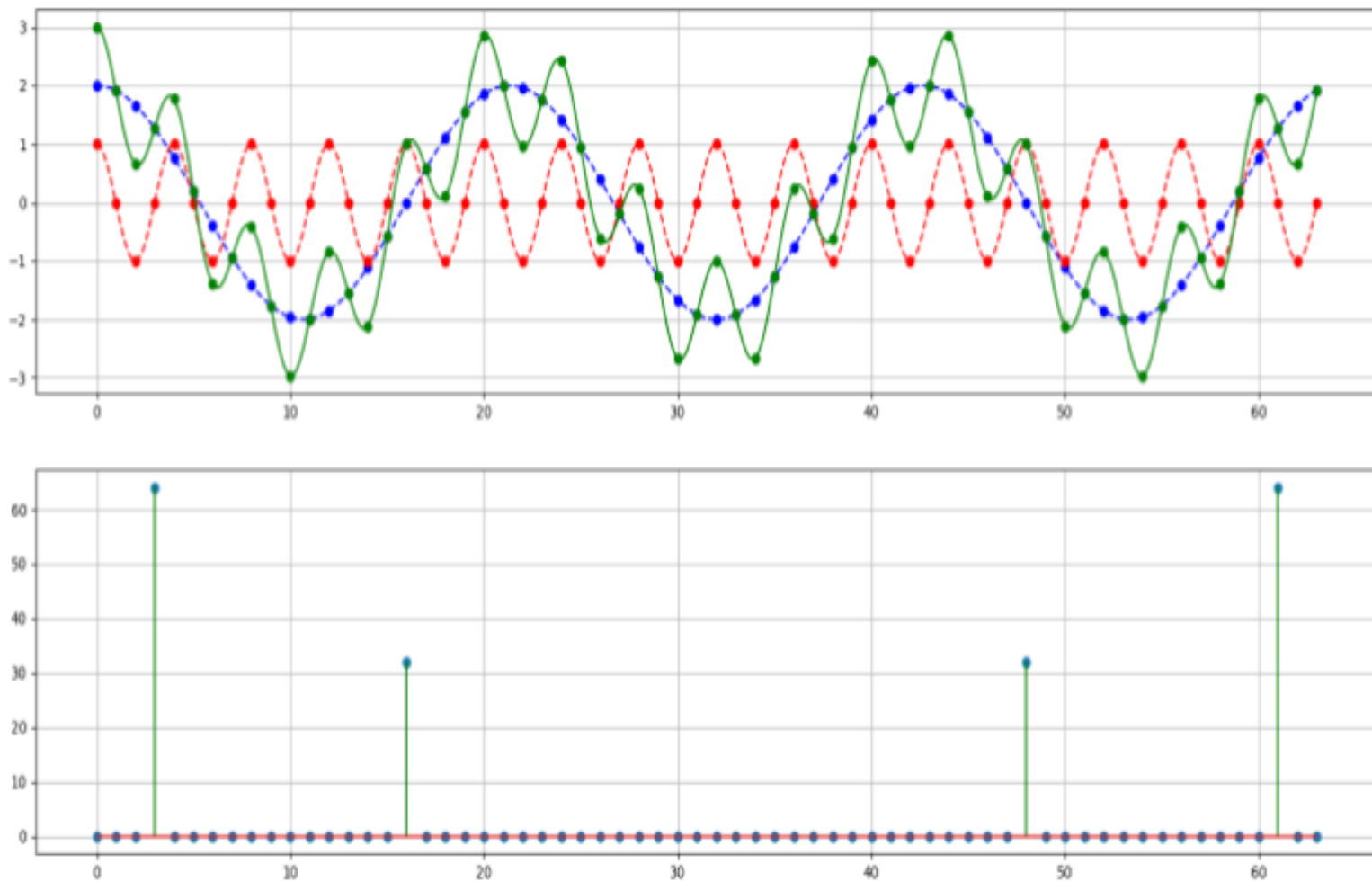
内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法





为什么要做傅里叶变换





$$\frac{4 \sin \theta}{\pi}$$



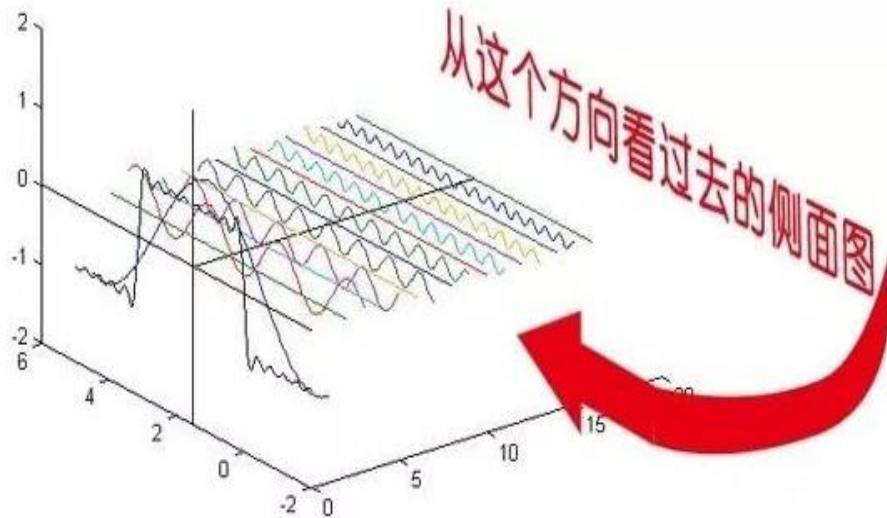
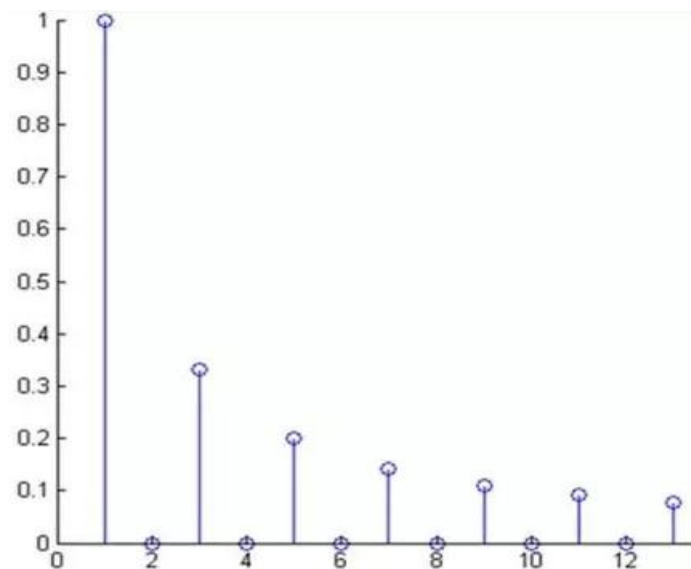
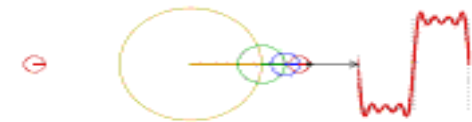
$$\frac{4 \sin 3\theta}{3\pi}$$

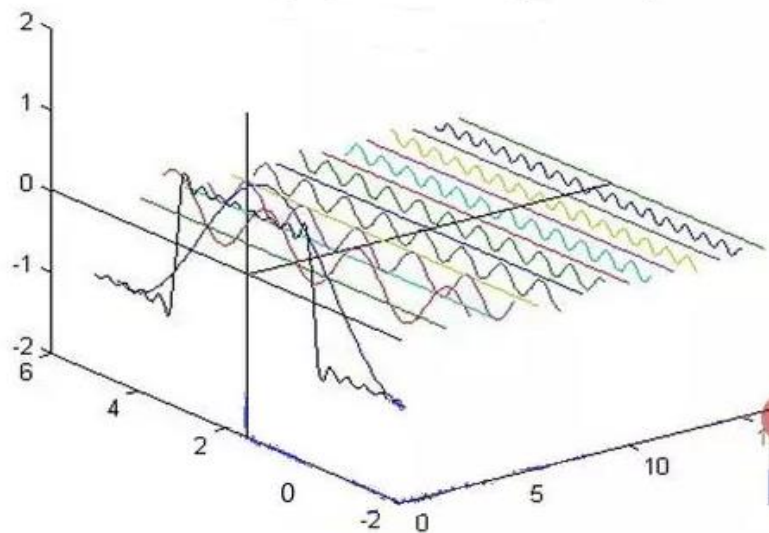
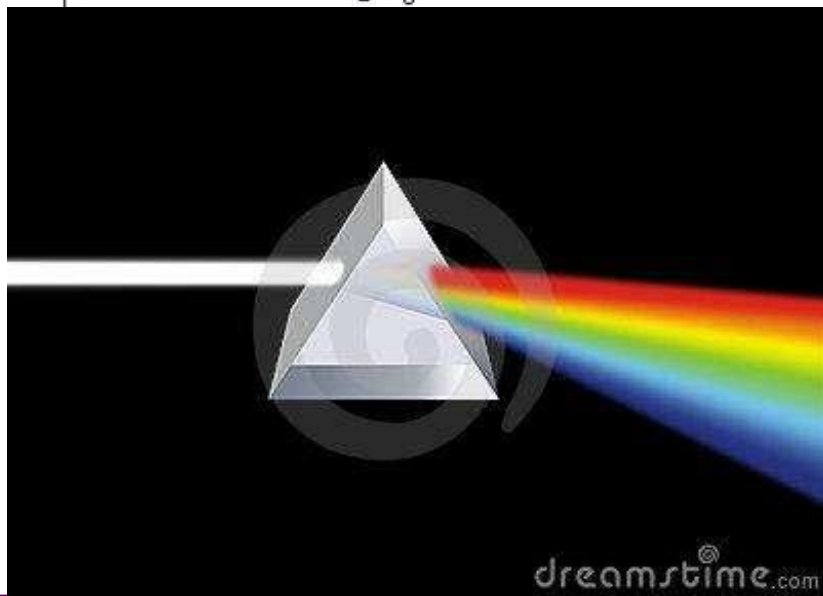
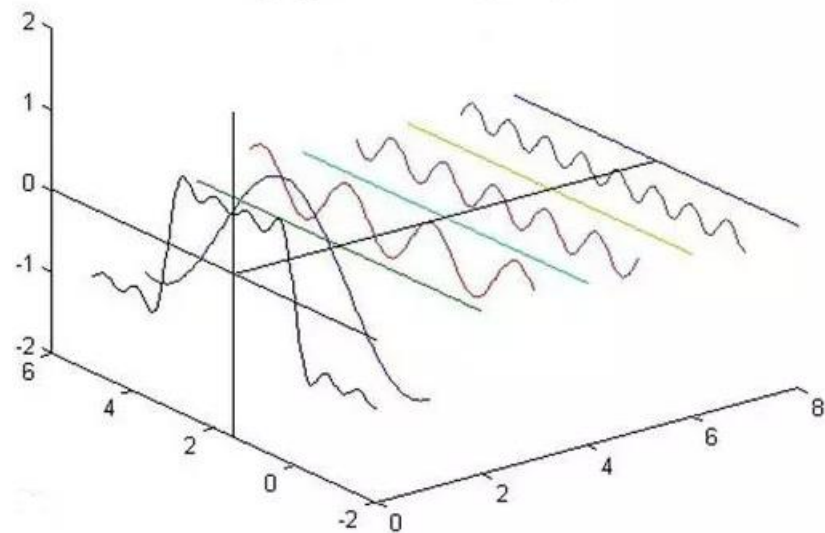
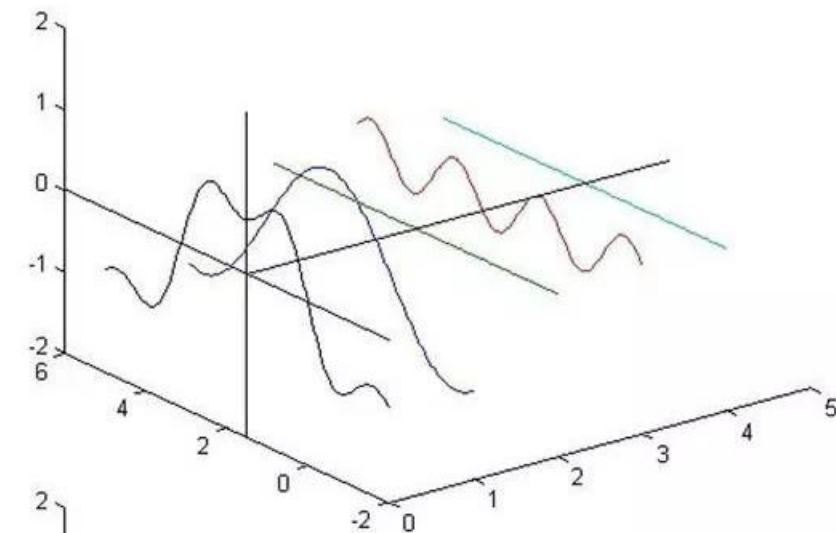


$$\frac{4 \sin 5\theta}{5\pi}$$



$$\frac{4 \sin 7\theta}{7\pi}$$







为什么要做傅里叶变换

- 离散傅里叶变换 (DFT) 的公式:

$$\begin{aligned} X(f) &= \sum_{t=0}^{N-1} x(t) e^{i * \frac{f}{N} * 2\pi * t}, t, f = 0, 1, \dots, N-1 \\ &= \sum_{t=0}^{N-1} x(t) \cos\left(\frac{f}{N} * 2\pi * t\right) + i * x(t) \sin\left(\frac{f}{N} * 2\pi * t\right) \end{aligned}$$



