文章编号:1001 - 9081(2005)03 - 0649 - 03

状态演化模式挖掘在交通流预测中的应用

颜 镝,宋 苏
(北京工业大学 电子信息与控制工程学院,北京 100022)
(everman@ emails. bjut. edu. cn)

摘 要:在交通流诱导中,交通流量的预测是研究热点。为了提取交通流变化的特征规律,针对交通流的数据特点,采用了状态演化模式挖掘的框架对其进行挖掘,提出了一种交通流量模式和规则发现的方法,并且通过实验对这种方法进行了验证。

关键词:状态演化模式挖掘;线性分段化;k 中心点法;GSP
中图分类号: TP311.13 **文献标识码**:A

Application of state evolution patterns mining in traffic flow forecasting

YAN Di, SONG Su
(*School of Electronic Information & Control Engineering, Beijing University of Technology, Beijing* 100022, *China*)

Abstract: The traffic flow forecasting is the hot spot in the research of the traffic flow guidance. In order to extract the characteristic law that the traffic flow changes, a framework of state evolution patterns mining was adopted aiming at the characteristics of the traffic flow data. A method of discovering traffic flow patterns and rules were presented and validated through experiments.

Key words: state evolution patterns mining; piecewise linearization; k-center algorithm; GSP

0 引言

在现代化城市交通系统中,智能交通系统已经成为近年来迅速发展的城市道路交通管理系统,它能够有效地对地面道路交通系统作统一的规划,可以最大限度地发挥现有道路的承载能力。交通流诱导是智能交通系统中极重要的一环,它所能解决的问题是实时动态交通分配。有效的交通诱导是以准确的交通流预报为前提的,所以交通流预测成为国内外专家关注的热点。

但是众所周知,道路交通系统是一种有人为痕迹的、时变的、复杂的非线性大系统,这个系统有高度的不确定性,这种不确定性来自自然与非自然的多种因素,因而给交通流的预测带来了很大的困难。交通流动态预测还处在发展阶段,尤其是交通流短期预测还无法实现令人满意的结果。但是,不管多么复杂的系统都必然存在一定的规律性,这种规律并不受某个时刻的交通流数值所影响,正如一些研究学者[1]发现了交通流具有分形的特征,我们完全可以通过合适的手段找出这种规律性。

数据挖掘作为一类新型的数据分析方法发展非常迅速,几乎所有的数据都可以进行数据挖掘。状态演化模式挖掘[2]是数据挖掘技术的一种,它适用于具有时间维特征的序列。这种挖掘方法把序列看作一个有序的状态集合,每个状态会演化到下一个状态,状态的定义可以是趋势、偏差、分类规则和关联规则。交通流量数据是典型的实数型时间序列数据,针对这种类型的时间序列数据的挖掘技术还不是十分成熟。

本文结合张保稳博士等人所研究的状态演化模式数据挖掘框架,就具体的交通流量预测问题,提出了一种交通流量模式和规则发现的方法。

1 状态演化模式数据挖掘框架

在交通流中,状态体现的是一种变化趋势,即交通流的改变量,由变化时间长度与变化斜率来表达这种变化趋势,用符号表示为 $State = [Lw, Slope]$,其中,Lw 代表变化时间长度,$Slope$ 代表的是斜率。

状态演化模式数据挖掘的框架[2]如图 1 所示。

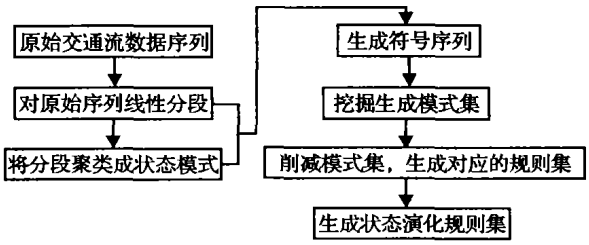

图 1 状态演化模式数据挖掘框架图

由图 1 可见,其具体操作分三步进行:

1) 由原始时序序列生成状态演化序列

因为序列数据挖掘比较成熟的部分是事务数据库的序列模式挖掘,所以解决思路是将时间序列数据转化为一般意义的事务序列,即将原始时序序列符号化,生成符号序列(每个符号代表的是一类的状态变化模式)。本文采用了与原框架不同的符号化方法。

2) 对状态演化序列进行关联规则挖掘,找出频繁出现的连续状态演化子序列

本文采用的是一种比较成熟的序列挖掘算法——GSP (Generalized Sequential Pattern)算法,根据 Apriori 性质进行挖掘,产生所有的强规则。

3) 将挖掘出的频繁状态演化子序列进行模式与规则的削减,最后生成有趣的强规则。在实际操作中会产生大量的

收稿日期:2004 - 08 - 05;**修订日期**:2004 - 10 - 21

作者简介:颜镝(1980 -),男,黑龙江勃利人,硕士研究生,主要研究方向:数据挖掘; 宋苏(1963 -),男,江西永新人,教授,博士,主要研究方向:鲁棒控制、自适应控制、数据挖掘.

