流式数据快速处理与分析方法研究

张露露

2017 年 12 月

中图分类号 TQ028.1

UDC分类号： 540

流式数据快速处理与分析方法研究

作 者 姓 名 张露露

学 院 名 称 计算机学院

指 导 教 师 计卫星

答辩委员会主席 石峰教授

申 请 学 位 工学硕士

学 科 专 业 计算机科学与技术

学 位 授 予 单 位 北京理工大学

论 文 答 辩 日 期 2017 年 12 月

Fast Processing and Analysis of Streaming Data

Candidate Name:	Zhang Lulu
School or Department:	Computer College
Faculty Mentor:	Prof. Ji WeiXing
Chair, Thesis Committee:	Prof. Shi Feng
Degree Applied:	M.Phil in Computer Science
Major:	Computer Science and Technology
Degree by:	Beijing Institute of Technology
The Data of Defence:	December, 2017

流式数据快速处理与分析方法研究

北京理工大学

研究成果声明

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下进行的研究工作获得的研究成果。尽我所知，文中除特别标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京理工大学或其它教育机构的学位或证书所使用过的材料。与我一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此申明。

作者签名：_____　　签字日期：_____

关于学位论文使用权的说明

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：① 学校有权保管、并向有关部门送交学位论文的原件与复印件；② 学校可以采用影印、缩印或其它复制手段复制并保存学位论文；③ 学校可允许学位论文被查阅或借阅；④ 学校可以学术交流为目的, 复制赠送和交换学位论文；⑤ 学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘要

随着大数据时代的到来，信息技术的快速发展，流式数据的应用越来越广泛，包括智能电网、网络访问量监测、传感器网络、金融股票市场等等。在对流式数据的处理与分析中，数据压缩与异常数据检测是其中的两个重要的处理环节，也是研究的热点。本文分析了当前国内外各种数据压缩与异常数据检测的算法，针对流式数据数据量多，计算实时性要求高的问题，提出了加速算法。本文的主要研究成果如下：

一、提出一种改进的增量 LOF 算法，更加适合大规模流式数据的异常数据检测，可以解决流式数据数据量太大导致的无法存储与计算的问题，减少检测数据点的计算时间。流式计算是将数据存储在内存中进行计算的，随着数据点迅速连续地产生，数据量越来越大，会造成存储困难和计算量越来越大的问题。因此，本文提出一种改进的增量 LOF 快速检测算法。该算法采用了将空间划分为网格的方法，将数据点映射到网格中，并进行一系列变换与处理，大大简化存储空间与计算量，从而可以有效解决需要进行计算的数据点的存储问题，同时减少每次数据记录到来后的检测时间，提高计算效率。本文介绍了网格的定义，数据点如何映射到对应的网格中，以及网格的特征向量，如何用网格来代替其中的数据记录进行计算等相关内容。并且，重新定义了 LOF 算法中的相关定义，使其能适应网格代替数据点的计算。通过大量的实验证明，在大量的流式数据的计算中，本文的算法的内存使用量更少，速度更快。

二、对分段多项式压缩算法进行改进，可以对流式数据进行快速压缩，提高压缩速度，保证压缩处理的实时性。数据压缩最常用的是多项式拟合算法，该算法计算过程中涉及多次矩阵运算，计算时间较长。本文首先选择滑动窗口算法来进行流式数据的在线压缩，然后针对时序数据的特点，对周期采样的时序数据采用缓存的方式，减少重复计算量，加快计算时间。对非周期时间采样的时序数据，采用增量计算的思想，临时保存每次计算的中间结果，作为下一次计算的中间结果使用，减少了计算量，提高了计算效率。通过大量实验证明，本文提出的加速方法，在不同的多项式次数和不同的拟合误差下，都可以有效减少压缩时间，保证流式数据压缩计算的实时性。

关键词： 流式数据；异常数据检测；数据压缩；大数据分析

Abstract

With the advent of the big data era and the rapid development of information technology, the application of streaming data is becoming more and more widely, including smart grid, network traffic monitoring, sensor network, financial stock market and so on. Data compression and abnormal data detection are important parts in data processing. This paper analyzes various algorithms of data compression and abnormal data detection both at home and abroad. Because of high data volume and real-time requirements, an acceleration algorithm is proposed for streaming data. The main research results of this paper are as follows:

(1) This paper presents an improved incremental LOF algorithm which is suitable for the detection of abnormal data in large-scale streaming data. This algorithm can solve the problem of storage and calculation which is caused by too large amount of data in streaming data, reduce the calculation time of detecting data points, and detect abnormally fast. In streaming computing, data is stored in memory. As data points are generated rapidly and continuously, the amount of data is becoming larger and larger, causing storage problems and increasing computational complexity. Therefore, this paper proposes an improved incremental LOF fast detection algorithm. The algorithm divides the space into grids, which greatly simplifies the storage space and computation, so as to effectively solve the storage problem and reduce the computation time, thus improves the efficiency. This paper gives the definition of the grid, how the data points are mapped into the corresponding grids, and the eigenvectors of the grids, and how to replace the data records in the grids with grids. Also, the concepts in the LOF algorithm are redefined so that they can use grid calculations. Through a large number of experiments, the algorithm of this paper has less memory usage and faster speed in the calculation of a large amount of streaming data.

(2) In this paper, we improve the segmentation polynomial compression algorithm, which can compress the streaming data rapidly, increase the compression speed and ensure the real-time performance of the compression. The most commonly used method of data compression is the polynomial fitting algorithm, which involves multiple matrix operations in computation and takes a long time to compute. We discuss the characteristics of streaming time-series data and propose a method for faster fitting and compression. Polynomial fitting is one of the most

important approaches of time-series data compression, and the calculation process involves multiple matrix multiplication and inversion, which lead to high computation complexity. Due to the large number of time series instances, high performance and real-time processing of incoming data is highly demanded. Considering the characteristics of time-series data, especially for the periodically sampled time series data, this paper puts forward a polynomial fitting acceleration method. By reusing the intermediate calculation results, our approach can significantly speed up the polynomial fitting of both periodically sampled and aperiodically sampled time-series data. Experimental results show the effect of the proposed method.

Key Words: streaming data; abnormal data detection; data compression; big data analysis

目录

摘要 I

Abstract II

第 1 章　绪论 1

1.1　本论文研究的目的和意义 1

1.2　国内外研究现状及发展趋势 2

1.3　主要研究内容 4

1.4　章节安排 6

第 2 章　研究现状 7

2.1　异常数据检测及相关方法 7

2.1.1　基于统计的方法 8

2.1.2　基于聚类的方法 9

2.1.3　基于距离的方法 10

2.1.4　基于密度的方法 12

2.1.5　其他方法 12

2.2　数据压缩及相关方法 14

2.2.1　分段线性法 15

2.2.2　其他方法 16

2.3　流式数据的处理 17

2.4　本章小结 19

第 3 章　基于 LOF 的流式数据异常检测方法研究 20

3.1　LOF 方法及增量的 LOF 方法 20

3.2　改进的增量 LOF 快速检测算法 24

3.2.1　网格定义及划分 25

3.2.2 网格的特征向量 26
3.2.3 网格的坐标以及代表点 26
3.2.4 算法实现 28
3.3 算法加速分析 32
3.4 本章小结 33
第 4 章 流式数据分段多项式压缩的加速方法研究 34
4.1 数据分段多项式压缩 34
4.1.1 多项式拟合 34
4.1.2 最小二乘法 35
4.2 静态时序数据压缩加速方法研究 37
4.2.1 周期时间采样的时序数据 37
4.2.2 非周期时间采样的时序数据 39
4.3 流式时序数据压缩加速方法研究 41
4.3.1 周期时间采样的时序数据 42
4.3.2 非周期时间采样的时序数据 44
4.4 本章小结 46
第 5 章 实验 48
5.1 实验环境 48
5.2 异常数据检测对比实验 48
5.2.1 人工数据集的对比实验 48
5.2.2 KDD CUP-99 数据集的对比实验 50
5.3 压缩算法对比实验 52
5.3.1 周期时间采集的流式时序数据对比实验 52
5.3.2 非周期时间采样的流式时序数据对比实验 53
5.4 本章小结 55

结论 57

参考文献 59

攻读学位期间发表论文与研究成果清单 64

致谢 65

表格

3.1 网格的特征向量符号及其含义 26

插图

1.1 流式数据分析与挖掘系统模型 3

2.1 基于正态分布统计方法的异常数据检测 9
2.2 维数为 2 时某单元 C 及其 L_1, L_2 层示意图 12

3.1 一个二维数据集 21
3.2 $k = 3$ 时，点 q 的 k 邻域 22
3.3 两种网格坐标表示 27
3.4 二维网格，代表点为 (x_0, y_0)，任意一数据点 (x, y) 28
3.5 网格的 k 邻域的两种边界情况 29
3.6 内存使用量变化图 (横坐标为运行时间，纵坐标为内存容量) 33

4.1 生产数据平均分段下两种方法的耗时情况 38
4.2 生产数据非平均分段下两种方法的耗时情况 39
4.3 人体运动数据平均分段下两种方法的耗时情况 40
4.4 非周期时间采样的时序数据所用缓存数据结构 40
4.5 缓存区域的数据结构 44

5.1 人工数据集 RandomSet 49
5.2 在 RandomSet 上两种方法的异常数据检测结果 49
5.3 在 RandomSet 上两种方法的运行时间对比 50
5.4 在 RandomSet 上两种方法的运行内存变化 50
5.5 在 KDD CUP-99 数据集上两种方法的异常数据检测结果 51
5.6 在 KDD CUP-99 数据集上两种方法的运行内存变化 51
5.7 周期时间采样的生产数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差 53
5.8 周期时间采样的生产数据上的加速比，其中 k 是多项式次数，ε 是拟合误差 53

5.9 周期时间采样的 UCI 气体数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差 54
5.10 周期时间采样的 UCI 气体数据上的加速比，其中 k 是多项式次数，ε 是拟合误差 54
5.11 模拟非周期时间采样的 UCI 气体数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差 55
5.12 模拟非周期时间采样的 UCI 气体数据上的加速比，其中 k 是多项式次数，ε 是拟合误差 55
5.13 非周期时间采样的 UCI 人体移动传感器数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差 56
5.14 非周期时间采样的 UCI 人体移动传感器数据集上的加速比，其中 k 是多项式次数，ε 是拟合误差 56

第 1 章　绪论

1.1　本论文研究的目的和意义

近年来，随着社会各方面的快速发展，移动互联和社交网络的普及，全球的数据量以指数级快速增长，人类社会已经迈入大数据时代[1–3]。随着计算机信息技术和多种新兴技术如云计算、物联网、分布式计算的快速发展，对大数据处理和挖掘成为可能，从浩瀚数据中挖掘出有用的知识，成为了学术界和工业界的研究的热点。

大数据是什么，麦肯锡全球研究所给出的定义是[4]：大数据一种数据集合，这种数据集合规模很大，在获取、存储、管理、分析等方面大大超出了传统数据库软件工具能力的范围，并且具有海量的数据规模、快速的数据流转、多样的数据类型和较低的价值密度四大特征。大数据数据量巨大，已经超出传统数据库的存储能力，目前，个人计算机硬盘容量可达 TB 量级，一些企业的数据量接近 EB 量级。大数据的处理速度快，因为数据量庞大，如何在海量数据中快速得到有用信息是十分重要的。大数据的种类繁多，数据可以被分为结构化数据和非结构化数据，结构化数据包括常见的关系数据库存储的数据类型，非结构化数据是数据结构不规则的数据，如网络日志数据、音频数据、视频数据、图片数据等等。非结构化数据的存在对数据的处理能力提出了更高的要求。大数据的数据量大，而价值密度却低。一般而言，价值密度和数据总量的大小成反比。所以，如何从海量的低价值的数据中挖掘出更有价值的信息，是一个重要的研究内容。另外，大数据还具有分布极其不规律，有用的信息隐藏程度很深等特点。从海量数据中，可以提炼出大知识，从而可以对人类社会带来很大的价值，这是大数据带来的机遇与挑战。