为什么要做傅里叶变换

大学生破译周鸿祎手机号 李开复放“橄榄枝”

2012年08月31日21:32

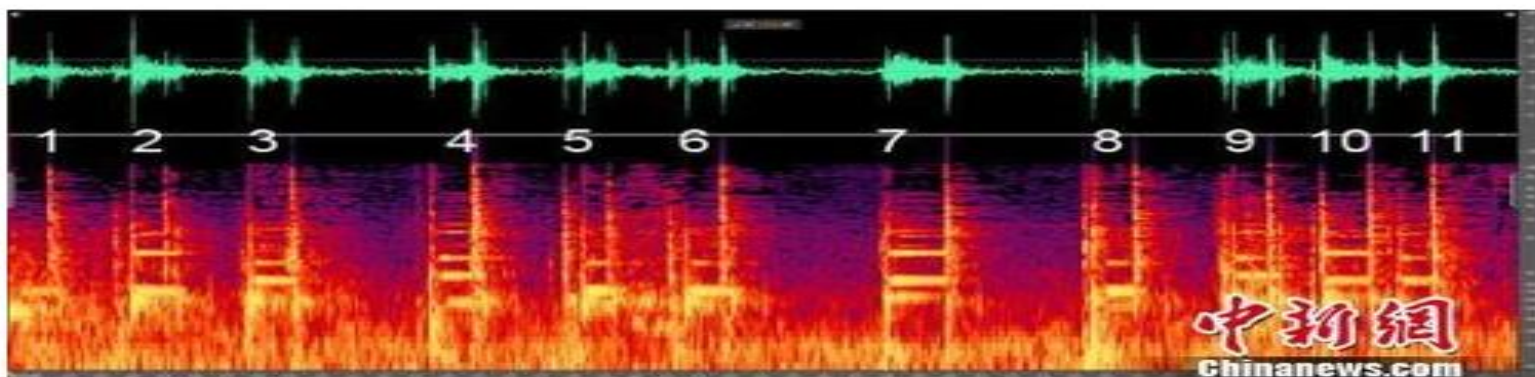
中国新闻网[微博]

孟祥磊

我要评论(0)

字号：T | T

[导读] 30日，南京大学学生刘靖康突发奇想，利用视频中记者采访时的拨号声音还原出了手机号码。



号码按键声波图 刘靖康供图 摄



第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法





为了方便讨论，我们首先给出一些符号来表示序列及序列的相似性：

- $X = \{x_t | t = 0, 1, 2, \dots, n-1\}$ 表示一个**序列**；
- $\text{Len}(X)$ 表示序列X的**长度**；
- $\text{First}(X)$ 表示序列X的**第一个元素**；
- $\text{Last}(X)$ 表示序列X的**最后一个元素**；
- $X[i]$ 表示X在i时刻的取值, $X[i] = x_i$ ；
- 序列上元素之间的“**<**”**关系**，在序列X上，如果 $i < j$ ，那么 $X[i] < X[j]$ ；
- 本文用 x_s 表示X的**子序列**，如果序列X有k个子序列，则把这些子序列分别表示为 $X_{s_1}, X_{s_2}, \dots, X_{s_k}$ 。
- 子序列间的<关系， X_{s_i}, X_{s_j} 为X的子序列，如果 $\text{First}(X_{s_i}) < \text{First}(X_{s_j})$ ，则称 $X_{s_i} < X_{s_j}$ 。
- **子序列重叠** (Overlap) ，假定 X_{s_1}, X_{s_2} 为X的两个子序列，如果 $\text{First}(X_{s_1}) \leq \text{First}(X_{s_2}) \leq \text{Last}(X_{s_1})$ 或 $\text{First}(X_{s_2}) \leq \text{First}(X_{s_1}) \leq \text{Last}(X_{s_2})$ 成立，则 X_{s_1} 与 X_{s_2} 重叠。



- 一般地，相似性匹配可分为两类：
- **完全匹配** (Whole Matching)。给定 N 个序列 Y_1, Y_2, \dots, Y_n 和一个查询序列 X ，这些序列**有相同的长度**，如果存在 $D(X, Y_i) \leq \varepsilon$ ，那么我们称 X 与 Y_i 完全匹配。
- **子序列匹配** (Subsequence Matching)。给定 N 个具有任意长度的序列 Y_1, Y_2, \dots, Y_n 和一个查询序列 X 以及参数 ε 。子序列匹配就是在 $Y_i (1 \leq i \leq N)$ 上找到**某个子序列**，使这个子序列与 X 之间的距离小于等于 ε 。



基于规范变换的查找方法

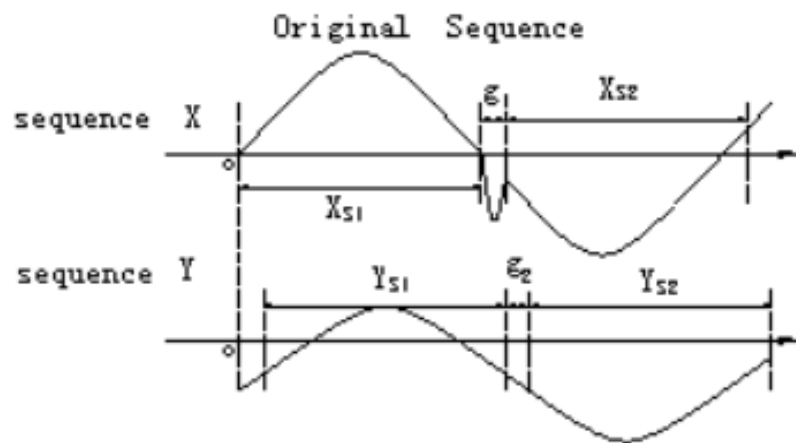


图6-3 序列X与Y

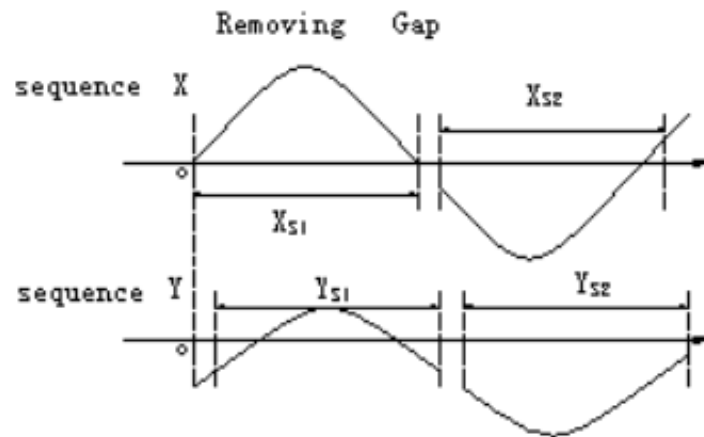


图6-4 去掉Gap后的序列X与Y

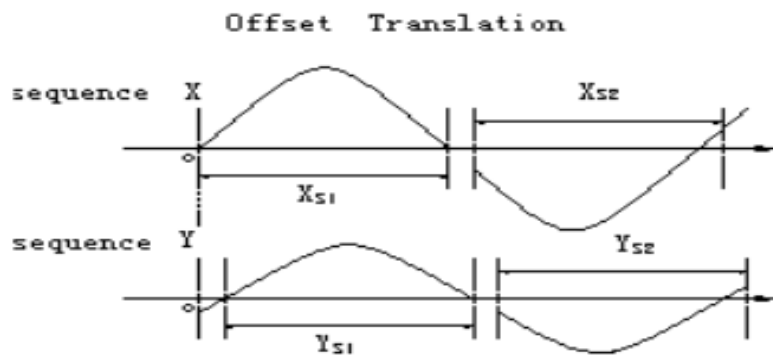


图6-5 偏移变换后的序列X与Y

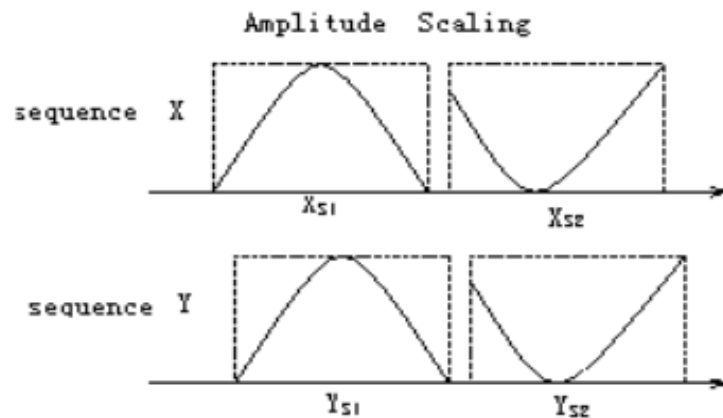


图6-6 幅度缩放后的序列X与Y



定义6-1 如果序列 X 所包含的不相互重叠的子序列和 $X_{S_1}, X_{S_2}, \dots, X_{S_m}$ 与 Y 所包含的不相互重叠的子序列 Y_{S_1}, \dots, Y_{S_m} 满足如下3个条件, 可以认为 X 与 Y 是 ξ -similar:

- (1) 对任意的 $1 \leq i < j \leq m$, $X_{S_i} < X_{S_j}$ 与 $Y_{S_i} < Y_{S_j}$ 都成立;
- (2) 存在一些比例因子 λ 和一些偏移 θ 使得下式成立:

$$\forall_{i=1}^m \theta(\lambda(X_{S_i})) \approx Y_{S_i}$$

其中“ \approx ”表示两个子序列相似, 表示对子序列以 λ 为比例因子进行缩放, 按照 θ 进行偏移变换。

- (3) 给定 ξ , 下式成立

$$\frac{\sum_{i=1}^m \text{Len}(X_{S_i}) + \sum_{i=1}^m \text{Len}(Y_{S_i})}{\text{Len}(X) + \text{Len}(Y)} \geq \xi$$

这个定义意味着如果序列 X 与 Y 匹配的长度之和与这两个序列的长度之和的比值不小于 ξ , 则认为序列 X 与 Y 是 ξ -similar。这样事先给定的阈值 ξ , 就找到一个序列相似的评价函数。

当被比较的序列 X 与 Y 的长度非常悬殊, 我们可以用如下函数来评价相似度:

$$\frac{\sum_{i=1}^m \text{Len}(X_{S_i}) + \sum_{i=1}^m \text{Len}(Y_{S_i})}{2 \times \min(\text{Len}(X), \text{Len}(Y))} \geq \xi$$



基于规范变换的查找方法

Agrawal把X与Y的相似性比较问题分为三个子问题：原子序列匹配；窗口缝合；子序列排序。

1. 原子序列匹配

- 与基于离散傅立叶变换的时间序列查找方法相同，原子序列匹配（Atomic Matching）也采用了滑动窗口技术。根据用户事先给定的一个 ω （通常为5~20），将序列映射为若干长度为 ω 的窗口，然后对这些窗口进行幅度缩放与偏移变换。
- 我们首先讨论窗口中点的标准化问题。通过下面的转换，可以将窗口内不规范的点转换成标准点：

$$\tilde{W}[i] = \frac{W[i] - \frac{W_{\min} + W_{\max}}{2}}{\frac{W_{\max} - W_{\min}}{2}}$$

其中 $W[i]$ 表示窗口中第 i 个点的值， W_{\max} 、 W_{\min} 分别表示窗口内所有点的最大值与最小值。通过上面的公式使得窗口内的每个点的值落在 $(-1, +1)$ 之间。把这种标准化后的窗口称为原子。



基于规范变换的查找方法(续)

2. 窗口缝合

窗口缝合 (Window Stitching) 即子序列匹配, 其主要任务是将相似的原子连接起来形成比较长的彼此相似的子序列。

$\tilde{X}_1, \dots, \tilde{X}_m$ 和 $\tilde{Y}_1, \dots, \tilde{Y}_m$ 分别为 X 与 Y 上 m 个标准化后的原子, 缝合

$\tilde{X}_1, \dots, \tilde{X}_m$ 和 $\tilde{Y}_1, \dots, \tilde{Y}_m$ 使它们形成一对相似的子序列的条件如下:

- (1) 对于任意的 i 都有 \tilde{X}_i 与 \tilde{Y}_i 相似;
- (2) 对于任何 $j > i$, $First(X_{wi}) \leq First(X_{wj})$, $First(Y_{wi}) \leq First(Y_{wj})$;
- (3) 对任何 $i > 1$, 如果 \tilde{X}_i 不与 \tilde{X}_{i-1} 重叠, 且 \tilde{X}_i 与 \tilde{X}_{i-1} 之间的 Gap 小于等于 γ , 同时 Y 也满足这个条件; 如果 \tilde{X}_i 与 \tilde{X}_{i-1} 重叠, 重叠长度为 d , \tilde{Y}_i 与 \tilde{Y}_{i-1} 也重叠且重叠长度也为 d 。
- (4) X 上的每个窗口进行标准化时所用的比例因子大致相同, Y 上的每个窗口进行标准化时所用的比例因子也大致相同。



基于规范变换的查找方法(续)

3. 子序列排序

- 通过对窗口缝合得到一些相似的子序列，再对这些子序列排序 (Subsequence Ordering)，则可以找到两个彼此匹配的序列。
- 子序列排序的主要任务是从没有重叠的子序列匹配中找出匹配得最长的那些序列。
- 如果把所有的相似的原子对看作图论中的顶点，两个窗口的缝合看作两个顶点之间的边的话，那么从起点到终点有多条路经，子序列排序就是寻找最长路径。


第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法



美国国立生物技术信息中心



NCBI Resources How To Sign in to NCBI

NCBI National Center for Biotechnology Information

All Databases Search

NCBI Home

Resource List (A-Z)

All Resources

Chemicals & Bioassays

Data & Software

DNA & RNA

Domains & Structures

Genes & Expression

Genetics & Medicine

Genomes & Maps

Homology

Literature

Proteins

Sequence Analysis

Taxonomy

Training & Tutorials

Variation

Welcome to NCBI

The National Center for Biotechnology Information advances science and health by providing access to biomedical and genomic information.