模式,其中有很多冗余的模式,同样的,规则中也存在这种冗余,所以我们通过这步操作来获得简洁的、有效的规则。

1.1 由原始时序序列生成状态演化序列

1.1.1 线性分段化

我们要研究的数据构成的时间序列是一种非平稳序列,变化非常复杂。由于线性化分段方法有很好的形态表达和分割能力,而且这种方法可以很好地滤除噪音和进行数据抽象,所以我们采用这种方法对交通流数据进行分段化处理。

实现分段化的方法主要有窗口平移分段,自上而下分段和自下而上分段。本文采用自下而上的分段,并且针对本文具体情况对这种分段法[3,4]做了一定的改动。这种分段法将一个时间序列 TS(包含 n 个取值点)分为 k 个线性分段,其中 $k = n/3$,这时候每个分段包括3个点,最后的分段可以包含更多的点。分段以后利用一元线性回归来对每个分段中的数据点进行拟合,拟合得到的线段并不能完全反映原有的序列,我们引入了分段平均误差来表示这部分残余误差:

$$e_i = (1/j)\sum_{m=1}^{j} d_m^2$$

其中,d_m 表示每个点到拟合以后的线段的垂直距离。

一般来说,这种误差有一定的不一致性,我们用 B_k 来表达这种不一致性。B_k 定义为:

$$B_k = \left(\frac{1}{k-1}\sum_{i=1}^{k}(e_i - \bar{e})^2\right)^{\frac{1}{2}}$$

原算法是将现在得到的分段进行合并,但是经过实验,一些分段产生的峰值误差会严重地影响 B_k 的值,使得合并无法依据 B_k 来正确进行,因此我们必须对这些分段进行一定的处理。下面是具体的处理过程:

1) 将那些残差超过 0.5 的分段还原为原始分段,从而可以使 B_k 能够合理地表达这些误差的不一致;

2) 随机选择相邻的分段进行合并,并进行拟合,计算 $k-1$ 个分段得出的 B_k,使得其中一组合并以后得到分段的 B_k 最小,然后取这组合并后的分段为下一次分段的基础。

重复这样的操作,直到分段数达到我们的要求。

这种分段线性化的方法计算复杂度要低于自上而下分段法,而且能够很好地表达相对独立的变化模式。一般来讲,当分段数越少,偏差将会越大,我们无法估计合适的分段数,所以我们试验了一组不同分段数分段以后的结果,选择了一个相对稳定的分段结果。

1.1.2 聚类

分段化处理以后,我们可以获得一组形态各异的线段来代表原来的时间序列,每个线段与其他线段都会有所不同,如果我们对每个不同分段配置一个标识符,就根本无法发掘出有意义的模式,而且计算的代价将会呈指数级的增长,所以我们要对这些分段进行聚类,使得每一个符号可以独立代表一类状态变化模式。

本文要对交通流的数据序列进行聚类,由于数据采集线圈的误差和出现的异常记录,我们必须要考虑到这些可能的孤立点对聚类结果的影响,所以在本文中采用了 k-中心点的聚类算法[5,6]。算法如下:

输入 结果簇的数目 k,包含 n 个对象的数据库

输出 k 个簇,使得所有对象与其最近中心点的相异度总和最小

算法

随机选择 k 个对象作为初始的中心点;

Repeat

指派每个剩余的对象给离它最近的中心点所代表的簇;

随机地选择一个非中心点对象 O_{random};

计算用 O_{random} 代替 O_j 的总代价 S;

If $S < 0$ then O_{random} 替换 O_j,形成新的 k 个中心点的集合;

Until 不发生变化

1.2 GSP(Generalized Sequential pattern)算法[7]

简单介绍一下这种算法:

1) 第一次扫描以确定数据库中每项的支持度,将支持度大于我们规定的最小支持度阈值的项确认为频繁项,产生频繁一项集。再将频繁一项集构造成频繁一项序列。

2) 由频繁 k 序列集合 Lk 可以产生候选($k+1$)序列集合 Ck,候选($k+1$)序列集合中的每条候选序列均包含相同个数的项,且其项的个数均比其对应的种子频繁序列集合 Lk 中项的个数大一。