相比于传统数据的处理，大数据的计算模式主要分为两种，批量计算和流式计算。批量计算是将数据先存储下来，再对数据进行分批次的计算，这种方式对数据计算的实时性要求不高，主要适用于不需要及时处理数据的场景，但是其处理数据的精度和全面性有保证。现在常用的批处理计算系统主要有 Apache 的 Hadoop 框架[5]。与之相反，流式计算不会先将数据存储起来，而是直接对数据进行计算，因此对数据的计算速度有很高的要求。流式计算一般直接对数据在内存中进行计算，实时性强，但对数据的处理的精确度要求较低。在流式计算中，只对数据进行一次读取，不存在再次读取情况，这是流式计算的“one-pass”原则。由此可见，批量计算和流式计算相互补

充，为了达到更好的处理效果，一般场景下，可以将两种计算模式结合使用，结合两者的优势来处理数据。

现如今，数据呈爆炸式增长，流式数据的应用越来越广泛，流式计算的重要性越来越多地显示出来。数据的流式计算可以广泛地应用于生活和生产的多个方面，如金融市场，天气气象领域，航空航天领域，传感器网络和流量监控领域等等。流式数据是指按照时间顺序不断增加的动态数据序列，具有潜在的无限体积，其计算的实时性要求高，对精度的要求比较宽松[6]。目前，主流的流式计算系统有 Twitter 的 Storm，LinkedIn 的 Kafka[7,8]，Yahoo 的 S4(Simple Scalable Streaming System)[9–11], 以及传统行业金融领域中比较知名的 StreamBase[12] 和 Borealis[13] 等。

由以上可知，流式数据作为大数据的一种重要的形态，在多个领域中有着非常广泛的应用前景，但该技术作为近几年快速发展应用的技术，仍然具有很大的挑战，主要在数据的收集、存储、处理和可视化上面。流式数据的实时性和庞大的数据量，连续快速到达的特点，以及在线分析的应用需求，带来了挑战，也带来了很多机遇[14]。

1.2 国内外研究现状及发展趋势

目前，随着各个领域中处理数据量的增大，流式计算模式的普遍应用，流式数据的处理越来越广泛，国内外很多大学和研究机构都对数据流的管理系统进行了研究，其中一种流式数据分析与处理的典型模型结构如图1.1所示[15]。从中可以看出流式数据的处理系统模型在理论上主要包括两大类，一类是流式数据的预处理，一类是流式数据的数据挖掘。

流式数据分析与挖掘系统模型的数据来源是流式数据的采集器，传感器是一种常见的采集器，比如要获取发动机的数据，则使用发动机上的传感器进行采集。流式数据的预处理包括几部分，拿到原始数据后，可以根据实际处理需要，对原始数据进行汇总、压缩、降维或者动态索引等操作，得到一个概要的数据集或者是近似的数据集。数据的降维主要发生在数据维数较高的场景中，将高维的数据映射到低维的空间，其中，主成分分析 (PCA) 方法[16] 是一种广泛应用的高维数据的降维方法，该方法通过正交变换，将一组可能存在相关性的变量转换为一组线性不相关的变量，减少数据关系之间的重叠性，通过少数几个主成分来表示多个变量之间的内部结构，重新组合成一组新的互相无关的综合指标来代替原来的指标。数据的压缩可以减少数据的存储量，动态索引可以更快地查找数据。这些预处理操作可以降低时间复杂度和空间复杂

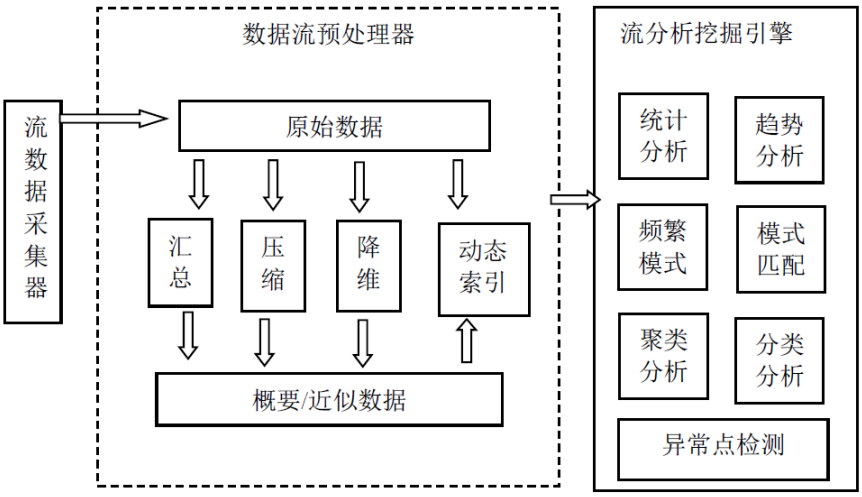


图 1.1　流式数据分析与挖掘系统模型

度，可以去除原始数据中一些不必要的特征，减少数据中的噪音，对后续的数据处理具有很重要的意义。在数据的挖掘处理中，对数据进行统计分析，可以很好地体现数据的特征，如常见的平均值、方差、均值等特征值的统计；对数据进行趋势分析，可以根据历史数据的变化情况，对将要到来的数据进行预测，多用于金融财务和股价方面的预测；对数据进行频繁模式的挖掘，可以找出频繁地出现在数据集中的模式，比如项集、子序列或者子结构；对数据进行模式匹配，可以在浩瀚数据中找到特定的数据；对数据进行聚类分析和分类分析，可以将无分类的无规律的错综复杂的原始数据分为多类，每个类里的数据具有相似性，反映相同的规律性和特性；而对数据进行异常数据检测，可以找出不符合正常行为模式的数据，在网络入侵、金融欺诈和机器监测等方面具有重要应用。

流式数据是连续的随着时间产生的数据序列，其中的数据可以是关系元组，也可以是数据项。目前在数据流研究领域存在多种数据流模型，分别有时序模型，现金登记模型和十字转门模型[14]。假设流式数据 X 的数据项为 $x_1, x_2, x_3, ..., x_i, ...$，数据项序列的下标 i 按顺序递增的，每个下标出现且只出现一次, 则

时序模型： 每个数据项都代表所描述数据的属性值，$X[i] = x_i$。

现金登记模型：多个数据项增量地描述数据的属性值，令 $x_i = (j, I_i)$，其中 $I_i > 0$，则有 $X_i[j] = X_{i-1}[j] + I_i$。

十字转门模型：多个数据项描述数据的属性值，与现金登记模型的区别是现金登记模型的数据的属性值是一直在增大的，而十字转门模型的属性值可以增大，也可以

减小。令 $x_i = (j, U_i)$，其中 U_i 可以为正数，也可以为负数，则有 $X_i[j] = X_{i-1}[j] + U_i$。

其中，时序模型在流式数据的分析和挖掘中应用比较广泛。时序数据的数据项是一个关系元组 $\langle t, s \rangle$，其中，t 为时间戳，s 为 t 时刻对应数据的属性值。数据流的数据具有时效性，因此数据携带有时间特征，通常为产生时刻的时间戳。流式数据要求对它进行实时性处理，数据项一旦流过就不复存在，不能再次进行访问，流式数据是源源不断产生的，具有潜在的无限体积，因此不能被全部存储下来。这些都是流式数据的特点。

1.3 主要研究内容

本文主要对流式数据的分析和处理进行研究。针对流式数据的特点，研究数据压缩和异常数据检测的常用算法，探讨在流式计算环境下，如何对算法更好地使用和优化加速，提高流式数据的处理效率。

由大数据的特点可知，数据的数据量巨大，价值密度很低，对数据进行存储，不但需要耗费比较昂贵的存储资源，而且直接存储价值较低的原始数据没有意义。原始数据常常有噪音干扰，将原始数据直接存储后进行分析和挖掘会消耗大量的计算时间和存储空间，而且也会影响算法的准确性和可靠性。对原始数据进行压缩，可以减少数据的存储空间，节省存储资源，减少存储和后续的计算代价，而且数据压缩不计较细节上的差异，用整体数据的一些特征点来刻画整个数据的主要形态，保留这些主体特征，反应了数据的自身特点。数据压缩可以通过控制压缩度来实现不同的精度层次上的搜索和匹配，更能符合大多数领域只关心数据的变化规律或模式的需求。因此，数据压缩是数据预处理环节中重要的一项内容，具有重要的研究意义。在流式计算中，数据流是源源不断持续增长的，在对数据进行压缩时，数据产生的速度很快，如果压缩花费的时间较多，数据的处理不及时，未处理的数据容易拥塞聚集，最后很可能造成数据丢失的严重后果，因此在流式计算中，对压缩算法进行加速有很重要的意义。

异常数据检测的任务主要是从数据集中有效地检测出异常数据，异常数据是指在数据集中不符合正常行为模式定义的数据模式，也称为离散点。异常数据在数据集中所占比重小，采样困难，但是其中一般存在重要的信息，或者如果不进行检测，会对后续的处理产生重要的甚至灾难性的影响。比如在工业中，机器发生故障，此时通过异常点检测可以及时发现故障机器，减少生产损失。在日常生活中，电信欺诈和信用卡欺诈等行为已经造成了恶劣的社会影响，通过对正常数据生成模型，通过算法检测

异常行为，可以有效地避免欺诈行为的发生，保护财产安全。异常数据检测还可以应用在网站维护方面，如果网站的访问量突然大幅度增加，可能原因是网站被黑客攻击或者被爬虫爬取数据等，所以异常数据的检测，在各个方面都具有实际的应用价值，对其进行研究具有重要的意义。目前为止形成了很多较为成熟且实用的方法，例如基于统计的方法、基于距离的方法和基于聚类的方法等。但是，由于时间复杂度和空间复杂度的影响，部分异常数据检测算法多为静态检测方法，适用于维度较低规模较小的数据集的离线检测，而流式数据大多维度高规模大，并且数据是源源不断到来的，数据量不断动态增长，行为模式可能随时间发生变化，所以在流式计算中，许多成熟的静态异常数据检测算法往往会出现检测结果发生偏差的问题，存在精度不高，运行效率低，不再适用等不足。

本文主要研究流式数据的数据压缩和异常数据检测的时效性问题，来满足流式数据数据规模大，实时性强的特征，以及在线处理的要求。具体来说，针对流式数据的压缩与异常数据检测，完成以下工作：

第一，针对增量 LOF 算法在流式数据处理中存在的问题，提出了一种改进的增量 LOF 快速异常数据检测方法，该方法将数据的分布空间进行划分，将数据点映射到对应的网格单元中，可以有效解决流式数据数据量巨大无法在内存中存储计算的问题，而且减少了计算量。在此基础上，将同一网格内的数据点集中到中心一点，对数据点进行基于密度的异常数据检测，这种方法一方面可以减少数据的存储量，另一方面减少了需要进行距离运算的对象，大大减少了计算量，具有很好的时效性和较少的空间消耗。

第二，根据流式数据在线压缩的要求，采用滑动窗口算法与分段多项式拟合算法来对数据进行压缩。针对时序数据的采样时间是否是周期的，分析在多项式拟合中计算多项式系数时所用的最小二乘法的过程，分别采用缓存方法和增量计算的方法，减少计算量，来对数据压缩进行加速。