[About the NCBI](#) | [Mission](#) | [Organization](#) | [Research](#) | [NCBI News](#)

Get Started

- [Tools](#): Analyze data using NCBI software
- [Downloads](#): Get NCBI data or software
- [How To's](#): Learn how to accomplish specific tasks at NCBI
- [Submissions](#): Submit data to GenBank or other NCBI databases

Genetic Testing Registry

A portal to clinical genetics resources with detailed information about genetic tests and laboratories.

GO

1 2 3 4 5 6 7 8

Popular Resources

- PubMed
- Bookshelf
- PubMed Central
- PubMed Health
- BLAST**
- Nucleotide
- Genome
- SNP
- Gene
- Protein
- PubChem

NCBI Announcements

GenBank release 204.0 is now available via FTP
Oct 22, 2014

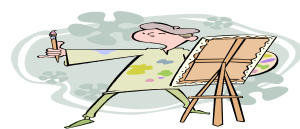
Release 204.0 (10/20/2014) has
430,000,000 1,000 2014

dbSNP human Build 142 released
Oct 17, 2014

dbSNP human Build 142, based on the GRCh38 and GRCh37.p13 assemblies, is
1,000 1,000 2014

Amazon Web Services (AWS) Marketplace provides the easiest way to start an NCBI BLAST instance
Oct 16, 2014

点击
进入



美国国立生物技术信息中心

Solanum lycopersicum chromosome chr7, complete genome

Sequence ID: emb|HG975519.1 Length: 65272350 Number of Matches: 1

Range 1: 63965831 to 63965973 [GenBank](#) [Graphics](#) 序列编号 匹配序列长度 对比空白 Next Match Previous Match

Score	Expect	Identities	Gaps	Strand
228 bits(123)	2e-56	137/143(96%)	3/143(2%)	Plus/Minus

Query 21 匹配范围 GTACTAATGGA--TGTCGAAATTGGAAATTGAGA-GGCAAACGGGGAAGAAAGTCCAGC 77

Sbjct 63965973 GTACTAATGGAATGTGTGCGAAATTGGAAATTGAGA-GGCAAACGGGGAAGAAAGTCCAAC 63965914

Query 78 输入序列被随机搜索出来的概率, 该值越小越好 TTTTCTCTCTCTTTCCATATAG 137

Sbjct 63965913 TAAGCTGAGTTCAGTGAGATTTTCTAATTGGACCAGAGACTTGTAAATCACTCACTACT 63965854

Query 138 TCTTTCTCTCTCTTTCCATATAG 160

Sbjct 63965853 TCTTTCTCTCTCTTTCCATATAG 63965831

相似序列, 即输入序列和搜索到序列的匹配率



BLAST®

Basic Local Alignment Search Tool

Home

Recent Results

Saved Strategies

Help

My NCBI

Sign In Register

NCBI/BLAST/blastn suite/Formatting Results - 4EXV87RA014

[Edit and Resubmit](#) [Save Search Strategies](#) [Formatting options](#) [Download](#)

[YouTube](#) [How to read this page](#) [Blast report description](#)

ZB04091969(MALINGSHU)-A68-M13+_D09

RID [4EXV87RA014](#) (Expires on 10-23 17:22 pm)

Query ID [ldj7191](#)

Description ZB04091969(MALINGSHU)-A68-M13+_D09

Molecule type nucleic acid

Query Length 333

Database Name nr

Description Nucleotide collection (nt)

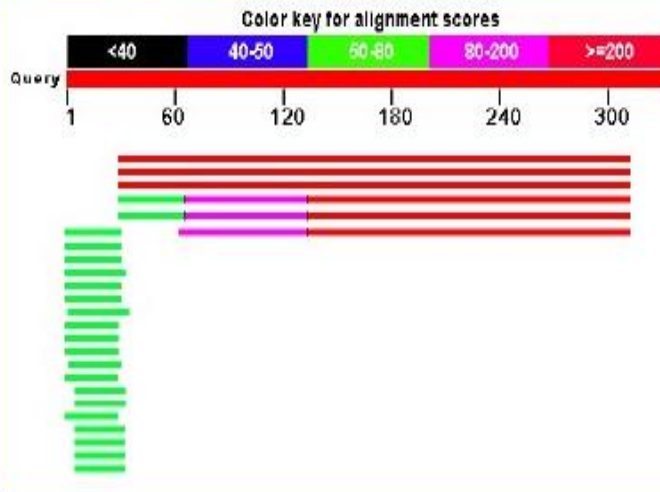
Program BLASTN 2.2.30+ [Citation](#)

Other reports: [Search Summary](#) [Taxonomy reports](#) [Distance tree of results](#)

Graphic Summary

Distribution of 30 Blast Hits on the Query Sequence

Mouse over to see the define, click to show alignments





- **序列挖掘**或称序列模式挖掘，是指从序列数据库中发现蕴涵的序列模式。时间序列分析和序列模式挖掘有许多相似之处，在应用范畴、技术方法等方面也有很大的重合度。但是，序列挖掘一般是指相对时间或者其他顺序出现的序列的高频率子序列的发现，典型的应用还是限于离散型的序列。
- 近年来序列模式挖掘已经成为数据挖掘的一个重要方面，其应用范围也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。



序列挖掘—基本概念

- **定义6-3** 一个序列 (Sequence) 是项集的有序表,记为 $\alpha = \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_n$, 其中每个 α_i 是一个项集 (Itemset)。一个序列的长度 (Length) 是它所包含的项集。具有 k 长度的序列称为 k -序列。
- **定义6-4** 设序列 $\alpha = \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_n$, 序列 $\beta = \beta_1 \rightarrow \beta_2 \rightarrow \dots \rightarrow \beta_m$ 。若存在整数 $i_1 < i_2 < \dots < i_n$, 使得
$$\alpha_1 < \beta_{i_1}, \alpha_2 < \beta_{i_2}, \dots, \alpha_n < \beta_{i_n},$$
则称序列 α 是序列 β 的子序列, 或序列 β 包含序列 α 。在一组序列中,如果某序列 α 不包含其他任何序列中, 则称 α 是该组中**最长序列** (Maximal sequence)。
- **定义6-5** 给定序列 S , 序列数据库 D_T , 序列 S 的**支持度** (Support) 是指 S 在 D_T 中相对于整个数据库元组而言所包含 S 的元组出现的百分比。支持度大于最小支持度 (min-sup) 的 k -序列, 称为 D_T 上的频繁 k -序列。



序列挖掘—数据源的形式

表6-1带交易时间的交易数据源示例

客户号 (Cust_id)	交易时间 (Tran_time)	物品 (Item)
1	June 25'99	30
1	June 30'99	90
2	June 10'99	10, 20
2	June 15'99	30
2	June 20'99	40, 60, 70
3	June 25'99	30, 50, 70
4	June 25'99	30
4	June 30'99	40, 70
4	July 25'99	90
5	June 12'99	90

带交易时间的交易数据库的典型形式是包含客户号 (Customer-id)、交易时间 (Transaction-Time) 以及在交易中购买的项 (Item) 等的交易记录表。表6-1给出了一个这样数据表的示例。这样的数据源需要进行形式化的整理，其中一个理想的预处理方法就是转换成顾客序列，即将一个顾客的交易按交易时间排序成项目序列。例如表6-2给出了表6-1对应的所有顾客序列表。

表6-2顾客序列表示例

客户号 (Cust_id)	顾客序列 (Customer Sequence)
1	< (30) (90) >
2	< (10, 20) (30) (40, 60, 70) >
3	< (30, 50, 70) >
4	< (30) (40, 70) ((90)) >
5	< (90) >



序列挖掘—数据源的形式(续)

操作系统及其系统进程调用是评价系统安全性的一个重要方面。通过对正常调用序列的学习可以预测随后发生的系统调用序列、发现异常的调用。因此序列挖掘是从系统调用等操作系统审计数据中发现有用模式的一个理想的技术。表6-3给出了一个系统调用数据表示意，它是利用数据挖掘技术进行操作系统安全性审计的常用数据源。

表6-3系统进程调用数据示例

进程号 (Pro_id)	调用时间 (Call_time)	调用号 (Call_id)
744	04: 01: 10: 30	23
744	04: 01: 10: 31	14
1069	04: 01: 10: 32	4
9	04: 01: 10: 34	24
1069	04: 01: 10: 35	5
744	04: 01: 10: 38	81
1069	04: 01: 10: 39	62
9	04: 01: 10: 40	16
-1		

表6-4系统调用序列数据表示例

进程号 (Pro_id)	调用序列 (Call_sequence)
744	< (23, 14, 81) >
1069	< (14, 24, 16) >
9	< (4, 5, 62) >



序列模式挖掘的一般步骤

我们分五个具体阶段来介绍基于上面概念发现序列模式的方法。这些步骤分别是**排序阶段**、**大项集阶段**、**转换阶段**、**序列阶段**以及**选最大阶段**。

1. 排序阶段

■对数据库进行排序 (Sort)，排序的结果将原始的数据库转换成序列数据库（比较实际可能需要其他的预处理手段来辅助进行）。例如，上面介绍的交易数据库，如果以客户号 (Cust_id) 和交易时间 (trans-time) 进行排序，那么在通过对同一客户的事务进行合并就可以得到对应的序列数据库。

2. 大项集阶段

■这个阶段要找出所有频繁项集（即大项集）组成的集合 L 。实际上，也同步得到所有大1-序列组成的集合，即 $\{ \langle I \rangle \mid I \in L \}$ 。

■在上面表6-2给出的顾客序列数据库中，假设支持数为2，则大项集分别是 (30)，(40)，(70)，(40, 70) 和 (90)。实际操作中，经常将大项集被映射成连续的整数。例如，上面得到的大项集映射成表6-6对应的整数。当然，这样的映射纯粹是为了处理的方便和高效。

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40, 70)	4
(90)	5



序列模式挖掘的一般步骤(续)

3. 转换阶段

- 在寻找序列模式的过程中，要不断地进行检测一个给定的大序列集合是否包含于一个客户序列中。

表6-7给出了表6-2数据库经过转换后的数据库。比如，在对ID号为2的客户序列进行转换的时候，交易(10, 20)被剔除了，因为它并没有包含任何大项集；交易(40, 60, 70)则被大项集的集合{(40), (70), (40, 70)}代替。

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40, 70)	4
(90)	5

4. 序列阶段

- 利用转换后的数据库寻找频繁的序列，即大序列 (Large Sequence)。

5. 选最大阶段

- 在大序列集中找出最长序列 (Maximal Sequences)。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用：时间序列挖掘的概念
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法





- 在AprioriAll之前先说一下Apriori思想：如果某个项集是频繁的，那么它的所有子集也是频繁的。
- **AprioriAll**本质上是Apriori思想的扩张，只是在产生候选序列和频繁序列方面**考虑序列元素有序**的特点，将项集的处理改为序列的处理。
- AprioriAll算法将序列模式挖掘过程分为**5个具体阶段**，即排序阶段、找频繁项集阶段、转换阶段、产生频繁序列阶段、最大化阶段。