3) 在产生每一条候选($k+1$)序列的同时对其计数,当所有的候选($k+1$)序列均产生以后,算法根据每条候选($k+1$)序列的计数,确定哪些候选($k+1$)序列形成频繁($k+1$)序列,并作为下一步的种子集合。

4) 当由某个种子集合 Lk 产生的候选序列集合为空时,算法结束。

由于状态演化序列是有向的,所以在 GSP 算法的候选子集的生成步骤里,算法还要做出相应改动才能适用。

1.3 从模式到规则(模式集和规则集的削减)

我们所获得的每一个状态演化子序列都可以诱导出一系列的规则。如果模式很长,可以诱导出的规则是相当多的,因此,我们必须对获得的演化模式进行约简。这种约简的思路非常简单,就是如果短模式是长模式的子串,那么就删除这个短模式,直到模式集没有这种特征的短模式存在为止。

2 基于规则的预测

提取状态演化的规则是为了找出交通流量变化的规律,最终还是为预测服务。

预测步骤如下:

1) 将要预测的交通流时序数据进行线性分段化,将得到的状态矢量按时序排列。

这里采用的线性分段化要求与挖掘时符号化的方法参数相同,以保证相同的标准。

2) 计算每一个状态矢量与各个聚类中心的距离,把它归属到距离最小的类中,由此生成状态演化符号序列。

3) 参照规则库中的规则,从序列开始进行符号匹配,如果规则前件满足,则将规则后件存入预测集,并且输出相应的支持度和置信度。

3 实验

我们采集的是长虹桥南内环的交通流量数据,采集间隔是十分钟。图 2 是流量图。经过线性分段化以后,得到图 3,我们可以直观地发现此图能较好地描绘原始流量序列的形状特征。

我们选择聚类簇为 10,最小支持度为 3%,对这组线段序列进行聚类,然后绘制由聚类中心还原的时间序列分段,如图 4 所示。

在聚类过程中,每个类别都被赋予一个类标识(用大写字母代表),即可得到需要的符号序列。在经过 GSP 算法对得到的符号序列进行挖掘以后,我们发现在给定的参数条件下能够发现一些有意义的频繁状态演化模式。我们选择挖掘出的几条规则作为验证对象,如表 1 所示。

表 1　挖掘出的几条规则

规则前件	规则后件	规则支持度(%)	置信度(%)
E D	E	4.41	100
D E	E	3.03	100
F D	E	3.03	100
E A	E	5.30	100
E I	E	4.55	100

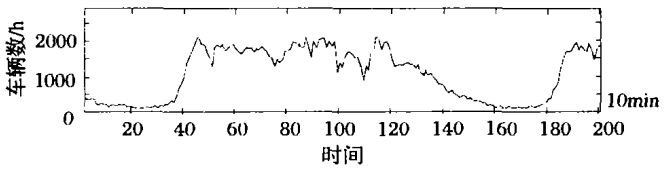

图 2　交通流量图

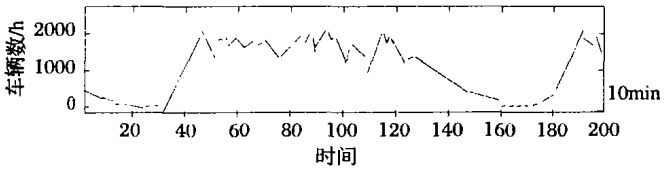

图 3　线性分段后获得的离散的符号序列图

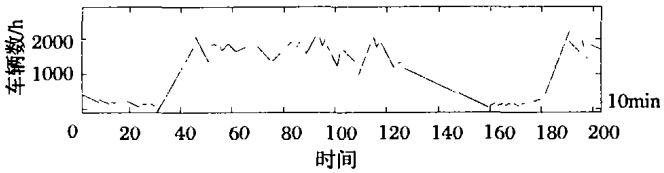

图 4　由聚类中心还原所得的时间序列分段表示

我们用 2002 年 11 月 11 日至 17 日的朝阳门桥北的交通流量数据进行了验证。情况如下:

规则(ED⇒E)前件匹配到 9 次,后件预测结果预测对了 7 次,有 2 次错误,预测的后验有效度为 77.8%。

规则(DE⇒E)前件匹配 7 次,后件预测结果预测对了 3 次,4 次错误,后验有效度为 42.9%。

规则(FD⇒E)前件匹配 4 次,后件预测结果预测对了 3 次,1 次错误,后验有效度为 75%。

规则(EA⇒E)前件匹配 15 次,后件预测结果预测对了 11 次,4 次错误,后验有效度为 74.4%。

规则(EI⇒E)前件匹配 5 次,后件预测结果预测对了 5 次,后验有效度为 100%。

由此我们可以看到,通过这种挖掘方法发掘出几条有意义的规则,可以对未来的趋势作出一定的分析推测。

4　结语

本文在原有的状态推演数据挖掘框架之下,提出了一种对交通流量数据进行数据挖掘的方法,并且给出了挖掘的流程,而后在实验环节对这种方法的有效性进行了验证。

在线性分段化的环节,对分段算法进行了一定的改进,使得分段误差有明显降低。

挖掘出的规则是否有效在很大的程度上取决于每一步的参数选择,包括分段数、平均误差指数以及聚类的簇个数等,这个问题有待于进一步解决。

参考文献:

[1] 李作敏,黄中祥,张亚平. 高速公路交通流分形特性分析[J]. 中国公路学报, 2000, 13 (3).

[2] 张保稳. 时间序列数据挖掘研究[D]. 西北工业大学, 2002.

[3] DAS G, LIN K, MANNILA H, *et al*. Rule Discovery from Time Series[M]. KDD, 1998. 16 - 22.

[4] 李斌,谭立湘,章劲松, 等. 面向数据挖掘的时间序列符号化方法研究[J]. 电路与系统学报, 2000, 5(2): 9 - 14.

[5] HAN JW, KAMBER M. 数据挖掘概念与技术[M]. 范明,孟小峰,译. 北京:机械工业出版社, 2001.

[6] KAUFMAN L, ROUSSEEUW PJ. Finding Groups in Data: An Introduction to Cluster Analysis[M]. New York: John Wiley & Sons, 1990.

[7] AGRAWAL R, SRIKANT R. Mining Sequential Patterns: Generalizations and performance improvements[Z]. IBM Almaden Research Center, 1996.

(上接第 638 页)

比用 K-means 算法和 CEADC 处理后的精度有所提高,只有在 Glass 中,EADC 处理后的精度较差,这表明 EADC 是有效的。

表 1　Breast-cancer-wisconsin 采用 EADC 的离散化结果

属性名称	不重复样本数	划分点个数	划分点
Bland Chromatin	10	3	1.5, 2.5, 3.5
Bare Nuclei	10	3	1.5, 4.5, 9.5
Clump Thickness	10	3	1.5, 3.5, 5.5
Marginal Adhesion	10	3	1.5, 5.5
Mitoses	9	2	1.5, 3.5
Normal Nucleoli	10	2	1.5, 7.5
Unif. of Cell Size	10	2	1.5, 5.5
Unif. of Cell Shape	10	3	1.5, 3.5, 7.5
Single Epithelial Cell Size	10	3	1.5, 2.5, 4.5

表 2　3 种方法在分类结果上的比较

数据集	K-means	CEADC	EADC
Breast-Cancer	0.889	0.762	0.986
Glass	0.593	0.487	0.450
Iris	0.800	0.908	0.976
Thyroid-disease	0.751	0.632	0.839

需要指出的是,算法 EADC 可以用于多种需要离散化连续属性的知识发现问题中,但由于它的计算量较大,因而适合于小规模数据集或规模缩减后的数据库。

参考文献:

[1] KURGAN LA, CIOS KJ. CAIM discretization algorithm[J]. IEEE Transactions on Knowledge and Data Engeering, 2004, 16(2): 145 - 153.

[2] USAMA FM, KEKI IB. Multi-interval discretization of continuous-valued attributes for classification learning[A]. Proceedings of the 13th International Joint Conference on Artificial Intelligence[C]. San Mateo, CA: Morgan Kaufmann, 1993, 2.1022 - 1027.

[3] 李刚, 童频. 基于混合概率模型的无监督离散化算法[J]. 计算机学报, 2002, 25(2): 158 - 164.

[4] HONG SJ. Use of contextual information for feature ranking and discretization[J]. IEEE Transactions on Knowledge and Data Engeering, 1997, 9(5): 718 - 730.

[5] 苗夺谦. Rough Set 理论中连续属性的离散化方法[J]. 自动化学报, 2001, 27(3): 296 - 302.

[6] COVER TM, THOMAS JA. Elements of information theory[M]. New York: John Wiley & Sons, 1991.

[7] CLARKE EJ, BARTON BA. Entropy and MDL discretization of continuous variables for Bayesian belief networks[J]. International Journal of Intelligence Systems, 2000, 15(1): 61 - 92.

[8] ISHIBUCHI H, YAMAMOTO T. Deriving fuzzy discretization from interval discretization[A]. The IEEE International Conference on Fuzzy Systems[C], 2003. 749 - 754.

[9] BLAKE CL, MERZ CJ. UCI repository of machine learning databases [DB/OL]. http://www.ics.uci.edu/ ~mlearn/MLRepository.html, 1998.