第三，通过实验，验证了本文针对数据压缩和异常数据检测提出的加速算法的有效性，减少了计算时间，保证了流式数据处理的高实时性。针对基于 LOF 方法的快速异常检测算法，通过实验对比，验证了本文的算法可以很好地适用于流式计算环境中，并且取得较好的结果，并且处理数据的速度更快，运行内存更小。针对数据压缩的加速算法，通过实验，验证了本文提出的算法在不同多项式次数和不同误差率的情况下，可以很好地减少运行时间。

1.4 章节安排

本文主要分析异常数据检测和数据压缩在流式环境下的主流算法，探求算法的可行性和效率，基于这些算法提出改进，使计算提高效率，减少计算时间，达到加速效果。然后进行实验分析，保证其在实际中可以应用的可行性。本文的章节安排如下。

第一章，绪论，介绍了研究背景和意义，介绍了大数据的定义与特点，并介绍了大数据的计算模式，分为批量计算和流式计算，介绍了流式数据的特点和流式数据的处理计算模型。

第二章，针对异常数据检测方法和数据压缩方法，分别阐述了其应用背景，方法概述，和相关实现方法。这部分主要介绍异常数据检测方法和数据压缩方法的研究现状和主要使用算法。总结了异常数据检测的相关研究成果，介绍了主流的异常数据检测算法，包括基于统计的、基于聚类的、基于距离的和基于密度的方法。并且介绍了经典的算法，如 iForest 算法，LOF 算法等等。总结了经典的数据压缩的方法，包括奇异值分解法、分段线性法、符号表示法和频域法，分析了各种方法优缺点及应用范围。以及很多方法都是应用在静态数据集中的，在流式数据的计算中，该如何改进使用这些方法。

第三章，介绍了异常数据检测方法的 LOF 方法和适用于流式数据的增量 LOF 算法，并且基于这两种方法，提出了一种新方法，该方法相比于增量的 LOF 方法，减少了存储量和计算量，更适合流式数据。

第四章，介绍了分段多项式压缩的方法，首先分析了在静态数据集上如何进行加速计算，然后针对流式数据给出了滑动窗口算法与分段多项式相结合的压缩算法，最后通过分析其压缩过程，对于不同的数据类型，给出了不同的加速计算方法。

第五章，实验章节，主要分为异常数据检测的对比实验和压缩算法的对比实验。异常数据检测算法主要与增量的 LOF 算法进行对比，压缩算法与原计算方法进行不同多项式次数的时间对比与不同分段方法的时间对比。

最后对论文的主要研究内容进行了总结。

第 2 章　研究现状

2.1　异常数据检测及相关方法

随着科学技术的发展以及存储和计算能力的急剧增长，大数据的价值被日益体现了出来，数据挖掘技术被广泛地应用在了工业中和日常生活中。异常数据检测，作为数据挖掘算法中较为重要的一部分，引起了众多学者的关注和研究。本节介绍关于异常数据及其检测方法的相关内容。

异常数据是指在数据集中，不符合正常行为模式定义的数据模式[17]，比如异常类或者反类。与正常模式的数据相比，异常数据在某些方面的特性不同，异常数据通常是少量存在的，也被称为离群点，所以异常数据检测也称为离群点检测。异常数据可以被分为以下三类[17]，(1) 点异常，如果一个单独的数据点与其他的数据点相比是异常的，这种情况被称为点异常，点异常是最简单的一种异常形式。(2) 上下文相关异常，也被称为是条件异常[18]，上下文异常的数据，在全局看来可能不是异常的，但是从上下文的环境中看就是异常的，比如一年中每天的平均气温，假设十二月份的平均气温为零度，八月份的平均气温都为三十度左右，但是八月份某一天的平均气温只有零度，此时，从全局看，那一天的平均气温不是异常值，因为全年其它时间点有很多这样的值，可是从前后时间的温度来看，那一天的温度是异常的，这就是上下文相关异常的一个例子。上下文的环境一般由数据的某种属性来生成，比如语境属性和行为属性，上面举的例子中，一年中的某个时间就是数据的上下文属性，不同的时间下，数据点具有与时间属性相关的特性，这属于语境属性，而行为属性，是与数据产生者本身的行为有关系，比如全球的平均气温，不同地区的气温不同，气温与所在地区有关系，此时地区属性属于行为属性。(3) 集体异常[19]，顾名思义，集体异常不是某个数据是异常的，而是一批数据都是异常的，而在这批异常数据中，单个的数据是不被检测为异常的，因为与它一起的数据同它具有一样的特性。这种情况可能发生在工业机器的传感器采集的数据中，当机器在运行中发生故障，一段时间后进行了修复，那么从故障发生的那一刻到修复好的那一段时间中，传感器采集的数据都是异常的，这就是集体异常。点异常是所有异常中最简单也出现最多的一种异常方式，现在学术界关于异常数据的研究大部分都是检测点异常的数据。

异常数据检测方法可以分为基于监督的[20,21] 和无监督的[22] 的方法。基于监督的异

常数据检测方法的一种典型做法是，对正常数据分别进行学习和训练，建立一个正常数据行为模式，从而来进行异常数据的检测。而在有些数据集中，没有标记是否是异常数据，此时人工去进行标记类别是费时费力的，所以可以训练一个自适应的小模型，来对数据进行建模，从而检测异常数据，这就是无监督的异常数据检测。此外，异常数据检测方法可以分为基于统计的方法，基于聚类的方法，基于距离的方法和基于密度的方法，这些方法在下文中将会详细描述。

有关异常数据检测，一个重要问题是检测到的异常数据会以哪种形式输出出来，也就是检测后的输出是什么。一般来说，异常数据检测技术的输出有两种类型[17]，分数型和标记型。分数型会输出测试数据集中的异常分数的序列，异常分数表示数据实例被认为是异常值的程度，所以可以设置一个阈值，比该阈值大的数据可被认为是异常数据。标记型则是对每一个数据实例添加一个类别标记，这种方式相对于分数型来讲，比较简单直接，但是精确度没有分数型高。本文采用的基于密度的 LOF 方法，就是采用的分数型输出方式。

目前比较常用的异常数据检测方法基本分为以下几类：基于统计的方法，基于聚类的方法，基于密度的方法和基于距离的方法等。

2.1.1 基于统计的方法

基于统计的方法[23–25] 假设测试的数据集服从某个随机分布，为此数据集创建一个模型，然后通过不一致性测试来识别异常。数据的分布可以为正态分布，泊松分布，极值分布等等。

基于统计的学习方法是最早的异常数据检测方法，在 1969 年，Grubbs 采用了基于统计的方法来检测单变量的异常数据[26]，之后，Rousseeuw，Leroy 和 Barnett, Lewis 进一步对基于统计的方法进行研究，提出了三倍偏差的方法[27,28]，即一组随机分布的数据，数值大部分分布在期望值减去 3 倍标准差和期望值加上三倍标准差这个区间之内，数据不在这个范围内的概率仅有不到 0.3%。因此可以设置阈值来更精确地检测异常数据，这在正态分布中，就是著名的 3δ 准则。如图2.1所示是一个简单的基于统计的方法的例子，为了可视化方便，假设该数据 X 为一维数据，符合正态分布，记做 $X \sim N(100, 0.6)$，其中 $\mu = 100$ 为数学期望，$\delta^2 = 0.6$ 为方差，则其概率分布图如图2.1所示，根据 3δ 准则，可以设置两个阈值 $\mu - 3 * \delta = 97.879$，$\mu + 3 * \delta = 102.121$，当数据点的值在 97.879 到 102.121 之间，则认为该数据是正常的，如果数据点的值小

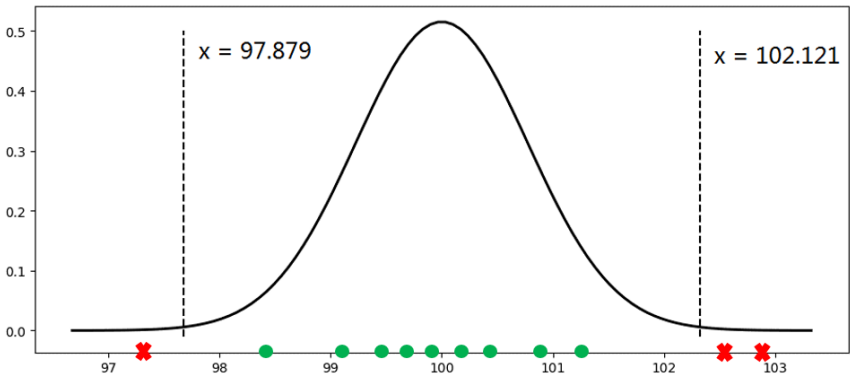


图 2.1　基于正态分布统计方法的异常数据检测

于 97.879 或者大于 102.121，则认为数据是异常数据。

基于统计的方法建立在统计学基础上，对分布参数进行估计，当数据量充足而且检验类型已知的时候，这种方法能发挥很好的效果。但是在现实应用中，数据集一般不会完全符合某一种理想的数学分布方式，分布方式有时会随着时间的变化而发生改变，而且人们不清楚数据真正的分布情况，特别是在高维数据的情况下，很难去估测数据的分布，所以这种方式比较适用于低维的，知道数据分布方式的情况[29]。

2.1.2　基于聚类的方法

基于聚类的方法[30] 是一种无监督的分类方法，聚类分析将数据集根据其相似性分为多个族群，数据点可以为测试集的向量值，或者是多维空间中的一个点[31]。直观地看，在使用聚类方法对数据集进行分类后，同一个族群中的数据点比其他类中的数据点更加相似。那么，在使用聚类方法对数据点进行分类后，如果在族群中的数据点，则与同一族群中的数据点相近相似，而不在分类的族群中的数据点，没有与其相近的点，根据异常数据点的定义，这些点可以看做是异常数据点。

基于聚类的异常数据检测方法根据某些假设，可以被分为三类[17]。(1) 假设正常的数据点必定属于数据聚类后的集群，而异常数据点不属于任何集群。基于此假设的方法将已知的聚类算法应用到数据集上，然后对数据集进行聚类，从而得到不属于任何集群的数据点作为异常数据点。此类方法的优点是比较简单，可以直接使用已有的较为成熟的聚类算法，缺点是此类方法不是优先寻找异常数据点，而是将重点放在数据集的聚类上面，先将数据集聚类，分为多个集群，然后才能找到异常数据点。(2) 假设正常的数据点是距离与其最接近的集群的形心较近的数据点，而异常数据点是距

离与其最接近的集群的形心较远的数据点。基于此假设的方法需要先对数据集进行聚类，然后对每个数据点，计算它到距其最近的集群的形心的距离，以此距离作为判断其是否异常的参照。但是如果异常数据自成一个集群的话，此类方法不能很好地找出异常数据。(3) 假设正常数据属于大的而且密集的集群，而属于小的或者稀疏的集群的数据点是异常数据。基于此假设的方法可以很好地解决异常数据自成一个集群的问题，它可以为集群的大小和密度设置阈值，集群的大小小于数量阈值或者集群密度小于密度阈值，即可将此集群中的数据点认为是异常数据点。

总的来说，基于聚类的异常数据检测方法[32] 是一种典型的无监督的方法，不需要使用者了解数据本身，它可以适用于多种复杂数据类型的数据，具有较好的应用性，但是，这类方法的性能主要依赖与所选取的聚类算法，聚类算法的复杂度如果比较高，那么聚类算法很可能成为异常数据检测算法的性能瓶颈，而且，异常数据的检测只是聚类结果的“副产品”，如果异常点可以自身聚集成一个集群的话，或者异常点被分类到某个很大的集群的时候，这种方法就无法有效地检测异常数据。