- AprioriAll也是如此：
- 1) **排序阶段**。数据库D以客户号为主键，交易时间为次键进行排序。这个阶段将原来的事务数据库转换成由客户序列组成的数据库。
- 2) **频繁项集阶段**。找出所有频繁项集组成的集合L。也同步得到所有频繁1-序列组成的集合。
- 3) **转换阶段**。在找序列模式的过程中，要不断地进行检测一个给定的频繁集是否包含于一个客户序列中。
- 4) **序列阶段**。利用已知的频繁集的集合来找到所需的序列。类似于关联的Apriori算法。
- 5) **最大化阶段**。在频繁序列模式集合中持出最大频繁序列模式集合



1.排序阶段

- 对原始数据表按客户号（作为主关键字）和交易时间（作为次关键字）进行排序，排序后通过对同一客户的事件进行合并得到对应的序列数据库，minsupport=40%。

交易数据库D

客户号SID	交易时间TID	商品列表（事件）
s_1	6月25日	30
	6月30日	80
s_2	6月10日	10, 20
	6月15日	30
	6月20日	40, 60, 70
s_3	6月25日	30, 50, 70
s_4	6月25日	30
	6月30日	40, 70
	7月25日	80
s_5	6月12日	80



1.排序阶段

- 对原始数据表按客户号（作为主关键字）和交易时间（作为次关键字）进行排序，排序后通过对同一客户的事件进行合并得到对应的序列数据库，minsupport=40%。

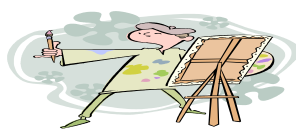
交易数据库D

客户号SID	交易时间TID	商品列表（事件）
s_1	6月25日	30
	6月30日	80
s_2	6月10日	10, 20
	6月15日	30
	6月20日	40, 60, 70
s_3	6月25日	30, 50, 70
s_4	6月25日	30
	6月30日	40, 70
	7月25日	80
s_5	6月12日	80



序列数据库S

客户号	客户序列
s_1	<{30}, {80}>
s_2	<{10, 20}, {30}, {40, 60, 70}>
s_3	<{30, 50, 70}>
s_4	<{30}, {40, 70}, {80}>
s_5	<{80}>



客户号	客户序列
s_1	$\langle \{30\}, \{80\} \rangle$
s_2	$\langle \{10, 20\}, \{30\}, \{40, 60, 70\} \rangle$
s_3	$\langle \{30, 50, 70\} \rangle$
s_4	$\langle \{30\}, \{40, 70\}, \{80\} \rangle$
s_5	$\langle \{80\} \rangle$

1-项集	计数
{10}	1
{20}	1
{30}	4
{40}	2
{50}	1
{60}	1
{70}	3
{80}	3

删除少于 min_sup 的项集

1-项集	计数
{30}	4
{40}	2
{70}	3
{80}	3

自连接产生候选 2-项集

2-项集	计数
{40, 70}	2

删除少于 min_sup 的项集

2-项集	计数
{30, 40}	0
{30, 70}	1
{30, 80}	0
{40, 70}	2
{40, 80}	0
{70, 80}	0

https://blog.csdn.net/w_z_y1997



频繁项集阶段

- 然后将频繁1-项集**映射**成连续的整数。例如，将上面得到的L1映射成表6.4对应的整数。由于比较频繁项集花费一定时间，这样做后可以减少检查一个序列是否被包含于一个客户序列中的时间，从而使处理过程方便且高效。

频繁项集	映射成整数
{30}	1
{40}	2
{70}	3
{40, 70}	4
{80}	5

https://blog.csdn.net/w_z_y1997



3.转换阶段

- 在寻找序列模式的过程中，要不断地进行检测一个给定的大序列集合是否包含于一个客户序列中。为此做这样的转换：
- 每个事件被包含于该事件中所有频繁项集替换。
- 如果一个事件不包含任何频繁项集，则将其删除。
- 如果一个客户序列不包含任何频繁项集，则将该序列删除。
- 转换后，一个客户序列由一个频繁项集组成的集合所取代。每个频繁项集的集合表示为 $\{e_1, e_2, \dots, e_k\}$ ，其中 e_i ($1 \leq i \leq k$) 表示一个频繁项集。

1-项集	计数
{30}	4
{40}	2
{70}	3
{80}	3

2-项集	计数
{40, 70}	2

频繁项集	映射成整数
{30}	1
{40}	2
{70}	3
{40, 70}	4
{80}	5

客户号	原客户序列	新客户序列	映射后
s_1	$\langle \{30\}, \{80\} \rangle$	$\langle \{30\}, \{80\} \rangle$	$\langle \{1\}, \{5\} \rangle$
s_2	$\langle \{10, 20\}, \{30\}, \{40, 60, 70\} \rangle$	$\langle \{30\}, \{\{40\}, \{70\}, \{40, 70\}\} \rangle$	$\langle \{1\}, \{2, 3, 4\} \rangle$
s_3	$\langle \{30, 50, 70\} \rangle$	$\langle \{\{30\}, \{70\}\} \rangle$	$\langle \{1, 3\} \rangle$
s_4	$\langle \{30\}, \{40, 70\}, \{80\} \rangle$	$\langle \{30\}, \{\{40\}, \{70\}, \{40, 70\}\}, \{80\} \rangle$	$\langle \{1\}, \{2, 3, 4\}, \{5\} \rangle$
s_5	$\langle \{80\} \rangle$	$\langle \{80\} \rangle$	$\langle \{5\} \rangle$



4.产生频繁序列阶段

- $\langle 1, 2, 3, 4 \rangle$
- $\langle 1, 5 \rangle$

客户号	原客户序列	新客户序列	映射后
s_1	$\langle \{30\}, \{80\} \rangle$	$\langle \{30\}, \{80\} \rangle$	$\langle \{1\}, \{5\} \rangle$
s_2	$\langle \{10, 20\}, \{30\}, \{40, 60, 70\} \rangle$	$\langle \{30\}, \{\{40\}, \{70\}, \{40, 70\}\} \rangle$	$\langle \{1\}, \{2, 3, 4\} \rangle$:
s_3	$\langle \{30, 50, 70\} \rangle$	$\langle \{\{30\}, \{70\}\} \rangle$	$\langle \{1, 3\} \rangle$
s_4	$\langle \{30\}, \{40, 70\}, \{80\} \rangle$	$\langle \{30\}, \{\{40\}, \{70\}, \{40, 70\}\}, \{80\} \rangle$	$\langle \{1\}, \{2, 3, 4\}, \{5\} \rangle$
s_5	$\langle \{80\} \rangle$	$\langle \{80\} \rangle$	$\langle \{5\} \rangle$

https://blog.csdn.net/w_z_y1997



AprioriAll算法

- 利用转换后的序列数据库寻找频繁序列，AprioriAll算法如下：
- 输入：转换后的序列数据库 S ，所有项集合 I ，
最小支持度阈值 min_sup
- 输出：序列模式集合 L
- 方法：链接&剪枝



AprioriAll算法

- AprioriAll算法源于频繁集算法Apriori，它把Apriori的基本思想扩展到序列挖掘中，也是一个多遍扫描数据库的算法。
- 在每一遍扫描中都利用前一遍的大序列来产生候选序列，然后在完成遍历整个数据库后测试它们的支持度。
- 在第一遍扫描中，利用大项目集阶段的输出来初始化大1-序列的集合。
- 在每次遍历中，从一个由大序列组成的种子集开始，利用这个种子集，可以产生新的潜在的大序列。
- 在第一次遍历前，所有在大项集阶段得到的大1-序列组成了种子集。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- **AprioriAll 算法**
- AprioriSome 算法
- GSP算法





AprioriAll算法

- AprioriAll算法源于频繁集算法Apriori，它把Apriori的基本思想扩展到序列挖掘中，也是一个多遍扫描数据库的算法。
- 在每一遍扫描中都利用前一遍的大序列来产生候选序列，然后在完成遍历整个数据库后测试它们的支持度。
- 在第一遍扫描中，利用大项目集阶段的输出来初始化大1-序列的集合。
- 在每次遍历中，从一个由大序列组成的种子集开始，利用这个种子集，可以产生新的潜在的大序列。
- 在第一次遍历前，所有在大项集阶段得到的大1-序列组成了种子集。



1. AprioriAll算法描述

算法6-1 AprioriAll算法

输入：大项集阶段转换后的序列数据库 D_T

输出：所有最长序列

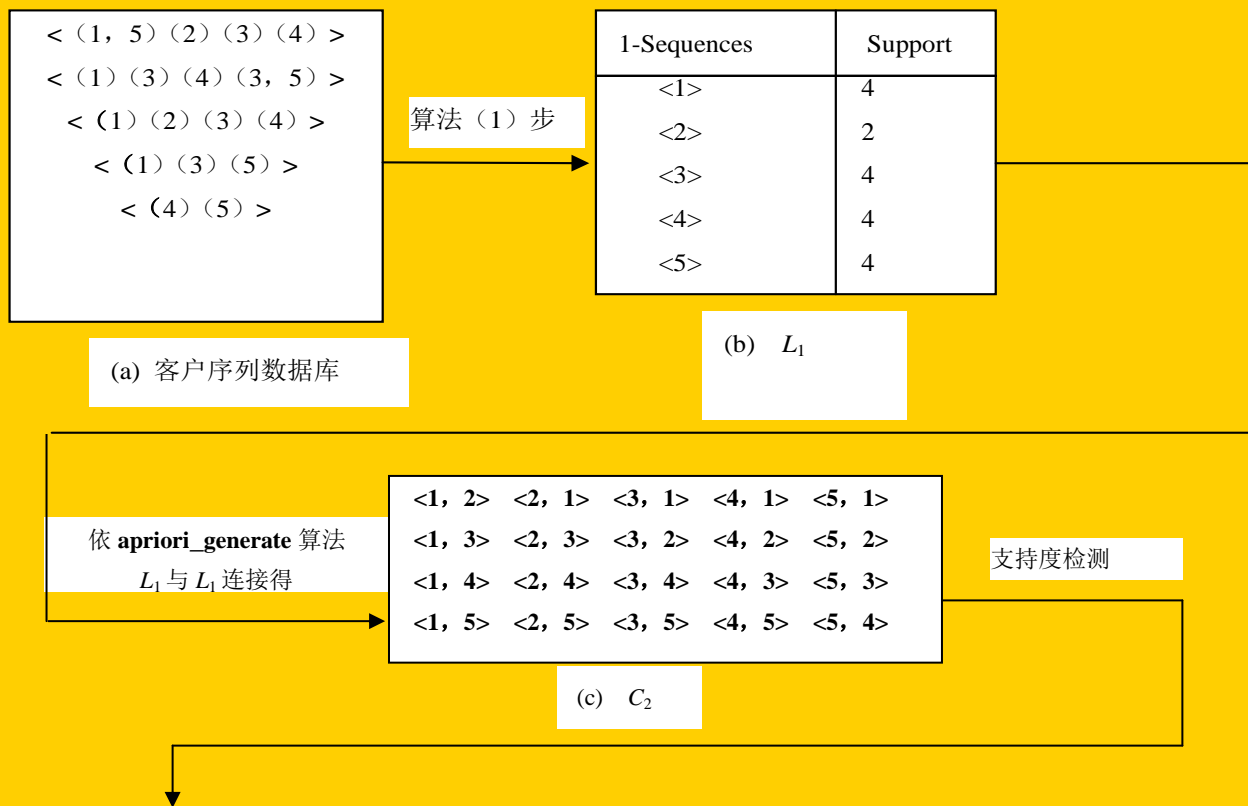
- (1) $L_1 = \{\text{large 1-sequences}\}$; // 大项集阶段得到的结果
- (2) FOR ($k=2$; $L_{k-1} \neq \emptyset$; $k++$) DO BEGIN
- (3) $C_k = \text{aprioriALL_generate}(L_{k-1})$; // C_k 是从 L_{k-1} 中产生的新的候选者
- (4) FOR each customer-sequence c in D_T DO
//对于在数据库中的每一个顾客序列 c
 - (5) Sum the count of all candidates in C_k that are contained in c ;
//被包含于 c 中 C_k 内的所有候选者计数
- (6) $L_k = \text{Candidates in } C_k \text{ with minimum support}$
// $L_k = C_k$ 中满足最小支持度的候选者
- (7) END;
- (8) $\text{Answer} = \text{Maximal Sequences in } \bigcup_k L_k$;

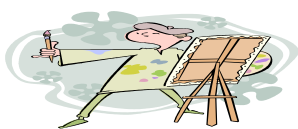
由于我们在关联规则一章对Apriori算法进行了详尽地介绍，因此上面给出的算法流程很容易理解。它的关键仍然是候选集的产生，具体候选者的产生如下：



AprioriAll算法举例

例子6-2给出一个客户序列组成的数据库如图6-11 (a) 所示, 在这里我们没有给出源数据库的形式, 即客户序列已经以转换的形式出现。假如最小支持度为40% (也就是至少两个客户序列) 那么在大项集阶段可以得到大1-序列, 之后应用AprioriAll算法可以逐步演变成最终的大序列集。图6-11给出了整个过程。