2.1.3 基于距离的方法

为了解决异常数据检测方法在检测大数据集时，只能有效处理二维的或者只有两种属性的数据集的问题，1998 年，Knorr 和 Ng 最先提出了一种基于距离的方法[33,34]。他们提到，一个数据集 T 中的对象 O，如果在数据集 T 中有至少 p 部分的对象与对象 O 的距离都大于 D，那么就称对象 O 是 $DB\,(p, D)$ 异常点，其中，$DB\,(p, D)$ 异常点是使用参数 p 和 D 的基于距离检测的异常点的简写。这种方法有效解决了当时只能处理低维的数据集的异常检测问题，可以适用于任意维度的数据集。并且，他们提出了几种简单的查找所有 $DB\,(p, D)$ 异常点的算法，包括基于索引的算法，嵌套循环的算法，和基于单元的算法。

基于索引的算法：该方法将异常点的定义进行简化，定义一个异常点的 D 邻域内最多有 M 个点，则查找异常点的问题可以通过搜索每个对象的邻域解决。首先对数据集 T 建立多维索引结构，然后对每个对象 o 以半径 D 进行搜索，一旦在 D 邻域内搜索到 $(M+1)$ 个对象，则停止搜索，并认定该对象 o 不是异常点，否则，o 是个异常点。多维索引结构主要有 $k-d$ 树[35,36]，$R-$ 树[37,38] 等，搜索一定范围内的时间复杂度的最好情况是 $\Omega(N^{1-\frac{1}{k}})$，$k$ 是数据的维数，N 是数据集的个数。当 k 增大时，搜索时间逼近 $O(N)$。所以，查找所有的 $DB(p, D)$ 异常点的时间复杂度为 $O(kN^2)$。但

是，这些时间只考虑了搜索的时间，没有考虑索引结构的建立的时间。实际上，索引结构的建立与维护是十分耗时的。

嵌套循环的算法：嵌套循环的算法可以避免基于索引的算法中建立索引结构的时间，它使用到了缓存，并将缓存分为两块，将整个数据集 T 分为多块。首先该算法将数据块读到第一个缓存中，对于其中某个数据点 t_i，将其属性值 $count(t_i)$ 赋值为 0，计算其与其它数据点 t_j 的距离，如果 $dist(t_i,t_j) \leq D$，则 $count(t_i)$ 做加 1 操作，如果 $count(t_i) > M$，则将 t_i 标注为正常的数据点，并且计算下一个数据点。否则，将其它数据块依次读入到第二个缓存中，依次计算两个缓存中数据的距离，直到全部数据块都计算过之后，第一个缓存中没有被标注的数据点就为异常数据点。该算法的时间复杂度为 $O(kN^2)$。

基于单元的算法：该算法将数据集的属性值空间划分为边长为 $l = \frac{D}{2\sqrt{k}}$ 的超立方体，k 为数据的维数，每个立方体称为一个单元，然后将数据集中的数据点映射到单元中。每个单元 C 有两层邻居单元，第一层 L_1 为紧邻 C 的一层单元格，第二层为紧邻 L_1 的外部的 $\left\lceil 2\sqrt{k}-1 \right\rceil$ 层单元格。如图2.2所示，是 $k=2$ 时的单元 C 及其 L_1, L_2 层的示意图。每个超立方体的边长为 $l = \frac{D}{2\sqrt{k}}$，则每个超立方体中的任意两个数据点的距离最长为 $\sqrt{k}l = \frac{D}{2}$, 两个相邻单元中任意两个数据点的距离最长为 D, 则单元 C 中的数据点与 L_1 层中的数据点的距离都不超过 D。而单元 C 中的任意数据点距离 L_2 层中任意数据点的最长距离最短是 $(\left\lceil 2\sqrt{k}-1 \right\rceil + 1) * l \geq D$。也就是说，$L_1$ 层中的数据点与 C 中的数据点的距离一定小于等于 D, 而 L_1, L_2 层包含了所有与 C 中的任意数据点的距离小于等于 D 的数据点。根据这些几何性质，每次计算单元 C 中的数据点时，只对 L_1, L_2 层中的数据点进行搜索计算即可。这种方法相当于对数据集进行剪枝，减少了计算量，提高了异常数据检测的效率。该算法的时间复杂度为 $O(m(2\left\lceil 2\sqrt{k} \right\rceil + 1)^k + N)$。

与基于统计的方法相比，基于距离的方法[39] 可以很好地适用于不满足某种特定的行为模式的数据集，而且在数据空间的维数比较高的时候，基于距离的方法也可以很好地检测出异常数据。但是对于参数 p 和 D 的确定，目前没有好的规律和算法，对于不同密度的数据集，D 的取值不同，结果也会很不相同，因此测试结果具有较大的不稳定性[40]。而且当数据的属性值为非数值时，两个数据点之间的距离无法直接计算，需要将属性值转换为数值。

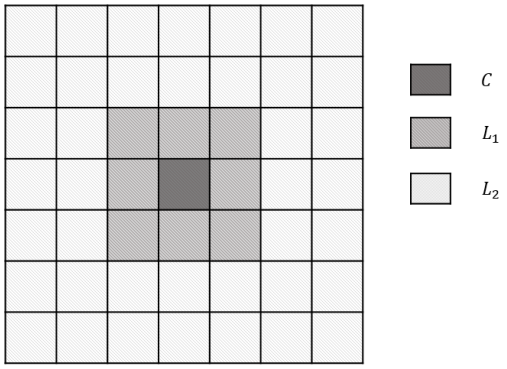


图 2.2　维数为 2 时某单元 C 及其 L_1, L_2 层示意图

2.1.4　基于密度的方法

之前提到的基于统计的方法和基于聚类的方法等，都是考虑全局数据的，而基于密度的异常数据检测方法[41]，是针对局部数据的，这种方法计算每个数据点的邻近的密度，如果一个数据点的邻近点比较稀疏，则认为该数据点为异常数据，如果一个数据点的邻近点比较稠密，则认为该数据点是正常的。常用的密度定义有两种，一种是距离某个对象 o 最近的 k 个点到 o 的距离和的平均值的倒数，即假设距离对象 o 最近的 k 个对象的集合为 S，则对象 o 的密度为

$$\rho = 1/\frac{\sum_{p \in S} d(o,p)}{k} \tag{2.1}$$

如果距离和的平均值小，则密度高。另一种定义是距离对象 o 为 d 的范围内的所有对象的个数。如果 d 邻域内的对象多，则密度大，但是该定义中的参数 d 不好确定，参数 d 如果太小，则很多正常点的密度也小，如果 d 太大，则异常点的密度也大。比较常用的基于密度的检测方法是 LOF 算法，该算法为了精确找出异常因子，定义了对象的局部可达密度，而且不仅仅考虑计算对象的密度，还考虑计算对象的邻域内的对象的密度。该算法为每一个对象计算出局部异常因子，表示对象的异常程度。该算法在下一章会详细介绍。

2.1.5　其他方法

大部分的异常检测方法，比如基于聚类的方法，基于分类的方法等，都是构建正常数据的模型，以这个模型来检测其它的数据是否异常，这类方法主要有两个缺点，

一是异常数据的检测不是靠异常数据的建模来检测的，而主要是靠正常数据的建模程度来优化的，正常数据的建模越好，异常数据的检测才能越成功。这样会导致异常数据的检测不如期待的好，可能会导致有些异常数据检测不出来，而有些正常数据会被误判定为异常数据。二是因为计算复杂度的影响，大部分方法在实际应用中局限于较低维度和小规模的数据。

异常数据有着“少而不同”的特征，也就是说，异常数据在数据集中是占很少一部分的，而且异常数据的行为模式、属性值与正常数据有着显著区别。这个特点是根据异常数据的定义得到的，异常数据也称为离群点，是数据集中明显偏离大部分数据的数据点，所以在数据集中的比重不大，而且属性值偏离常见属性值之外。基于异常数据的特性，刘飞和周志华提出了一种不需要检测正常数据，直接隔离异常数据的方法[42,43]。因为异常数据的“少且不同”的特点，异常数据是特别容易与其它数据分隔开的。异常数据“少”，所以更容易被分割开来，分割的路径也相对较短。异常数据“不同”，所以在早期的分割中，就与其它数据分离了，基于这些考虑，他们提出了 Isolation Forest 方法，简称为 iForest，这种方法对于一个给定的数据集，构建一个 iTree 的组合，然后在 iTree 中表现的路径短的数据点被认为是异常数据。该方法的主要思想是根据异常数据的特征，结合了机器学习中的随机森林算法，对数据样本的属性值进行了随机选取，从而对样本空间进行了随机划分，对数据的隔离过程表示成了对一棵树的建立过程。由于树的建立是随机的，所以建立多棵树，来提高准确率。这种方法的输入参数有两个，分别为构造的树的数目和数据的采样大小。以下为该方法的详细介绍。

iTree 定义：iTree 是一棵二叉树，全称为 Isolation Tree。如果 T 是 iTree 的一个结点，那么 T 要么是一个没有子结点的叶子结点，要么是有个两个子结点的内部结点，且这个内部结点包含一个属性 q 和一个分隔值 p。

建立 iTree 的过程如下，给出一个包含 d 维数据的数据集，随机选择一个属性和该属性上的一个值，将数据集进行分类。依次迭代分类，停止的状态为达到树的高度，数据集只剩一个，或者在数据集中的所有数据的值都相同。一个数据点的路径长度定义为这个数据点从根结点被分隔到叶子结点时所走的树的深度。

具体过程定义如下：

1. 从原始数据集中随机选取数据，得到一个子数据集，作为根结点；

2. 随机指定一个维度，选择该维度上的一个数作为切割点，在此维度上将数据划

分为两类，作为两个子结点；

3. 依次对子结点进行维度选择和切割，再生成子结点，直至树的高度达到一定值，或者子结点中只有一个数据值，或者子结点中的数据值都一样。此时，得到 iTree；

4. 以上过程进行 t 次，得到 t 棵 iTree。

此时，可以设置一个阈值，如果一个数据点在多棵 iTree 上的路径长度低于此阈值，则认定该数据点为异常数据。

与其他常见方法相比，iForest 方法的优点在于，采用了集成学习的思想，学习器由多个个体学习器构成，而且检测起来特别灵活，不需要构造正常数据的全部模型，只需要构建部分模型即可。iForest 方法中，没有距离和密度的计算，可以减少了大量的计算工作。该方法是线性复杂的，内存要求也很低，这对于大规模数据的计算有着很大的优势。而且该方法适合于部署在大规模分布式系统上进行加速运算，因为每棵树都是相互独立的。且树的数量越多，该算法越稳定。但是该方法是随机选取维度，很多维度的信息都没有使用，也会有噪音维度影响精度，如果树的数目较少，算法可靠性必定会降低，而且该方法对全局的异常点检测比较准确，但是对于局部的相对稀疏点，检测可靠度较低。

2.2 数据压缩及相关方法

由上一章可知，时序数据是最常见的数据流模型之一。时序数据在多个行业中都有应用，如股票金融业中每日股票的开盘价最高价等序列值，天气气象中每日气温降雨量等序列值，工业中电网每日的流量等等。大数据时代，数据流的数量庞大，直接对原始数据进行存储难度较大，容易造成昂贵的存储代价和传输代价，且大数据的数据量大，价值密度低，大多数情况下，都只关注数据整体的特性和发展的趋势，没有必要对每个数据进行存储。所以，在保留数据的主体的特性的前提下，对数据进行模式表示，可以对原始数据进行压缩，减少存储空间和计算的代价，也可以保留数据的价值与特性，也有助于后续的数据分析与可视化，符合大多数应用领域的需求。

在流式环境下，时序数据成为随时间顺序不断增加的动态序列，可能具有无限的体积。而对流式数据进行处理时，如果将数据全部缓存起来再进行处理，需要大量的存储资源和计算资源，这是不可能的，所以需要在数据流入的同时对数据进行处理。在无法获知全局数据的情况下对数据进行压缩，如果数据处理不及时很可能造成数据丢失，因此，对计算效率提出了更高的要求。

时序数据的模式表示主要分为四类，分别有分段线性法，频域法，奇异值分解法和符号表示法。

2.2.1 分段线性法

分段线性法[44,45]，顾名思义，是先用算法将数据按照一定规律分段，然后在每段数据上，用线性函数对其进行拟合。该方法可以看做是用一系列不重叠的线段，近似地表达一条曲线。其中，最早出现的也是最简单的分段线性法是 PAA 方法，PAA 方法将整体数据按照一定的长度平均分段，每段的数据用该段数据的平均值来表示。这种方法简单方便，但是比较粗糙，数据段与数据段之间不连续。之后的研究，为了解决这种问题，分别就分段与拟合两个方面，提出了多种不同的方法。

在数据分段方面，最简单的是平均分段，就是给定一个参数作为每段数据段的长度，对整体数据均匀分段。但是，平均分段不能很好地描述数据的特征，所以出现了非平均分段。非平均分段的算法有自顶向下算法，自底向上算法和滑动窗口算法，前两种算法是在静态数据整体上进行的，而滑动窗口算法可以在动态数据集上运行，支持在线分段。

自顶向下算法可以定义三个参数，分别为每个分段的最大误差，整个时间序列的最大误差，和分段数。该算法首先需要扫整个数据集，根据分割后两段的误差都处于较小等条件找到最佳的分割点，然后将原始数据序列分割为两个子序列，此时计算子序列的拟合误差，如果子序列的拟合误差小于误差阈值，则停止划分，否则继续划分子序列，直至所有的子序列的拟合误差都小于误差阈值。自底向上算法计算过程与自顶向下相反，该算法首先将整个数据序列划分为一个个相邻的短序列，这些短序列包含相邻的数据点，且拟合误差不超过阈值。将数据序列分为短序列后，计算相邻的序列合并后的拟合误差，将拟合误差最小的两个序列段合并，依次进行这个操作，直至任意相邻两段合并后的误差大于误差阈值。自底向上算法也可以人为定义每个分段的最大误差，整个序列的最大误差和分段数这三个参数。滑动窗口算法与前两种都不一样，它只可以定义每个分段的最大误差这一个参数，它的计算过程是有一个数据窗口，窗口中有一些相邻的数据点，如果窗口内的数据点的拟合误差小于阈值，则这个窗口增加窗口的宽度，同时接收新的数据点，并再次计算窗口内数据点的拟合误差，重复执行此操作直到窗口内的数据点的拟合误差大于阈值，此时该窗口内的数据为一个划分的数据段，并且开启一个新的窗口接收新的数据，开始下一个数据段的划分。在这

些算法中，自顶向下算法算法时间复杂度最大，但是划分最为准确，不支持在线动态划分数据序列，自底向上算法也不支持在线动态划分数据序列，但是时间复杂度低，算法简单，精度介于其余两种方法之间。滑动窗口算法的划分效果最差，但是算法复杂度低，计算简单，而且具有在线划分的特性。所以，在选择算法时，应该结合应用场景和需求，选择最适合的算法。

每段数据的线段拟合分为两种，一种是直线插补法，直接将数据段的开始点和结束点用线段相连，作为该数据段的拟合线段，然后相邻段之间首尾相连，整个数据段被分为一系列连接的线段。一种是直线回归表示，这种方法通过最小二乘法拟合段内所有的原始数据，相邻段可以不连续，精确度更高，与原始数据更为接近，较为常用。

在这些基础上，国内外研究人员提出了许多更好分段和拟合的方法。为了更好地进行分段，许多方法将数据本身的特点作为划分依据，将整体数据序列划分为更适合本身趋势变化的数据段，如基于极值点的[46,47]，基于斜率的[48]，基于特征点和趋势点的[49] 方法。基于极值点的方法找到数据的极值点来作为分段点，但是这种方法划分容易过于精确而忽略了趋势加速点，但是其计算很简单，性能也比较好，而且不依赖于阈值。为了弥补其划分过于精确的缺点，可以设定一个距离阈值，加上判断，当两个相邻极值点的数据差不大于这个距离阈值时，只选择其中一个极值点，这种方法减少了极值点的数量，减少了分段的数量，被称为基于特征点的方法。趋势点是数据变化较大的点，包括极值点和趋势加速点和趋势减速点。基于趋势点的方法需要计算某点和相邻点的角度、弧度或者余弦等，将大于特定阈值的点称为趋势点。为了更好地拟合数据段，不仅仅局限于用线段来拟合数据段，而是可以用多种函数拟合数据段，如多项式函数，指数函数等多种曲线函数，曲线函数能更好地拟合数据的变化，线段表示其实就是多项式曲线表示的一种形式。

2.2.2 其他方法

频域法[50–52] 是将一条时间序列看作是时间域上的一个信号，通过离散傅里叶变换[53] 或者离散小波变换[54,55]，将时间序列从时域空间映射到频域空间，将时域信号变换为频域信号。一个时间序列的时域信号可以看成各种周期扰动的叠加，时域分析只能反映信号的幅值随时间的变化的情况，很难表达出信号的频率组成和各频率分量的大小。而经过傅里叶变换，可以把任意的函数分解成为简单的周期函数的和，然后再将这些周期函数映射到频域内，此时，频域的自变量是频率，函数值为对应的振幅。

频域分析可以确定各周期的振动能量的分配。

符号表示法[56] 的主要做法是将时间序列离散化，映射到由不同符号组成的符号空间，将时间序列表示为有限符号的有序集合。其过程如下，首先，将一个时序数列 $X=\{x_1,x_2,...,x_n\}$ 进行 $z-score$ 标准化，标准化式子为

$$q_i=\frac{(x_i-mean\,(X))}{std\,(X)} \tag{2.2}$$

, 其中 $i=1,2,\ldots,n$，得到数列 $Q=\{q_1,q_2,...,q_n\}$。然后对其进行平均分段，并且每一段用平均值表示，即

$$q_i{}'=\frac{1}{k}\sum_{j=(i-1)k+1}^{ik}q_j \tag{2.3}$$

，得到序列 $Q'=\left\{q_1{}',q_2{}',...,q_w{}'\right\}$，其中 k 是压缩率，原长度为 n，转换后的数列长度为 w，$w=\frac{n}{k}$。然后得到这些离散的值后，将其映射到符号空间，转化为离散的符号。

奇异值分解法[57] 是一种基于统计概率分布的投影方法，可以对整个时序数据库进行整体表示。奇异值分解简称 SVD，如果有一个矩阵 A，对其进行奇异值分解，可以得到

$$A=U\Sigma V^T \tag{2.4}$$

，其中，假设矩阵 A 规模为 $m\times n$，则矩阵 U 的规模为 $m\times m$，矩阵 Σ 的规模为 $m\times n$，矩阵 V^T 的规模为 $n\times n$。矩阵 Σ 是一个对角矩阵，且对角元素从大到小排列，这些元素便是奇异值。由于排列在后面的许多对角元素接近于 0，所以，可以只保留比较大的 r 个奇异值，即只保留矩阵 U 的前 r 列，矩阵 σ 的前 r 行前 r 列，矩阵 V^T 的前 r 行。所以，只需要保存三个比较小的矩阵，就可以表示一个大的矩阵，实现了数据的压缩。该方法多用于图像压缩方面。

2.3 流式数据的处理

上文提到的异常数据检测方法是用于静态数据集的方法，在进行检测前，数据集是完整的，可以对数据集进行多次的计算和分析。而流式数据一般具有数据量大，数据维度高，存储昂贵的特点，对流式数据进行异常数据检测，需要对上面提及的异常检测方法进行改进或者研究新的算法。目前，针对流式数据的异常数据检测方法的思想可以分为两类，第一类是对静态方法的改进，这类方法是最直接的一种方式，可以

先创建缓存，积累一部分数据，对这部分数据进行训练，之后当新的数据到达时，对该数据进行检测，然后对模型进行更新和优化。该方法较为简单，但是如果流式数据有很大的动态变化，后来的数据的分布和行为模式发生很大改变，而模型是依据最前面的数据进行建模的，这种情况下该方法不能很好地检测后来的数据点。由此可见，该类方法适应性较差，比较适合数据的分布变化不是特别大的流式数据的检测。第二类是引入增量学习[58]和在线集成学习[59,60]的思想，增量学习是指模型可以保留以前学习到的知识，并且不断地从新的数据中学到新的知识，集成学习是指不只构建一个模型，一个检测器，而是构建多个模型，然后从多个模型中检测到结果，最后将结果合成得到最终结果，上小节提到的 iForest 方法就是采用了集成学习的思想。这类方法相比与对静态方法的改进，可以很好地适用于数据分布变化很大的流式数据，具有很高的泛化能力。

对流式数据进行异常数据检测的方法中，基于网格的方法[61]是常用的方法之一。基于网格的方法需要将数据点映射到网格中，计算网格的密度，当超过一定阈值时，认为这个网格是稠密网格，将相邻的稠密网格分成一类，网格密度小于一定阈值的网格则被认为是稀疏网格，当网格为稀疏网格时，可认为网格中的数据点为异常数据点。该方法可以满足流式数据“一次读取”的特征，每次数据点到来时只需要更新对应的网格的密度，处理速度较快，但是当数据的取值范围变大时，网格的数量也会变多，空间复杂度大大增加，此时进行异常数据的检测性能会下降。针对这个现象，相关文献中提到了改变网格划分，减少网格数量的方法。文献 [62] 中，使用了可变网格划分的思想，先将数据的每一维等间距划分，然后比较相邻区间段的相似性，并将相似性高的相邻区间段进行合并。在文献 [63] 中，针对网格的每个维度的划分粒度小，会使网格数量显著地增加，而划分粒度大，又会使聚类精度不准确，异常数据检测的精度受到影响的问题，引入了属性维半径，提出了一种可变网格的划分方法，每一维根据不同的维半径划分。文献 [64] 中，则提出一种动态划分网格的方法，首先生成初始网格，当网格的密度达到阈值的时候，选择方差最大的维度，将原网格划分为两个子网格，然后随着数据的逐步到来，对网格进行不断划分，直到一个合适的网格状态。但是基于网格的方法比较粗糙，精确度不高，当异常数据聚集时，容易认为其实正常的数据点。而一个聚类的外围数据点容易被认为是异常的。

2.4 本章小结

本章主要分为三部分，一部分介绍了异常数据检测的相关技术。首先介绍了异常数据的定义，异常数据的分类，以及异常数据检测方法的分类。然后，又从四个方面详细介绍了几种异常数据检测方法，分别有基于统计的方法，基于聚类的方法，基于距离的方法和基于密度的方法。最后，又介绍了工业上常用的 iForest 方法，并分析了该方法的优点与缺陷。第二部分介绍了现在的压缩算法，分别介绍了分段线性法，频域法，符号表示法和奇异值分解法的实现过程，并分析了三种分段方法，以及适用于流式数据压缩的滑动窗口算法。第三部分分析了当前的异常数据检测方法在流式数据上常用的方法。

第 3 章　基于 LOF 的流式数据异常检测方法研究

上节内容介绍了异常数据检测的几种方法，并介绍了如何对流式数据进行处理。可以看出，在这些异常数据检测方法中，基于统计的检测方法需要假定数据集服从特定的概率分布，然而在现实应用中，数据的分布情况一般较为复杂，而且通常随着时间会有变化，不是始终如一的，很难得到数据的分布。基于聚类的方法只能给出数据点是否是异常点的判断，不能量化数据点的异常程度，而且异常检测是聚类的“副产品”，很难对异常检测做到优化。而基于密度的 LOF 算法，可以很好地量化数据点的异常程度，并且能较好地识别异常点。