AprioriAll算法举例

2-Sequences	Support
<1, 2>	2
<1, 3>	4
<1, 4>	3
<1, 5>	3
<2, 3>	2
<2, 4>	2
<3, 4>	3
<3, 5>	2
<4, 5>	2

(d) L_2

依 apriori_generate 算法
 L_2 与 L_2 连接得

<1, 2, 3> <1, 3, 2> <1, 4, 2> <1, 5, 2>
<2, 3, 4> <2, 4, 3> <3, 4, 5> <3, 5, 4>
<1, 2, 4> <1, 3, 4> <1, 4, 3> <1, 5, 3>
<1, 2, 5> <1, 3, 5> <1, 4, 5> <1, 5, 4>

(e) C_3 (未修剪)

C_3 修剪

<1, 2, 3> <1, 3, 4>
<1, 4, 5> <2, 3, 4>
<3, 4, 5> <1, 2, 4> <1, 3, 5>

(f) C_3

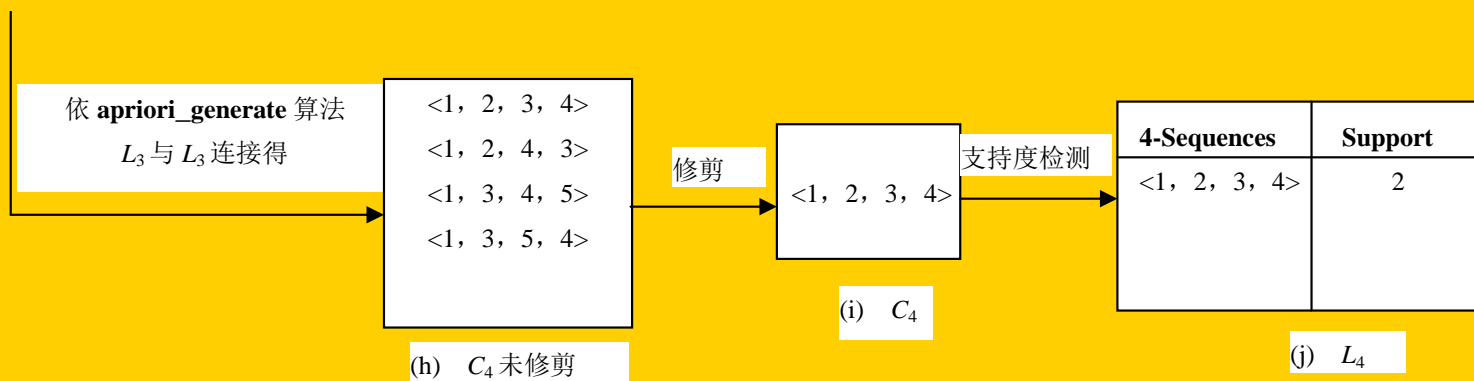
支持度检测

3-Sequences	Support
<1, 2, 3>	2
<1, 2, 4>	2
<1, 3, 4>	3
<1, 3, 5>	2
<2, 3, 4>	2

(g) L_3



AprioriAll算法举例



至此，AprioriAll算法找到了所有的大- k 序列集，即 L_1 、 L_2 、 L_3 、 L_4 ，对于大- k 序列集进行Maximal Sequences in $\cup_k L_k$ 运算，最终得到最大的大序列。上例结果如下表所示：

Sequences	Support
$\langle 1, 2, 3, 4 \rangle$	2
$\langle 1, 3, 5 \rangle$	2
$\langle 4, 5 \rangle$	2

AprioriAll利用了Apriori算法的思想，但是在候选产生和生成频繁序列方面需要考虑序列元素有序的特点进行相应地处理。表6-8只给出了最终的频繁序列，实际上在候选产生过程中有大量序列产生。例如图6-11中，在由 L_2 产生 C_3 过程中，诸如 $\langle 2, 3, 4 \rangle$ 和 $\langle 2, 4, 3 \rangle$ 都作为候选被测试，只不过因为 $\langle 2, 4, 3 \rangle$ 不满足支持度要求而在演化 L_3 过程中被裁减掉。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- **AprioriSome 算法**
- GSP算法





AprioriSome算法

AprioriSome算法可以看作是AprioriAll算法的改进，具体过程分为两个阶段：

- **前推阶段：**此阶段用于找出指定长度的所有大序列。
- **回溯阶段：**此阶段用于查找其他长度的所有大序列。

算法6-3 AprioriSome算法

输入：大项集阶段转换后的序列数据库 D_I

输出：所有最长序列

// *Forward Phase* — 前推阶段；

(1) $L_1 = \{\text{large 1-sequences}\}$; //大项目集阶段的结果；

(2) $C_1 = L_1$;

(3) $\text{last} = 1$; //最后计数的Clast

(4) FOR ($k=2$; $C_{k-1} \neq \emptyset$ and $L_{\text{last}} \neq \emptyset$; $k++$) DO BEGIN

(5) IF ($L_{k-1} \text{ know}$) THEN $C_k = \text{New candidates generated from } L_{k-1}$;

// C_k =产生于 L_{k-1} 新的候选集

(6) ELSE $C_k = \text{New candidates generated from } C_{k-1}$; // C_k =产生于 C_{k-1} 新的候选集

(7) IF ($k = \text{next}(\text{last})$) THEN BEGIN

(9) FOR each customer-sequence c in the database DO //对于在数据库中的每一个客户序列 c

(10) Sum the count of all candidates in C_k that are contained in c ;

//求包含在 c 中的 C_k 的候选者的数目之和

(11) $L_k = \text{Candidates in } C_k \text{ with minimum support}$; // L_k =在 C_k 中满足最小支持度的候选者

(12) $\text{last} = k$;

(13) END;

(14) END;



AprioriSome算法(续)

// Backward Phase — 回溯阶段:

(15) FOR (k - - ; k >= 1; k - -) DO

(16) IF (L_k not found in forward phase) THEN BEGIN // L_k 在前推阶段没有确定的情况

(17) Delete all sequences in C_k contained in Some L_i , $i > k$;

//删除所有在 C_k 中包含在 L_k 中的序列, $i > k$

(18) FOR each customer-sequence c in D_T DO //对于在 D_T 中的每一个客户序列 c

(19) Sum the count of all candidates in C_k that are contained in c ;

//对在 C_k 中包含在 c 中的所有的候选这的计数

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

(21) END;

(22) ELSE Delete all sequences in L_k contained in Some L_i , $i > k$; // L_k 已知

(23) Answer = $\cup_k L_k$; //从 k 到 m 求 L_k 的并集

在前推阶段 (forward phase) 中, 我们只对特定长度的序列进行计数。比如, 前推阶段我们对长度为1、2、4和6的序列计数 (计算支持度), 而长度为3和5的序列则在回溯阶段中计数。next函数以上次遍历的序列长度作为输入, 返回下次遍历中需要计数的序列长度。

算法6-4 next (k: integer)

IF ($hit_k < 0.666$) THEN return k + 1;

ELSEIF ($hit_k < 0.75$) THEN return k + 2;

ELSEIF ($hit_k < 0.80$) THEN return k + 3;

ELSEIF ($hit_k < 0.85$) THEN return k + 4;

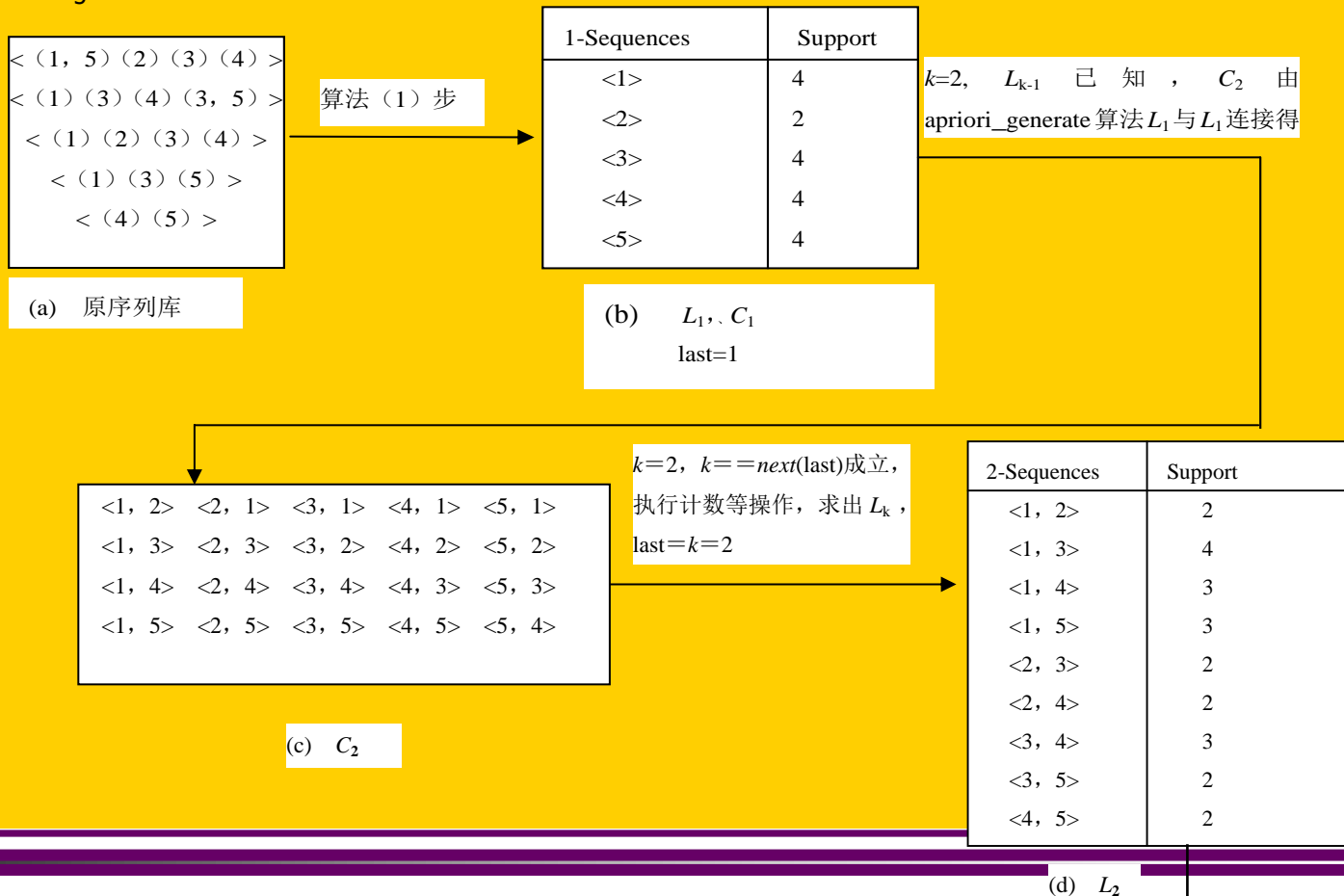
ELSE THEN return k + 5;

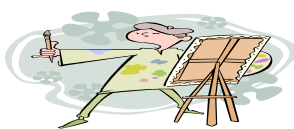
hit_k 被定义为大 k -序列 (large k -sequence) 和候选 k -序列 (candidate k -sequence) 的比率, 即 $|L_k|/|C_k|$ 。这个函数的功能是确定对哪些序列进行计数, 在对非最大序列计数时间的浪费和计算扩展小候选序列之间作出权衡。



AprioriSome算法

例子 6-3 我们仍然采用AprioriAll算法用过的图6-11 (a) 数据库例子来说明AprioriSome算法。在大项集阶段可以找出 L_1 (和图6-9 (b) 的 L_1 相同)。假设 $next(k) = 2k$, 则通过计算 C_2 可以得到 L_2 (和图6-11 (d) 中的 L_2 相同)。第三次遍历后, `apriori_generate`函数以 L_2 作为输入参数来产生 C_3 。图6-12 (e) 给出了 C_3 中的候选序列。我们不计算 C_3 , 因此也不产生 L_3 。下一步`apriori_generate`函数以 C_3 来产生 C_4 , 在经过剪枝后, 得到的结果和图6-9 (i) 所示的 C_4 相同。在以 C_4 计算 L_4 (图6-12 (i)) 之后, 我们试图产生 C_5 , 这时的结果为空。前推阶段的具体运行见下图6-12所示。





AprioriSome算法

$k=3, L_{k-1}=L_2$ 是已知的, 所以 C_k 由 L_{k-1} 即 C_3 由 L_2 产生, 同样调用 apriori_generate 算法

<1, 2, 3> <2, 3, 4>
<1, 2, 4> <3, 4, 5>
<1, 3, 4>
<1, 4, 5>
<1, 3, 5>

(e) C_3

$k=3, \text{next}(2)=4,$
 $k == \text{next}(\text{last})$ 成立

$k=4, C_{k-1}=C_3$ 不空,
 $L_{\text{last}}=L_2$ 不空, 继续执行

(f)