3.1　LOF 方法及增量的 LOF 方法

Breunig 和 Ng 等人发现，之前已存在关于异常检测的研究中，都把异常点当做是一个二分属性，对象只能分为是异常的或不是异常的，他们认为，在一些场景下，给出一个对象是异常点的度更有意义，这个度被称为局部异常因子，简称 LOF，并且提出一种算法，称为 LOF 算法[62]。为了证明 LOF 算法的优越性，他们提出了如下一个问题。如图3.1所示，是一个二维数据集，其中包括两个类 C_1 和 C_2，还有两个游离在外的数据点 o_1 和 o_2。从中可以看出 C_1 类中数据比较稀疏，C_2 类中数据比较密集。异常数据检测的目的，应该在整个数据集中找出异常数据点 o_1 和 o_2。可以很明显地看出，数据集的分布不规律，不能使用基于统计的方法，而使用基于聚类的方法，如果把 C_1 中的点聚为一类，则也会把 o_2 点聚类到 C_2 中，只能识别 o_1 点为异常点。如果使用基于距离的方法，可以看出，o_2 距离类 C_2 中的数据点的距离要小于 C_1 类中的某些点距离其邻近点的距离，如果距离阈值设置较大，则数据点 o_2 不会被检测为异常数据点，如果阈值设置较小，数据点 o_2 会被认为是异常数据点，但类 C_1 中的许多数据点也会被误认为是异常数据点。所以，基于距离的方法也不能准确地识别出异常点。基于密度的方法也存在这样的问题，数据点 o_2 周围点的数量不少于位于类 C_1 中的数据点，数据点 o_2 的密度不小于位于类 C_1 中的数据点的密度。而 LOF 方法可以解决这类问题。

LOF 方法中 LOF 的定义为局部异常因子，它依赖于对象周围邻居的分布情况，是邻居结点的密度与自身密度的比值，LOF 值越高，表明邻近对象都处于一个密集的状态，而本身密度较低，说明这个对象越可能是异常点，如数据点 o_2。LOF 越接

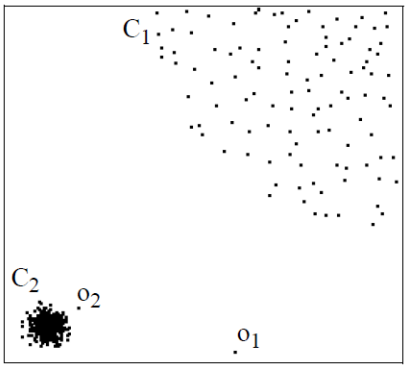


图 3.1　一个二维数据集

近 1 的话，表明该对象所处区域的密集程度与邻近对象越一致，说明这个对象越不可能是异常点，如位于类 C_1 中的数据点。关于 LOF 算法，解释如下。

定义 1 $d(p,q)$：对象 p 和对象 q 之间的欧式距离。

定义 2 $k-distance(p)$：对象 p 的 k 距离，是对象 p 和对象 o 之间的距离 $d(p,o)$，且满足

（1）在数据集中至少有 k 个对象 o'，满足 $d(p,o') \leq d(p,o)$；

（2）在数据集中最多有 $k-1$ 个对象 o'，满足 $d(p,o') < d(p,o)$。

其中，k 是任意的正整数。

定义 3 $N_{k-distance}(p)$：对象 p 的 k 近邻，其中的每一个对象 q 都满足 $d(p,q) \leq k-distance(p)$。一个对象 p 的 k 近邻，可以被记住 $kNN(p)$。

定义 4 $reach-dist_k(p,o)$：对象 p 到 o 的可达距离。如果 p 是 o 的 k 近邻内的点，则 p 到 o 的可达距离为对象 o 的 k 距离，否则，p 到 o 的距离是 $d(p,o)$，即，

$$reach-dist_k(p,o) = max\{k-distance(o), d(p,o)\} \tag{3.1}$$

如图3.2所示，$k=3$，则对象 p 的 k 距离就是 $d(p,q_3)$，对象 p 的 k 近邻 $N_{k-distance}(p) = \{q_1,q_2,q_3\}$，$reach-dist_k(q_2,p) = k-distance(p) = d(q_3,p)$，而 $reach-dist_k(q_4,p) = d(q_4,p)$。

定义 5 局部可达密度 (lrd)：对象 p 的局部可达密度定义为

$$lrd_k(p) = 1/\left(\frac{\sum_{o\in N_k(p)} reach-dist_k(p,o)}{|N_k(p)|}\right) \tag{3.2}$$

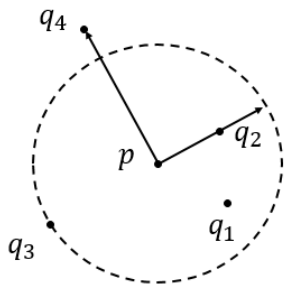


图 3.2　$k=3$ 时，点 q 的 k 邻域

定义 6 局部异常因子 (LOF)：对象 p 的局部异常因子定义为

$$LOF_k(p)=\frac{\sum_{o\in N_k(p)}\frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|} \tag{3.3}$$

对象的局部异常因子 (LOF) 代表了一个对象的异常程度，由式子3.3可知，一个对象的异常因子是该对象的邻居对象的平均密度与该对象的密度的比值，如果该值接近于 1，则表明该对象很可能是个正常的数据点，而该值越大，则越表明该对象可能是异常的。此时，对异常值的判断有了个程度的表示，于是可以通过控制阈值的大小，来检测不同粒度的异常数据。

当检测流式数据时，为了避免计算效率低下或者算法导致不正确的结果，LOF 算法需适应流式数据的特点。一般来说，LOF 算法用于流式数据可以有以下几种方式。

第一种方法，每次数据点到来后，对数据集进行一次静态的 LOF 算法检测，这种方法可以精确地检测出此数据是否为异常数据，但是计算量庞大。每次新数据点到来后，都要对整个数据集进行一次计算，计算每个点的 LOF 值。LOF 算法的时间复杂度是 $O(n\log n)$[62]，如果当前的数据集的数据点的个数为 N，则进行了 N 次检测，则总共的时间复杂度为[63]

$$O\left(\sum_{n=1}^{N} n\log n\right)=O\left(N^2\cdot\log N\right) \tag{3.4}$$

由此可以看出，这种方法虽然简单，当一个异常数据到来之后，可以立刻检测出来，但是，计算复杂度太大。

第二种方法对第一种方法进行了改进，不再是每当一个数据点到来后进行 LOF 算法检测，而是当一批数据点到来之后，再进行检测。数据流是动态的，源源不断的，

且无法全部缓存的，而 LOF 算法是静态的算法，适用于一个数据集的，那么可以将数据流依次缓存，定期地使用 LOF 算法，每当一定数量的数据点到达后，对整体数据点使用 LOF 算法检测[64]。这种方法比较简单，相当于把数据流分为了多个数据集，把动态的数据流变成了静态的数据集。但是计算复杂度依旧比较高，而且，这种方式对于检测具有延迟性，不能立刻检测出异常数据。

第三种方法引入滑动窗口的思想，每次只对窗口内的数据进行异常数据检测。这种方法相当于设置一个缓存区，将数据流缓存下来，缓存到一定的大小后，对其进行 LOF 算法检测。之后再进行缓存，周期性地进行 LOF 算法检测。这种方法计算复杂度不高，但是同样具有检测延迟的问题，而且，这种方法只检测缓存区中的数据，缓冲区的大小相当于对数据流的分割的块的大小。没有考虑历史数据的影响，很可能将正常数据点检测为异常的数据。

为了改进上面这些方法的不足，Pokrajac 提出了一种针对流式数据的增量式的异常因子检测算法（增量 LOF 算法）[63]，这种方法与第一种方法相似，每次新插入一个数据点后都需要计算局部异常因子，这样对数据的检测不具有延迟性，可以实时检测数据的异常度，但是不同的是，该方法通过数学推论，引入了 $kRNN$ 的定义，每次只更新计算新插入的数据点及其影响到的数据点，减少了计算复杂度。假设有对象 p，则 $kRNN(p)$ 定义为，所有的 k 邻域中包含 p 的对象点的集合，即如果对象 p 是对象 o 的 k 近邻内的一个点，则对象 $kRNN(p)$ 包含对象 o。当插入一个新的数据点时，该方法的伪代码[63] 如算法1所示。

该增量 LOF 算法的时间复杂度为 $O\left(N \cdot \log N\right)$，时间复杂度较低，而且在每次来一个数据点后都可以计算其局部异常因子，可以找到异常值，不具有延迟性，而且其准确度跟静态 LOF 算法一样，既可以考虑到历史数据的影响，又不会因为数据分段而造成将正常数据误认为是异常数据的情况。但是在流式环境下，该方法依旧存在问题，那就是计算时间与存储的问题。虽然这种方法在每来一个数据点后，都是只更新该数据点影响到的数据点，但是，全体数据量随着时间越来越多，一段时间后，达到海量的数据量，此时，如果不丢弃数据点，每次数据点 o 到来后，都需要计算数据点 o 的 k 邻域，kNN 和 $kRNN$ 序列，以及需要计算 k 邻域包含 o 的所有数据点，并且计算更新这些数据点的 kNN 和 $kRNN$，所以检测的时间也会越来越大，会造成计算缓慢。如果丢弃掉历史上的部分数据点，只选取近一段时期内的数据点，又会造成不精确的结果，甚至造成误报。而且会造成存储容量不够的问题。这些数据点都是存储

Algorithm 1: 增量 LOF 算法

Input: stream data $S\ p_1, ..., p_n$
Output: the LOF of data
while *insert(p_c)* **do**
  Compute $kNN(p_c)$;
  for $\forall p_j \in kNN(p_c)$ **do**
    compute $reach-dist_k(p_c, p_j)$;
  //Update neighbors of p_c
  $S_{update\_k\_distance} = kRNN(p_c)$;
  for $\forall p_j \in S_{update\_k\_distance}$ **do**
    update $k-distance(p_j)$;
  $S_{update\_lrd} = S_{update\_k\_distance}$;
  for $\forall p_j \in S_{update\_k\_distance}$, $\forall p_i \in kNN(p_j) \setminus \{p_c\}$ **do**
    $reach-dist_k(p_i, p_j) = k-distance(p_j)$;
    if $p_j \in kNN(p_i)$ **then**
      $S_{update\_lrd} = S_{update\_lrd} \cup \{p_i\}$;
  $S_{update\_LOF} = S_{update\_lrd}$;
  for $\forall p_m \in S_{update\_lrd}$ **do**
    update $lrd(p_m)$;
    $S_{update\_LOF} = S_{update\_LOF} \cup kRNN\{p_m\}$;
  for $\forall p_l \in S_{update\_LOF}$ **do**
    update $LOF(p_l)$;
  compute $lrd(p_c)$;
  compute $LOF(p_c)$;

在内存中的，一段时间后，数据越来越多，内存存储容量不够，会引起数据丢失和无法进行检测的后果。为了解决这个问题，我们提出了一种既可以在线检测，实时检测异常数据，又可以解决数据量很大内存容量不够计算时间长等问题的新的方法。

3.2 改进的增量 LOF 快速检测算法

由上文可知，流式数据的特性是数据量庞大，当时间越来越长后，接收到的数据量越来越多，数据存储在内存中进行异常数据检测，此时，会造成两个问题：(1) 存储空间问题，流式数据全部存储在内存中基本是不可能的。(2) 计算问题，当数据存储在内存中后，内存资源变少，会造成计算资源减少的问题，而且当数据量越来越多时，算法的运行时间也越来越长。为了解决这两个问题，本文提出一种改进的增量 LOF 快速检测算法，不但能够快速实时地计算新来数据点的局部异常因子，检测异常数据，也可以解决流式数据的数据存储问题。