$k=4, L_{k-1}=L_3$ 是未知的, 所以 C_k 由 C_{k-1} 即 C_4 由 C_3 产生, 同样调用 apriori_generate 算法

<1, 2, 3, 4>
<1, 2, 4, 3>
<1, 3, 4, 5>
<1, 3, 5, 4>

(g)

修剪

<1, 2, 3, 4>

(h) C_4

$k=4, k == \text{next}(2)$ 成立, 执行计数等操作, 求出 L_k ,
 $\text{last}=k=4$

4-Sequences	Support
<1, 2, 3, 4>	2

(i) L_4

$k=5, C_{k-1}=C_4$ 不空, $L_{\text{last}}=L_4$ 不空, 继续执行, 其后 C_5 产生为空。
 $k=5, \text{next}(4)=8$, 此时不满足条件, 前半部分到此结束

(j)



AprioriSome算法

回溯阶段的具体运行见下图6-13所示。

$k=4$, L_4 在前半部分确定, 删除所有在 L_4 中包含在某些 L_i 中的序列, $i>k$

4-Sequences	Support
<1, 2, 3, 4>	2

(a) L_4

$k=3$, L_3 在前半部分没有确定, 删除 C_3 中某些 L_i 中的序列, $i>k$

<1, 3, 5>
<1, 4, 5>
<3, 4, 5>

(b) 删除大4序列的子序列之后的候选 C_3 序列

对新的 C_3 计数
求出 L_3

3-Sequences	Support
<1, 3, 5>	2

(c) L_3

$k=2$, L_2 在前半部分确定, 删除所有在 L_2 中包含在某些 L_i 中的序列, $i>k$

2-Sequences	Support
<4, 5>	2

(d) L_2

$k=1$, L_1 在前半部分确定, 删除所有在 L_1 中包含在某些 L_i 中的序列, $i>k$

C_5 为空, 至此后半部分算法执行结束

Answer = $\cup_k L_k$

Sequences	Support
<1, 2, 3, 4>	2
<1, 3, 5>	2
<4, 5>	2

(e)

(f) 最大的大序列



AprioriAll和AprioriSome比较

AprioriAll和AprioriSome比较

- AprioriAll用 L_{k-1} 去算出所有的候选 C_k ，而AprioriSome会直接用 C_{k-1} 去算出所有的候选 C_k ，因为 C_{k-1} 包含 L_{k-1} ，所以AprioriSome会产生比较多的候选。
- 虽然AprioriSome跳跃式计算候选，但因为它所产生的候选比较多，可能在回溯阶段前就占满内存。
- 如果内存满了，AprioriSome就会被强迫去计算最后一组的候选，（即使原本是要跳过此项）。这样，会影响并减少已算好的两个候选间的跳跃距离，而使得AprioriSome会变的跟AprioriAll一样。
- 对于较低的支持度，有较长的大序列，也因此有较多的非最大序列，所以AprioriSome比较好。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- **GSP算法**





GSP算法主要包括三个步骤:

- ①扫描序列数据库, 得到长度为1的序列模式 L_1 , 作为初始的种子集;
- ②根据长度为 i 的种子集 L_i 通过连接操作和剪切操作生成长度为 $i+1$ 的候选序列模式 C_{i+1} ; 然后扫描序列数据库, 计算每个候选序列模式的支持数, 产生长度为 $i+1$ 的序列模式 L_{i+1} , 并将 L_{i+1} 作为新的种子集;
- ③重复第二步, 直到没有新的序列模式或新的候选序列模式产生为止。

其中, 产生候选序列模式主要分两步:

- **连接阶段:** 如果去掉序列模式 S_1 的第一个项目与去掉序列模式 S_2 的最后一个项目所得到的序列相同, 则可以将 S_1 与 S_2 进行连接, 即将 S_2 的最后一个项目添加到 S_1 中。
- **剪切阶段:** 若某候选序列模式的某个子序列不是序列模式, 则此候选序列模式不可能是序列模式, 将它从候选序列模式中删除。

候选序列模式的支持度计算按照如下方法进行:

- 对于给定的候选序列模式集合 C , 扫描序列数据库 D , 对于其中的每一条序列 d , 找出集合 C 中被 d 所包含的所有候选序列模式, 并增加其支持度计数。



算法6-5 GSP算法

输入：大项集阶段转换后的序列数据库 D_T 。

输出：最大序列

- (1) $L_1 = \{\text{large 1-sequences}\}$; // 大项集阶段得到的结果
- (2) FOR ($k = 2$; $L_{k-1} \neq \emptyset$; $k++$) DO BEGIN
- (3) $C_k = \text{GSPgenerate}(L_{k-1})$;
- (4) FOR each customer-sequence \mathbf{c} in the database D_T DO
- (5) Increment the count of all candidates in C_k that are contained in \mathbf{c} ;
- (6) $L_k = \text{Candidates in } C_k \text{ with minimum support}$;
- (7) END;
- (8) Answer = Maximal Sequences in $\bigcup_k L_k$;



GSP算法部分例子

例子 6-5 表6-9演示了从长度为3的序列模式产生长度为4的候选序列模式的过程。
演示了从长度为3的序列模式产生长度为4的候选序列模式的过程。

在连接阶段：

- 序列 $\langle (1, 2), 3 \rangle$ 可以与 $\langle 2, (3, 4) \rangle$ 连接，因为 $\langle (*, 2), 3 \rangle$ 与 $\langle 2, (3, *) \rangle$ 是相同的，两序列连接后为 $\langle (1, 2), (3, 4) \rangle$
- $\langle (1, 2), 3 \rangle$ 与 $\langle 2, 3, 5 \rangle$ 连接，得到 $\langle (1, 2), 3, 5 \rangle$ 。

剩下的序列是不能连接的，比如 $\langle (1, 2), 4 \rangle$ 不能与任何长度为3的序列连接，这是因为其他序列没有 $\langle (2), (4, *) \rangle$ 或者 $\langle (2), (4), (*) \rangle$ 的形式。

表6-9 GSP算法举例

Sequential patterns With Length 3	Candidate4-Sequences	
	After Join	After Pruning
$\langle (1, 2), 3 \rangle$ $\langle (1, 2), 4 \rangle$ $\langle 1, (3, 4) \rangle$ $\langle (1, 3), 5 \rangle$ $\langle 2, (3, 4) \rangle$ $\langle 2, 3, 5 \rangle$	$\langle (1, 2), (3, 4) \rangle$ $\langle (1, 2), 3, 5 \rangle$	$\langle (1, 2), (3, 4) \rangle$

在修剪阶段 $\langle (1, 2), 3, 5 \rangle$ 将被剪掉，这是因为 $\langle 1, 3, 5 \rangle$ 并不在 L_3 中，而 $\langle (1, 2), (3, 4) \rangle$ 的长度为3的子序列都在 L_3 因而被保留下来。

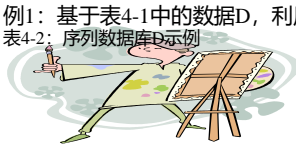


GSP算法完整举例

例：基于表中的数据D，利用GSP算法进行序列模式挖掘，最小支持数为2（最小支持度为50%）。

注：为了节省篇幅，在不引起误解的情况下表中省略了逗号和括号。

序列数据库D	
用户	访问序列
10	<a (abc) (ab) d (cf) >
20	<(ad) c (bc) (ae) >
30	<(ef) (ab) (df) cb>
40	<eg (af) cbc>



GSP算法完整举例

1) 对原始数据库D进行扫描，查找长度为1的频繁序列，构造种子集S1

扫描发现频繁序列及其支持度 $\langle a \rangle:4$; $\langle b \rangle:4$; $\langle c \rangle:4$; $\langle d \rangle:3$; $\langle e \rangle:3$; $\langle f \rangle:3$ ，从而得到种子集 $S1 = \{\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle, \langle f \rangle, \langle g \rangle, \langle h \rangle\}$;

由种子集S1构造长度为2的候选集C2

可以直接连接S1无需判断，也无需进行剪枝。

2) S2生成过程。

$\langle aa \rangle, \langle aa \rangle, \langle ab \rangle, \langle ab \rangle, \langle ba \rangle, \langle ac \rangle, \langle ac \rangle, \langle ca \rangle, \langle ad \rangle, \langle ad \rangle, \langle da \rangle, \langle ae \rangle, \langle ae \rangle, \langle ea \rangle, \langle af \rangle, \langle af \rangle, \langle fa \rangle, \langle bb \rangle, \langle bb \rangle, \langle bc \rangle, \langle bc \rangle, \langle cb \rangle, \langle bd \rangle, \langle bd \rangle, \langle db \rangle, \langle be \rangle, \langle be \rangle, \langle eb \rangle, \langle bf \rangle, \langle bf \rangle, \langle fb \rangle, \langle cc \rangle, \langle cc \rangle, \langle cd \rangle, \langle cd \rangle, \langle dc \rangle, \langle ce \rangle, \langle ce \rangle, \langle ec \rangle, \langle cf \rangle, \langle cf \rangle, \langle fc \rangle, \langle dd \rangle, \langle dd \rangle, \langle de \rangle, \langle de \rangle, \langle ed \rangle, \langle df \rangle, \langle df \rangle, \langle fd \rangle, \langle ee \rangle, \langle ee \rangle, \langle ef \rangle, \langle ef \rangle, \langle fe \rangle, \langle ff \rangle, \langle ff \rangle$

因为1项序列都频繁，所以无需剪枝，可以直接保留连接后的序列。

2的种子集:

$S2 = \{\langle aa \rangle, \langle a, b \rangle, \langle ab \rangle, \langle ba \rangle, \langle ac \rangle, \langle ca \rangle, \langle ad \rangle, \langle ea \rangle, \langle af \rangle, \langle bb \rangle, \langle b, c \rangle, \langle bc \rangle, \langle cb \rangle, \langle bd \rangle, \langle db \rangle, \langle bf \rangle, \langle cc \rangle, \langle dc \rangle, \langle ec \rangle, \langle cf \rangle, \langle fc \rangle, \langle ef \rangle, \langle ff \rangle\}$ 。



GSP算法完整举例

3) 由种子集S2构造长度为3的候选集C3

由于S2的序列太多，仅以S2中的三个序列 $\langle aa \rangle$, $\langle (a,b) \rangle$, $\langle ab \rangle$ 来演示连接和剪枝的原理：

3-1) 连接阶段

$\langle aa \rangle$ 序列去掉第一个元素a后得到序列 $\langle a \rangle$ ， $\langle (a,b) \rangle$ 序列去掉最后一个元素b后也得到序列 $\langle a \rangle$ ，所以 $\langle aa \rangle$ 与 $\langle (a,b) \rangle$ 这两个序列是可以连接的，将 $\langle (a,b) \rangle$ 的最后一个元素b连接到 $\langle aa \rangle$ 序列的最后，由于元素b在原序列属于最后一个集合中，因此要将b与a合并形成集合 $\langle a,b \rangle$ ，所以连接后得到的序列为 $\langle a(a,b) \rangle$ 。

同理，发现 $\langle aa \rangle$ 与 $\langle ab \rangle$ 也可以连接，得到序列 $\langle aab \rangle$ 。

3-2) 剪枝阶段

$\langle a(a,b) \rangle$ 其所有相邻子序列 $\langle ab \rangle$ ， $\langle aa \rangle$ ， $\langle (a,b) \rangle$ 都包含在S2中，不用剪掉； $\langle aab \rangle$ 同理

剪枝后：S3=

$\{ \langle aaa \rangle, \langle a(a,b) \rangle, \langle aab \rangle, \langle baa \rangle, \langle aac \rangle, \langle caa \rangle, \langle aad \rangle, \langle eaa \rangle, \langle aaf \rangle, \langle (a,b)a \rangle, \langle b(a,b) \rangle, \langle bab \rangle, \langle c(a,b) \rangle, \langle cab \rangle, \langle (a,b)b \rangle, \langle (a,b)c \rangle, \langle (a,b)d \rangle, \langle (a,b)f \rangle, \langle aba \rangle, \langle abb \rangle, \langle a(b,c) \rangle, \langle abc \rangle, \langle abd \rangle, \langle abf \rangle, \langle bac \rangle, \langle bad \rangle, \langle baf \rangle, \langle bba \rangle, \langle cba \rangle, \langle aca \rangle, \langle cac \rangle, \langle eac \rangle, \langle acb \rangle, \langle acc \rangle, \langle acf \rangle, \langle caf \rangle, \langle (b,c)a \rangle, \langle bca \rangle, \langle cca \rangle, \langle eca \rangle, \langle adb \rangle, \langle adc \rangle, \langle eaf \rangle, \langle afc \rangle, \langle aff \rangle, \langle bbb \rangle, \langle b(b,c) \rangle, \langle bbc \rangle, \langle cbb \rangle, \langle bbd \rangle, \langle dbb \rangle, \langle bbf \rangle, \langle (b,c)b \rangle, \langle c(b,c) \rangle, \langle cbc \rangle, \langle d(b,c) \rangle, \langle dbc \rangle, \langle (b,c)c \rangle, \langle (b,c)f \rangle, \langle bcb \rangle, \langle bcc \rangle, \langle bcf \rangle, \langle cbf \rangle, \langle ccb \rangle, \langle dcb \rangle, \langle bdb \rangle, \langle bdc \rangle, \langle bfc \rangle, \langle bff \rangle, \langle ccc \rangle, \langle dcc \rangle, \langle ecc \rangle, \langle ccf \rangle, \langle fcc \rangle, \langle ecf \rangle, \langle cfc \rangle, \langle fcf \rangle, \langle cff \rangle, \langle efc \rangle, \langle ffc \rangle, \langle eff \rangle, \langle fff \rangle \}$



GSP算法完整举例

3) 由种子集S3构造长度为3的候选集S4

$C4 = \{ \langle (a,b)(c,d) \rangle, \langle (a,b)dc \rangle, \langle a(a,b,c) \rangle, \langle a(b,c)a \rangle, \langle ea(a,c) \rangle, \langle eaca \rangle, \langle ea(b,c) \rangle, \langle each \rangle, \langle ea(c,c) \rangle, \langle eacc \rangle, \langle ea(c,f) \rangle, \langle eacf \rangle, \langle ad(b,c) \rangle, \langle adcb \rangle, \langle ea fc \rangle, \langle bd(b,c) \rangle, \langle bdc b \rangle \}$

$S4 = \{ \langle (a,b)dc \rangle, \langle a(b,c)a \rangle, \langle ea fc \rangle \}$ 。

4) 算法运行结束。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法





Thank you !!!

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- **AprioriSome 算法**
- GSP算法





AprioriSome算法

AprioriSome算法可以看作是AprioriAll算法的改进，具体过程分为两个阶段：

- **前推阶段：**找出指定长度的所有大序列。
- **回溯阶段：**查找其他长度的所有大序列。

算法6-3 AprioriSome算法

输入：大项集阶段转换后的序列数据库 D_T

输出：所有最长序列

// *Forward Phase* — 前推阶段；

(1) $L_1 = \{\text{large 1-sequences}\}$; //大项目集阶段的结果；

(2) $C_1 = L_1$;

(3) $\text{last} = 1$; //最后计数的Clast

(4) FOR ($k = 2$; $C_{k-1} \neq \emptyset$ and $L_{\text{last}} \neq \emptyset$; $k++$) DO BEGIN

(5) IF ($L_{k-1} \text{ known}$) THEN $C_k = \text{New candidates generated from } L_{k-1}$;

// C_k =产生于 L_{k-1} 新的候选集

(6) ELSE $C_k = \text{New candidates generated from } C_{k-1}$; // C_k =产生于 C_{k-1} 新的候选集

(7) IF ($k = \text{next}(\text{last})$) THEN BEGIN

(9) FOR each customer-sequence c in the database DO //对于在数据库中的每一个客户序列 c

(10) Sum the count of all candidates in C_k that are contained in c ;

//求包含在 c 中的 C_k 的候选者的数目之和

(11) $L_k = \text{Candidates in } C_k \text{ with minimum support}$; // L_k =在 C_k 中满足最小支持度的候选者

(12) $\text{last} = k$;

(13) END;

(14) END;



AprioriSome算法(续)

// Backward Phase — 回溯阶段;

(15) FOR (k -- ; k >= 1; k --) DO

(16) IF (L_k not found in forward phase) THEN BEGIN // L_k
在前推阶段没有确定的情况

(17) Delete all sequences in C_k contained in Some L_i , $i > k$;
//删除所有在 C_k 中包含在 L_k 中的序列, $i > k$

(18) FOR each customer-sequence c in D_T DO //对于在
DT中的每一个客户序列 c

(19) Sum the count of all candidates in C_k that are contained
in c ;

//对在 C_k 中包含在 c 中的所有的候选这的计数

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

(21) END;

(22) ELSE Delete all sequences in L_k contained in Some L_i , i
> k ; // L_k 已知

(23) Answer = $\bigcup_k L_k$; //从 k 到 m 求 L_k 的并集



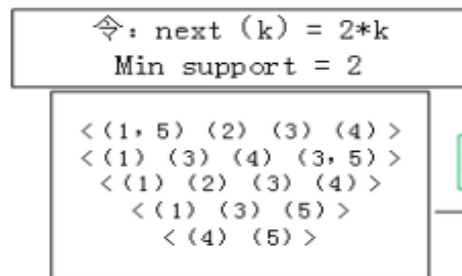
AprioriSome算法(续)

在前推阶段（forward phase）中，我们只对特定长度的序列进行计数。比如，前推阶段我们对长度为1、2、4和6的序列计数（计算支持度），而长度为3和5的序列则在回溯阶段中计数。**next**函数以上次遍历的序列长度作为输入，返回下次遍历中需要计数的序列长度。

算法**6-4** next (k: integer)

```
IF ( $\text{hit}_k < 0.666$ ) THEN return  $k + 1$ ;  
ELSEIF ( $\text{hit}_k < 0.75$ ) THEN return  $k + 2$ ;  
ELSEIF ( $\text{hit}_k < 0.80$ ) THEN return  $k + 3$ ;  
ELSEIF ( $\text{hit}_k < 0.85$ ) THEN return  $k + 4$ ;  
ELSE THEN return  $k + 5$ ;
```

hit_k 被定义为大k-序列（large k-sequence）和候选k-序列（candidate k-sequence）的比率，即 $|L_k|/|C_k|$ 。这个函数的功能是确定对哪些序列进行计数，在对非最大序列计数时间的浪费和计算扩展小候选序列之间作出权衡。



(a) 原序列数据

1-Sequences	Support
$\langle 1 \rangle$	4
$\langle 2 \rangle$	2
$\langle 3 \rangle$	4
$\langle 4 \rangle$	4
$\langle 5 \rangle$	4

(b) L1 C1
Last = 1

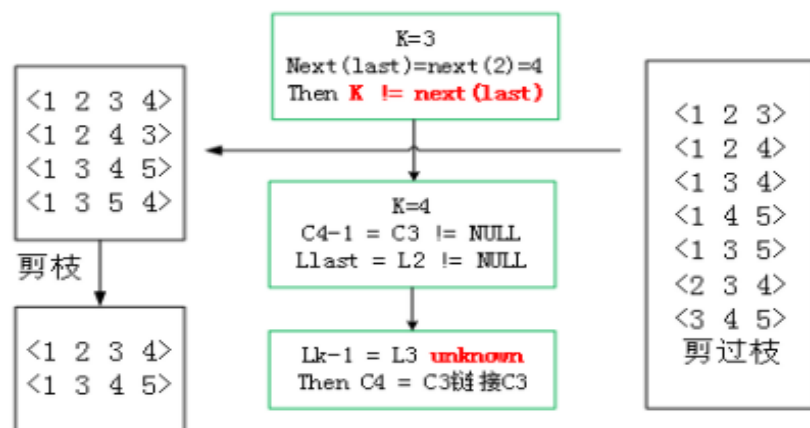
$K=2, C_{2-1} \neq \text{NULL}$
 $L_{\text{last}} = L1 \neq \text{NULL}$

$L_{k-1} = L1$ **known**
Then $C2 = L1$ 链接 $L1$

$\langle 1, 2 \rangle \langle 2, 1 \rangle \langle 3, 1 \rangle \langle 4, 1 \rangle \langle 5, 1 \rangle$
 $\langle 1, 3 \rangle \langle 2, 3 \rangle \langle 3, 2 \rangle \langle 4, 2 \rangle \langle 5, 2 \rangle$
 $\langle 1, 4 \rangle \langle 2, 4 \rangle \langle 3, 4 \rangle \langle 4, 3 \rangle \langle 5, 3 \rangle$
 $\langle 1, 5 \rangle \langle 2, 5 \rangle \langle 3, 5 \rangle \langle 4, 5 \rangle \langle 5, 4 \rangle$

(c) C2

$K=2$
 $\text{next}(\text{last}) = \text{next}(1) = 2$
Then $K = \text{next}(\text{last})$



(e) C3

$\text{Last} = k = 2$
 $K=3$
 $C_{3-1} \neq \text{NULL}$
 $L2 \neq \text{NULL}$

$L_{3-1} = L2$ **known**
Then $C3 = L2$ 链接 $L2$

2-Sequences	Support
$\langle 1 2 \rangle$	2
$\langle 1 3 \rangle$	4
$\langle 1 4 \rangle$	3
$\langle 1 5 \rangle$	2
$\langle 2 3 \rangle$	2
$\langle 2 4 \rangle$	2
$\langle 3 4 \rangle$	3
$\langle 3 5 \rangle$	2
$\langle 4 5 \rangle$	2

(d) L2

(f) C4

$K=4$
 $\text{Next}(\text{last}) = \text{next}(2) = 4$
Then $K = \text{next}(\text{last})$

$\text{Last} = k = 4$
 $K = 5$
 $C4 \neq \text{NULL} L4 \neq \text{NULL}$

$L_{5-1} = L4$ **known**
Then $C5 = L4$ 链接 $L4$

4-Sequences	Support
$\langle 1 2 3 4 \rangle$	2
$\langle 1 3 4 5 \rangle$	1

(g) L4

$C5 = \text{NULL}$

$K=5$
 $\text{Next}(\text{last}) = \text{next}(4) = 8$
 $C5 = \text{NULL}$

Forward
phase
End

Forward
phase

3-Sequences	Support
<1 4 5>	4
<1 3 5>	2
<3 4 5>	4

(j) L3

$K=K-- = 2$
L2 known
 删除<1 2 3 4>和
 <1 3 5>的子序列

2-Sequences	Support
<1 2>	2
<1 3>	4
<1 4>	3
<1 5>	2
<2 3>	2
<2 4>	2
<3 4>	3
<3 5>	2
<4 5>	2

(k) L2

$K=K-- = 1$
L1 known

L1 = NULL

Sequences	Support
<1 2 3 4>	2
<1 3 5>	2
<4 5>	2

(l) sequence pattern

<1 2 3>
<1 2 4>
<1 3 4>
<1 4 5>
<1 3 5>
<2 3 4>
<3 4 5>

(i) C3

$K=K-- = 3$
L3 unknown

4-Sequences	Support
<1 2 3 4>	2

(h) L4

$K=K-- = 4$
L4 known

**Backward
phase**



AprioriAll和AprioriSome比较

AprioriAll和AprioriSome比较

- AprioriAll用 L_{k-1} 去算出所有的候选 C_k ，而AprioriSome会直接用 C_{k-1} 去算出所有的候选 C_k ，因为 C_{k-1} 包含 L_{k-1} ，所以AprioriSome会产生比较多的候选。
- 虽然AprioriSome跳跃式计算候选，但因为它所产生的候选比较多，可能在回溯阶段前就占满内存。
- 如果内存满了，AprioriSome就会被强迫去计算最后一组的候选，（即使原本是要跳过此项）。这样，会影响并减少已算好的两个候选间的跳跃距离，而使得AprioriSome会变的跟AprioriAll一样。
- 对于较低的支持度，有较长的大序列，也因此有较多的非最大序列，所以AprioriSome比较好。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- **GSP算法**





GSP算法主要包括三个步骤:

- ①扫描序列数据库, 得到长度为1的序列模式 L_1 , 作为初始的种子集;
- ②根据长度为 i 的种子集 L_i 通过连接操作和剪切操作生成长度为 $i+1$ 的候选序列模式 C_{i+1} ; 然后扫描序列数据库, 计算每个候选序列模式的支持数, 产生长度为 $i+1$ 的序列模式 L_{i+1} , 并将 L_{i+1} 作为新的种子集;
- ③重复第二步, 直到没有新的序列模式或新的候选序列模式产生为止。

其中, 产生候选序列模式主要分两步:

- **连接阶段:** 如果去掉序列模式 S_1 的第一个项目与去掉序列模式 S_2 的最后一个项目所得到的序列相同, 则可以将 S_1 与 S_2 进行连接, 即将 S_2 的最后一个项目添加到 S_1 中。
- **剪切阶段:** 若某候选序列模式的某个子序列不是序列模式, 则此候选序列模式不可能是序列模式, 将它从候选序列模式中删除。

候选序列模式的支持度计算按照如下方法进行:

- 对于给定的候选序列模式集合 C , 扫描序列数据库 D_T , 对于其中的每一条序列 d , 找出集合 C 中被 d 所包含的所有候选序列模式, 并增加其支持度计数。



GSP算法部分例子

例子 **6-5** 表6-9演示了从长度为3的序列模式产生长度为4的候选序列模式的过程:

- 序列 $\langle (1, 2), 3 \rangle$ 可以与 $\langle 2, (3, 4) \rangle$ 连接, 因为 $\langle (*, 2), 3 \rangle$ 与 $\langle 2, (3, *) \rangle$ 是相同的, 两序列连接后为 $\langle (1, 2), (3, 4) \rangle$
- $\langle (1, 2), 3 \rangle$ 与 $\langle 2, 3, 5 \rangle$ 连接, 得到 $\langle (1, 2), 3, 5 \rangle$ 。