该算法考虑到流式数据数据量巨大的特性和异常数据“少而不同”的特点，认为，选取一个较小的数据子空间，当该子空间与数据空间相比足够小时，子空间中的数据点可以认为是分布在同一位置的。所以，本算法主要是将数据空间划分为网格，并将映射到网格中的数据点，看做是落在网格中的同一位置，即将网格某位置的数据点，作为本网格的代表点，来代表落在本网格的所有数据点。并且调整 LOF 算法，使其适应于网格模式下的数据点。

本节内容主要介绍算法的实现，包括网格的定义与划分，网格的特征向量的定义以及坐标的选择，数据点如何映射到网格中，数据点映射到网格中后选取哪一点作为代表点，代表点如何反映该网格的数据点的数量，以及在算法实现过程中，关于 LOF 算法的相关概念的重新定义等。本算法可以分为两个模块，一个模块在线接收数据，并且将数据映射到相对应的网格中。一个模块来进行计算，对到来的数据所影响的网格进行更新，计算新来的数据点的异常因子。以下来详解介绍该方法的实现。

3.2.1 网格定义及划分

流式数据因为存储问题无法保存所有的原始数据，但是如果将流式数据所在数据空间划分为一个个的网格，将数据点映射到对应的网格中，更新网格的特征值，用网格来代替其中所有的数据，就可以解决这个问题。所以该方法将数据空间划分为网格，使用网格来保存数据的概要信息。假设流式数据 S 的维度为 D，流式数据的属性值空间为

$$S = S_1 \times S_2 \times ... \times S_D \tag{3.5}$$

其中，S_i 为第 i 维空间，$i = 1, 2, ..., D$。将数据空间划分为网格，假设第 i 维的长度为 l_i，划分的每一段的长度为 d_i, 可以将空间的第 i 维均匀地划分为等长的 n_i 段，即

$$S_i = S_{i,1} \cup S_{i,2} \cup ... \cup S_{i,n_i} \tag{3.6}$$

其中，$l_i = d_i * n_i$。此时，整个数据空间 S 被划分为网格，第 i 维被划分为 n_i 段，则整个数据空间被划分为 $\prod_{i=1}^{d} n_i$ 个网格。每个网格 G 由每一维度上的一段 S_{i,j_i} 共同构成，记作：

$$G = (S_{1,j_1}, S_{2,j_2}, ..., S_{D,j_D}) \tag{3.7}$$

其中，$j_i = 1, 2, ..., n_i$。

表 3.1　网格的特征向量符号及其含义

符号	含义
C	网格的坐标向量
w	网格的权重
$reachDis$	k 邻域的可达距离
knn	k 邻域包含的网格的集合
$krnn$	k 近邻包含该网格的网格的集合
lrd	网格的局部可达密度
LOF	网格的局部异常因子

3.2.2　网格的特征向量

在本文中，一个网格的特征向量被定义为 $(C, w, reachDis, knn, krnn, lrd, LOF)$，如表格3.1所示，其中 C 是网格的坐标向量，w 是网格的权重，代表该网格中数据点的数量，$reachDis$ 是网格的 k 邻域的可达距离，knn 和 $krnn$ 是两个序列，分别存储 k 邻域内所包含的网格的集合和所有的 k 邻域中包含该网格的网格的集合，lrd 是该网格的局部可达密度，LOF 是该网格的局部异常因子。

3.2.3　网格的坐标以及代表点

网格的坐标向量用来确定一个网格的位置。当数据点映射到网格中时，需要确定映射到的网格。确定一个网格也就是确定网格的坐标向量。假设一个网格的坐标向量为

$$g = (j_1, j_2, ..., j_D) \tag{3.8}$$

网格的坐标向量表示有以下两种方式，如图3.3所示，左边的坐标向量表示方式是将网格各个维度最小值的坐标值的向量作为网格本身的坐标值向量，也就是说，该网格表示的数据空间的各个维度的范围为 $[j_1 * d_1, j_1 * d_1 + d_1), [j_2 * d_2, j_2 * d_2 + d_2), ...[j_D * d_D, j_D * d_D + d_D)$。右边的坐标向量表示方式是将网格的形心的坐标作为网格本身的坐标值，也就是说，该网格表示的数据空间的各个维度的范围为 $[j_1 * d_1 - 0.5 * d_1, j_1 * d_1 + 0.5 * d_1), [j_2 * d_2 - 0.5 * d_2, j_2 * d_2 + 0.5 * d_2), ..., [j_D * d_D - 0.5 * d_D, j_D * d_D + 0.5 * d_D)$。

假设有一个二维空间的数据序列，数据空间划分网格的方式分别采用图3.3中所示的两种方式。

假设数据点 (x_1, x_2) 可以映射到网格 $g = j_1, j_2$，为方便计算，假设其误差为距离

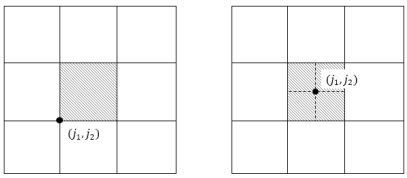


图 3.3　两种网格坐标表示

的平方和，即

$$f = \left((x - j_1 * d_1)^2 + (y - j_2 * d_2)^2\right) \tag{3.9}$$

则第一种方式中，网格的坐标向量是各维度表示范围的最小值的坐标值的向量，则其总误差如式子3.10所示：

$$\begin{aligned} R_1 &= \int_{j_2 d_2}^{j_2 d_2 + d_2} \int_{j_1 d_1}^{j_1 d_1 + d_1} ((x - (j_1 d_1))^2 + (y - (j_2 d_2))^2) \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{3} d_1 d_2 \left(d_1^2 + d_2^2\right) \end{aligned} \tag{3.10}$$

在第二种方式中，网格的坐标向量是网格形心的坐标向量，则其总误差如式子3.11所示，

$$\begin{aligned} R_2 &= \int_{j_2 d_2 - 0.5 d_2}^{j_2 d_2 + 0.5 d_2} \int_{j_1 d_1 - 0.5 d_1}^{j_1 d_1 + 0.5 d_1} ((x - j_1 d_1)^2 + (y - j_2 d_2)^2) \mathrm{d}x\mathrm{d}y \\ &= 4 * \int_{j_2 d_2}^{j_2 d_2 + 0.5 d_2} \int_{j_1 d_1}^{j_1 d_1 + 0.5 d_1} ((x - j_1 d_1)^2 + (y - j_2 d_2)^2) \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{4} * \frac{1}{3} d_1 d_2 \left(d_1^2 + d_2^2\right) \end{aligned} \tag{3.11}$$

由式子3.10和式子3.11可知，第二种方式中，数据映射到网格中用网格坐标代替后，其误差更小。所以，本文选择第二种方式来表示网格。第二种方式实际上是用网格的形心点的坐标向量作为网格的坐标向量。

当数据点到来时，确定其映射到的网格过程如下，假设新的数据点 $x = (x_1, x_2, ...x_D)$，则第 i 维的值 x_i，将其做如下转换：

$$j_i = \lfloor x_i / d_i + 0.5 \rfloor \tag{3.12}$$

从而得出数据点 x 所属的网格 $g = (j_1, j_2, ..., j_D)$

网格代表点的选取应该使得该点代表其他数据点时误差最小。如图3.4所示，是一

个二维网格，第一维长度为 d_1，第二维长度为 d_2，且 $d_1, d_2 > 0$。

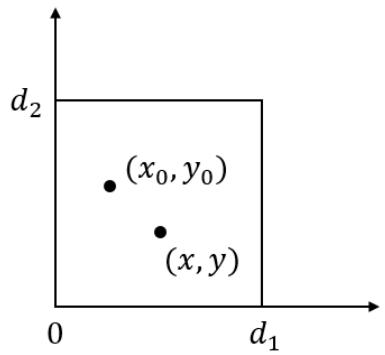


图 3.4　二维网格，代表点为 (x_0, y_0)，任意一数据点 (x, y)

假设网格代表点为 (x_0, y_0), 网格中任意一点为 (x, y)，且满足 $0 \leq x, x_0 \leq d_1, 0 \leq y, y_0 \leq d_2$。假设网格中数据点均匀分布，则其误差可定义为

$$\begin{aligned} R &= \int_0^{d_2} \int_0^{d_1} \left((x - x_0)^2 + (y - y_0)^2 \right) \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{3} d_2 \left((d_1 - x_0)^3 + x_0^3 \right) + \frac{1}{3} d_1 \left((d_2 - y_0)^3 + y_0^3 \right) \end{aligned} \tag{3.13}$$

因为 $0 \leq x_0 \leq d_2$，$0 \leq y_0 \leq d_2$，则当 $x_0 = \frac{1}{2} d_1, y_0 = \frac{1}{2} d_2$ 时，误差 R 最小。所以，代表点的坐标也就是该网格形心的坐标向量，这种情况可以扩展到多维空间。所以网格的坐标向量和代表点的坐标向量都是网格形心的坐标向量，因此，在网格的特征向量中，用 C 来代替网格的坐标向量和代表点的坐标向量，统称为网格的坐标向量。

为了区分不同网格中数据点的数量，引入权重的概念，也就是网格的特征向量中的 w，代表了该网格中数据点的个数。从以上内容可知，将数据点映射到对应的网格中，然后用该网格的具有权重值的形心点代替所有的数据点来进行下一步的计算。

3.2.4　算法实现

流式数据的数据量巨大，而且一般为高维数据。将数据空间划分为网格时，海量数据映射到网格中。此时，当网格相比于数据空间足够小时，同一个网格中的数据点，在空间上和属性上具有相似性，可以将这些映射到同一个网格中的数据点看做是一样的数据点，所以这些数据点可以用一个带有权重的数据点来近似表示。这个数据点是网格的坐标，而权重是网格内已存的数据点的个数。此时，LOF 算法中的数据点不再

是单个的数据点，而是具有权重的数据点，所以，对其进行重新定义。

定义 7 $d^{'}(p,q)$：网格 p 和网格 q 之间的距离，设网格 p 的坐标向量为 $(p_1,p_2,...,p_D)$，网格 q 的坐标向量为 $(q_1,q_2,...,q_D)$，则网格 p 和网格 q 之间的距离定义为

$$d^{'}(p,q)=(\sum_{i=1}^{D}(p_i-q_i)^2)^{\frac{1}{2}} \tag{3.14}$$

定义 8 w_p：网格 p 的权重，也可以称为网格 p 的密度，是网格 p 中数据点的个数。

定义 9 $k-distance^{'}(p)$：网格 p 的 k 距离，是网格 p 和网格 o 之间的距离 $d(p,o)$，且满足

(1) $\sum w_{o_i}<k$，其中 w_i 是网格 w_{o_i} 的权重，且满足 $d(p,o_i)\leq d(p,o)$。

(2) $\sum w_{o_i}+w_o>=k$，其中 w_{o_i} 是网格 o_i 的权重，满足 $d(p,o_i)\leq d(p,o)$，w_o 是网格 o 的权重。

此时有两种情况出现，如图3.5所示，一种是 $\sum w_i+w_o=k$，则网格 o 正好全部包括在网格 p 的 k 邻域内，且是网格 p 的 k 邻域的边界，此时对象 p 的 knn 包括网格 o。一种是 $\sum w_i+w_o>k$，此时网格 o 正好横跨在网格 p 的 k 近邻线上。如果网格 o 中有 100 个数据点，而网格 p 有 98 个数据点，$k=100$，此时，如果网格 p 的 k 近邻包括 o，则 k 近邻一共有 198 个数据点，极为不精确。为了计算的准确性，本文假设每个网格中的数据均匀分布，并为每一个网格 g 的 k 近邻网格的序列 knn 中加了一个属性 $Score$，称为比值，表示 knn 中的网格在网格 p 的 k 近邻中所占的比重。例如上面的问题，网格 o 在网格 p 的 knn 中，且其比值参数为 0.02，表示该网格中 $100*0.02=2$ 个数据点在网格 p 的 $k=100$ 近邻中。