剩下的序列是不能连接的, 比如 $\langle (1, 2), 4 \rangle$ 不能与任何长度为3的序列连接, 这是因为其他序列没有 $\langle (2), (4, *) \rangle$ 或者 $\langle (2), (4), (*) \rangle$ 的形式。

表6-9 GSP算法举例

Sequential patterns With Length 3	Candidate4-Sequences	
	After Join	After Pruning
$\langle (1, 2), 3 \rangle$ $\langle (1, 2), 4 \rangle$ $\langle 1, (3, 4) \rangle$ $\langle (1, 3), 5 \rangle$ $\langle 2, (3, 4) \rangle$ $\langle 2, 3, 5 \rangle$	$\langle (1, 2), (3, 4) \rangle$ $\langle (1, 2), 3, 5 \rangle$	$\langle (1, 2), (3, 4) \rangle$

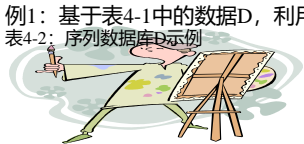
在修剪阶段 $\langle (1, 2), 3, 5 \rangle$ 将被剪掉, 这是因为 $\langle 1, 3, 5 \rangle$ 并不在 L_3 中, 而 $\langle (1, 2), (3, 4) \rangle$ 的长度为3的子序列都在 L_3 因而被保留下来。



GSP算法完整举例

例：基于表中的数据D，利用GSP算法进行序列模式挖掘，最小支持数为2
(在不引起误解的情况下表中省略了逗号和括号)

序列数据库D	
用户	访问序列
10	$\langle a(abc)(ab)d(cf) \rangle$
20	$\langle (ad)c(bc)(ae) \rangle$
30	$\langle (ef)(ab)(df)cb \rangle$
40	$\langle eg(af)cbc \rangle$



GSP算法完整举例

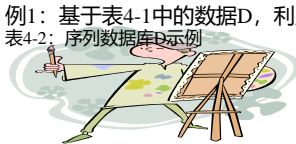
1) 对原始数据库D进行扫描，查找长度为1的频繁序列，构造种子集S1
扫描发现频繁序列及其支持度 $\langle a \rangle:4$; $\langle b \rangle:4$; $\langle c \rangle:4$; $\langle d \rangle:3$; $\langle e \rangle:3$
 $\langle f \rangle:3$ ，从而得到种子集 $S1 = \{\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle, \langle f \rangle, \langle g \rangle, \langle h \rangle\}$;
由种子集S1构造长度为2的候选集C2
可以直接连接S1无需判断，也无需进行剪枝。

2) S2生成过程。

$\langle aa \rangle, \langle aa \rangle, \langle ab \rangle, \langle ab \rangle, \langle ba \rangle, \langle ac \rangle, \langle ac \rangle, \langle ca \rangle, \langle ad \rangle, \langle ad \rangle,$
 $\langle da \rangle, \langle ae \rangle, \langle ae \rangle, \langle ea \rangle, \langle af \rangle, \langle af \rangle, \langle fa \rangle, \langle bb \rangle, \langle bb \rangle, \langle bc \rangle, \langle$
 $bc \rangle, \langle cb \rangle, \langle bd \rangle, \langle bd \rangle, \langle db \rangle, \langle be \rangle, \langle be \rangle, \langle eb \rangle, \langle bf \rangle, \langle bf \rangle, \langle fb$
 $\rangle, \langle cc \rangle, \langle cc \rangle, \langle cd \rangle, \langle cd \rangle, \langle dc \rangle, \langle ce \rangle, \langle ce \rangle, \langle ec \rangle, \langle cf \rangle, \langle cf \rangle,$
 $\langle fc \rangle, \langle dd \rangle, \langle dd \rangle, \langle de \rangle, \langle de \rangle, \langle ed \rangle, \langle df \rangle, \langle df \rangle, \langle fd \rangle, \langle ee \rangle, \langle$
 $ee \rangle, \langle ef \rangle, \langle ef \rangle, \langle fe \rangle, \langle ff \rangle, \langle ff \rangle$

因为1项序列都频繁，所以无需剪枝，可以直接保留连接后的序列。

S2的种子集: $\{\langle aa \rangle, \langle a,b \rangle, \langle ab \rangle, \langle ba \rangle, \langle ac \rangle, \langle ca \rangle, \langle ad \rangle,$
 $\langle ea \rangle, \langle af \rangle, \langle bb \rangle, \langle b,c \rangle, \langle bc \rangle, \langle cb \rangle, \langle bd \rangle, \langle db \rangle, \langle bf \rangle,$
 $\langle cc \rangle, \langle dc \rangle, \langle ec \rangle, \langle cf \rangle, \langle fc \rangle, \langle ef \rangle, \langle ff \rangle\}$ 。



GSP算法完整举例

3) 由种子集S2构造长度为3的候选集C3,由于S2的序列太多，仅以S2中的三个序列 $\langle aa \rangle$, $\langle (ab) \rangle$, $\langle ab \rangle$ 来演示连接和剪枝的原理：

连接阶段： $\langle aa \rangle$ 序列去掉第一个元素a后得到序列 $\langle a \rangle$ ， $\langle (a,b) \rangle$ 序列去掉最后一个元素b后也得到序列 $\langle a \rangle$ ，所以 $\langle aa \rangle$ 与 $\langle (a,b) \rangle$ 这两个序列是可以连接的，将 $\langle (a,b) \rangle$ 的最后一个元素b连接到 $\langle aa \rangle$ 序列的最后，由于元素b在原序列属于最后一个集合中，因此要将b与a合并形成集合 (a,b) ，所以连接后得到的序列为 $\langle a(a,b) \rangle$ 。同理，发现 $\langle aa \rangle$ 与 $\langle ab \rangle$ 也可以连接，得到序列 $\langle aab \rangle$ 。

剪枝阶段： $\langle a(a,b) \rangle$ 其所有相邻子序列 $\langle ab \rangle$ ， $\langle aa \rangle$ ， $\langle (a,b) \rangle$ 都包含在S2中，不用剪掉； $\langle aab \rangle$ 同理剪枝后

$S3 = \{ \langle aaa \rangle, \langle a(a,b) \rangle, \langle aab \rangle, \langle baa \rangle, \langle aac \rangle, \langle caa \rangle, \langle aad \rangle, \langle eaa \rangle, \langle aaf \rangle, \langle (a,b)a \rangle, \langle b(a,b) \rangle, \langle bab \rangle, \langle c(a,b) \rangle, \langle cab \rangle, \langle (a,b)b \rangle, \langle (a,b)c \rangle, \langle (a,b)d \rangle, \langle (a,b)f \rangle, \langle aba \rangle, \langle abb \rangle, \langle a(b,c) \rangle, \langle abc \rangle, \langle abd \rangle, \langle abf \rangle, \langle bac \rangle, \langle bad \rangle, \langle baf \rangle, \langle bba \rangle, \langle cba \rangle, \langle aca \rangle, \langle cac \rangle, \langle eac \rangle, \langle acb \rangle, \langle acc \rangle, \langle acf \rangle, \langle caf \rangle, \langle (b,c)a \rangle, \langle bca \rangle, \langle cca \rangle, \langle eca \rangle, \langle adb \rangle, \langle adc \rangle, \langle eaf \rangle, \langle afc \rangle, \langle aff \rangle, \langle bbb \rangle, \langle b(b,c) \rangle, \langle bbc \rangle, \langle cbb \rangle, \langle bbd \rangle, \langle dbb \rangle, \langle bbf \rangle, \langle (b,c)b \rangle, \langle c(b,c) \rangle, \langle cbc \rangle, \langle d(b,c) \rangle, \langle dbc \rangle, \langle (b,c)c \rangle, \langle (b,c)f \rangle, \langle bcb \rangle, \langle bcc \rangle, \langle bcf \rangle, \langle cbf \rangle, \langle ccb \rangle, \langle dcb \rangle, \langle bdb \rangle, \langle bdc \rangle, \langle bfc \rangle, \langle bff \rangle, \langle ccc \rangle, \langle dcc \rangle, \langle ecc \rangle, \langle ccf \rangle, \langle fcc \rangle, \langle ecf \rangle, \langle cfc \rangle, \langle fcf \rangle, \langle cff \rangle, \langle efc \rangle, \langle ffc \rangle, \langle eff \rangle, \langle fff \rangle \}$



GSP算法完整举例

3) 由种子集S3构造长度为3的候选集S4

$C4 = \{ \langle (a,b)(c,d) \rangle, \langle (a,b)dc \rangle, \langle a(a,b,c) \rangle, \langle a(b,c)a \rangle, \langle ea(a,c) \rangle, \langle eaca \rangle, \langle ea(b,c) \rangle, \langle eachb \rangle, \langle ea(c,c) \rangle, \langle eacc \rangle, \langle ea(c,f) \rangle, \langle eacf \rangle, \langle ad(b,c) \rangle, \langle adcb \rangle, \langle eafc \rangle, \langle bd(b,c) \rangle, \langle bdc b \rangle \}$

$S4 = \{ \langle (a,b)dc \rangle, \langle a(b,c)a \rangle, \langle eafc \rangle \}$ 。

4) 算法运行结束。



算法6-5 GSP算法

输入：大项集阶段转换后的序列数据库 D_T 。

输出：最大序列

- (1) $L_1 = \{\text{large 1-sequences}\}$; // 大项集阶段得到的结果
- (2) FOR ($k = 2$; $L_{k-1} \neq \emptyset$; $k++$) DO BEGIN
- (3) $C_k = \text{GSPgenerate}(L_{k-1})$;
- (4) FOR each customer-sequence c in the database D_T DO
- (5) Increment the count of all candidates in C_k that are contained in c ;
- (6) $L_k = \text{Candidates in } C_k \text{ with minimum support}$;
- (7) END;
- (8) Answer = Maximal Sequences in $\bigcup_k L_k$;



序列模式挖掘的一般步骤

序列模式挖掘主要有五个步骤：**排序阶段**、**大项集阶段**、**转换阶段**、**序列阶段**以及**选最大阶段**。

1. 排序阶段

■对数据库进行排序 (Sort)，排序的结果将原始的数据库转换成序列数据库（比较实际可能需要其他的预处理手段来辅助进行）。例如，上面介绍的交易数据库，如果以客户号 (Cust_id) 和交易时间 (trans-time) 进行排序，那么在通过对同一客户的事务进行合并就可以得到对应的序列数据库。

客户号	交易时间	物品
1	June 25'99	30
1	June 30'99	90
2	June 10'99	10, 20
2	June 15'99	30
2	June 20'99	40, 60, 70
3	June 25'99	30, 50, 70
4	June 25'99	30
4	June 30'99	40, 70
4	July 25'99	90
5	June 12'99	90

客户号	顾客序列
1	< (30) (90) >
2	< (10, 20) (30) (40, 60, 70) >
3	< (30, 50, 70) >
4	< (30) (40, 70) ((90)) >
5	< (90) >



序列模式挖掘的一般步骤

2. 大项集阶段

- 这个阶段要找出所有频繁项集（即大项集）组成的集合 L 。实际上，也同步得到所有大1-序列组成的集合，即 $\{ \langle I \rangle \mid I \in L \}$ 。
- 在上面表6-2给出的顾客序列数据库中，假设支持数为2，则大项集分别是 (30)，(40)，(70)，(40, 70) 和 (90)。

客户号	顾客序列
1	$\langle (30) (90) \rangle$
2	$\langle (10, 20) (30) (40, 60, 70) \rangle$
3	$\langle (30, 50, 70) \rangle$
4	$\langle (30) (40, 70) (90) \rangle$
5	$\langle (90) \rangle$

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40, 70)	4
(90)	5



序列模式挖掘的一般步骤(续)

序列模式挖掘主要有五个步骤：排序阶段、大项集阶段、转换阶段、序列阶段以及选最大阶段。

3. 转换阶段

- 在寻找序列模式的过程中，我们要不断地进行检测一个给定的大序列集合是否包含于一个客户序列中。

表6-7给出了表6-2数据库经过转换后的数据库。比如，在对ID号为2的客户序列进行转换的时候，交易(10, 20)被剔除了，因为它并没有包含任何大项集；交易(40, 60, 70)则被大项集的集合{(40), (70), (40, 70)}代替。

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40, 70)	4
(90)	5

4. 序列阶段

- 利用转换后的数据库寻找频繁的序列，即大序列 (Large Sequence)。

5. 选最大阶段

- 在大序列集中找出最长序列 (Maximal Sequences)。

第六章 时间序列和序列模式挖掘

内容提要

- 时间序列及其应用
- 时间序列预测的常用方法
- 基于ARMA模型的序列匹配方法
- 基于离散傅立叶变换的时间序列相似性查找
- 基于规范变换的查找方法
- 序列挖掘及其基本方法
- AprioriAll 算法
- AprioriSome 算法
- GSP算法





Thank you !!!