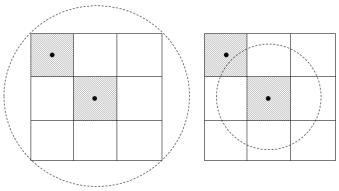


图 3.5　网格的 k 邻域的两种边界情况

定义 10 $Score(o,p)$：表示网格 o 在网格 p 的 k 邻域内所占的比值，满足 $0<Score(o,p)\leq 1$。

定义 11 局部可达密度 $\left(lrd^{'}\right)$：网格 p 的局部可达密度定义为

$$lrd_k^{'}(p) = 1/\left(\frac{\sum_{o\in N_k(p)}(reach - dist_k(p,o) * w_o * Score(o,p))}{|N_k(p)|}\right) \tag{3.15}$$

定义 12 局部异常因子 $\left(LOF^{'}\right)$：网格 p 的局部异常因子定义为

$$LOF_k^{'}(p) = \frac{\sum_{o\in N_k(p)}(lrd_k(o) * w_o * Score(o,p)) + lrd_k(p) * w_p}{|N_k(p)|\, lrd_k(p)} \tag{3.16}$$

基于以上相关定义，当数据空间被划分为足够小的网格时，映射到网格中的数据点具有相似性，将网格中的数据点用网格来代替，从而来进行异常数据检测。改进后的增量 LOF 快速检测算法如算法2所示。

改进后的增量 LOF 快速检测算法可以解决以下问题：

(1) 流式数据的存储问题。流式数据的数据量巨大，将其存储将消耗昂贵的存储资源。而将数据空间划分为网格结构后，网格的数量是确定的，对每个网格存储其特征向量，则无论有多少流式数据，存储的空间不变，在流式数据源源不断到来后，将数据点存储在网格结构中，总的存储量不会增加，可以解决流式数据数据量大，无法存储的问题。

(2) 流式数据的异常数据检测的计算时间问题。将流式数据的数据点映射到网格中时，计算量会大大减少。因为当新来数据点时，需要计算该数据点的 k 邻域，在原本的算法中，需要计算新来数据点与已有所有数据点的距离，并将所得距离排序，找到离新来数据点最近的 k 个点，而本文的算法中，分为两种情况，一种是新来的数据点映射到的网格中之前没有数据点，此时，根据网格的空间的相邻网格的特性，计算其 k 邻域，另一种是新来的数据点所映射到的网格已经存在数据点，此时，不需要重新计算 k 邻域，只需要更新网格的特征向量，包括其 k 邻域。当数据点几百几千个时，两者计算时间差别不大，但是当数据量远远超过网格的数量时，如数据达到上万百万以及更多时，本文的算法的计算量大大减少，计算速度很大幅度地提高，计算时间快速减少。其次，在存储方式上，网格根据其坐标向量可以使用数组结构存储，数据读取迅速，也可以减少计算时间。因此，本算法可以实现流式数据的快速异常数据检测。

Algorithm 2: 改进的增量 LOF 快速检测算法

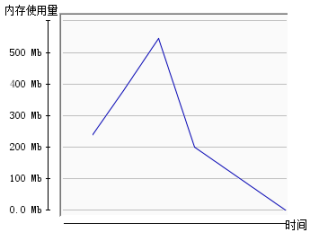
Input: stream data S $d_1, ..., d_n$
Output: the LOF of data
while *insert(d_c)* **do**
  Compute C_{d_c} for d_c;
  Get grid p_c by C_{d_c};
  if *grid p_c does not contain data* **then**
    $w_{p_c} = 1$;
    Compute $kNN(p_c)$;
    Compute $kRNN(p_c)$;
    for $\forall p_j \in kNN(p_c)$ **do**
      Compute $reach - dist_k(p_c, p_j)$;
  else
    $w_{p_c} = w_{p_c} + 1$;
    Update $kNN(p_c)$;
    for $\forall p_j \in kNN(p_c)$ **do**
      Update $reach - dist_k(p_c, p_j)$;
  //Update neighbors of p_c
  $S_{update\_k\_distance} = kRNN(p_c)$;
  for $\forall p_j \in S_{update\_k\_distance}$ **do**
    update $k - distance(p_j)$;
  $S_{update\_lrd} = S_{update\_k\_distance}$;
  for $\forall p_j \in S_{update\_k\_distance}$, $\forall p_i \in kNN(p_j) \setminus \{p_c\}$ **do**
    $reach - dist_k(p_i, p_j) = k - distance(p_j)$;
    if $p_j \in kNN(p_i)$ **then**
      $S_{update\_lrd} = S_{update\_lrd} \cup \{p_i\}$;
  $S_{update\_LOF} = S_{update\_lrd}$;
  for $\forall p_m \in S_{update\_lrd}$ **do**
    update $lrd(p_m)$;
    $S_{update\_LOF} = S_{update\_LOF} \cup kRNN\{p_m\}$;
  for $\forall p_l \in S_{update\_LOF}$ **do**
    update $LOF(p_l)$;
  compute $lrd(p_c)$;
  compute $LOF(p_c)$;
  $LOF(d_c) = LOF(p_c)$;

3.3 算法加速分析

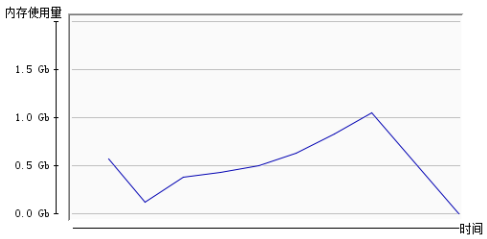
本算法是一种快速检测算法，主要从两方面可以减少计算时间。

(1) 算法计算量方面的差异。从上文分析可知，本算法与增量的 LOF 算法相比，主要变化在于计算的数据量的改变。在大数据时代，数据量呈爆炸式增长，据工信部统计显示，2016 年我国移动互联网接入流量达 936 万 TB，同比增长 123.7%，市场调研机构 IDC 预计，未来全球数据总量年增长率将维持在 50% 左右，到 2020 年，全球数据总量将达到 40ZB。由此可见，数据量是巨大的。在应用场景中，流式数据到来后，数据量越来越多，计算也越来越复杂，所以，计算量和存储量是一直随着时间增长的，而在本算法中，用空间中的网格来代替数据点，数据点的多少影响的只是网格的权重，当数据到来时，有数据的网格越来越多，当全部网格都有数据后，随着时间的增长，数据量变多，但是网格的数量不会增多，所以存储量和计算量也不会再加大。

(2) 计算过程中的差异。增量的 LOF 算法中，每来一个数据点，都是一个全新的数据点，相关的计算都要进行，其中包括计算其 knn 和 $krnn$。现有的计算数据点的 knn 的最有效的算法[65] 与计算 $krnn$ 的最有效的算法[66] 的时间复杂度都是 $O(\log n)$。而本算法中，如果该网格之前已经有了数据点，则与原方法的需要计算 knn 与 $krnn$ 相比，本算法只需要更新 knn 即可，而且更新的时间复杂度为 $O(1)$。异常数据点是较少的，而大部分数据点都会聚集，当数据密集分布时，流向同一网格的数据会较多，从而减少一定量的计算量。其次，本算法的主要目的是为了检测新来的数据点是否为异常数据点，当程序运行一定时间后，数据流趋于稳定时，如果新来的数据映射到的网格权值较大，异常数据值较小，那么表明该网格内的数据点很多，那么新来的数据点基本可以确定其不是异常数据点，从而不进行各种计算，只更新该网格的权值。但是，为了防止数据流发生变化，原先不是异常的网格一段时间后因为周围网格数据点的大幅度增多而变为异常网格，每隔一段时间后，可以更新一下每个网格的特征向量。如图3.6所示，是用大小为 100 的小数据集做的小对比实验，从中可以看出，本算法的内存使用量峰值大约为 520MB，而增量的 LOF 算法的内存使用量峰值大约为 1.0GB，而且可以看出，本算法的平均内存使用量更少，而运行时间也更短。



(a) 改进的增量 LOF 的快速检测算法



(b) 增量 LOF 检测算法

图 3.6　内存使用量变化图 (横坐标为运行时间，纵坐标为内存容量)

3.4　本章小结

由上可知，LOF 算法可以检测数据点的局部异常因子，根据局部异常因子来判断此数据点是否是异常数据点。在流式环境下，可以使用增量的 LOF 算法，只计算更新新来数据点及其影响到的数据点，计算这些数据点的局部异常因子。但是流式数据数据量巨大，数据维度高，该方法无法解决大规模数据的计算与存储问题。本文提出一种新算法，将增量的 LOF 算法与网格相结合，并设计了网格的特征向量，以网格代替其中的数据点来进行计算。该算法不但可以减少存储空间，也可以减少计算量，实现流式数据的快速异常数据检测。为了验证本算法的可实施性和精确性以及加速效果，将在第五章进行实验验证。

第 4 章　流式数据分段多项式压缩的加速方法研究

由第二章的内容可知，流式数据的压缩过程无法将数据流整个存储起来再进行压缩，当数据源源不断到来时，如果处理速度慢，则容易发生数据堆积，很可能造成数据丢失。而且，流式数据的计算只能对数据进行一次读取，之后就会丢弃。所以，对数据的压缩过程进行加速，提高数据拟合的效率，可以保证流式数据的实时性，可以减少数据丢失的可能。

时序数据是流式数据的一种重要的形式，与适用于特定数据形式的现金登记模型和十字转门模型相比，时序模型的数据表达具有普遍性，其应用更加广泛。大部分时序数据都是周期时间采集的，时间戳的间隔是相同的。本文针对时序数据的特点，分别对周期时间采样的与非周期时间采样的流式时序数据提出多项式拟合加速的方法，从而提高计算速度。

4.1　数据分段多项式压缩

分段线性法具有简单直观的特点，是时序数据压缩较为常用的一种方式。为了更好地拟合原始数据，每段的数据可以用多项式来进行拟合。

4.1.1　多项式拟合

时序数据是带有时间戳的数据，是一个按照时间递增的顺序排列的数据序列，假设有时序数据序列 $\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$，其中 x_i 是与时间标签有关的值，通常，$x_i=i$。其多项式拟合函数为

$$f(x)=a_kx^k+a_{k-1}x^{k-1}+\cdots+a_1x+a_0 \tag{4.1}$$

其中，k 是多项式的次数，a_i 是多项式的系数。

多项式拟合的目标是通过调整 $f(x)$ 的系数，使拟合函数更好地逼近原始数据，而在进行数据存储时，只需要存储这段时序数据的真实的开始时间和结束时间，以及多项式的各个系数即可。此时，用存储多项式的系数代替存储原始数据点，大大地减少了存储量，可以达到数据压缩的目的。

4.1.2 最小二乘法

最小二乘法是数学中的一种优化方法，该方法可以通过最小化数据段与拟合函数的误差平方和，从而来得到数据的最优拟合函数。因此在求解多项式函数时，可以使用最小二乘法求解拟合多项式函数，其过程如下。

(1) 原始数据段的点为 $\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$，拟合函数为 4.1，则此时的误差平方和为：

$$R^2=\sum_{i=1}^{n}(f(x_i)-y_i)^2 \tag{4.2}$$

(2) 当拟合多项式最接近原始数据时，误差平方和最小，则有：

$$\sum_{i=1}^{n}(f(x_i)-y_i)^2=min \tag{4.3}$$

(3) 将数据点代入，可得：

$$\sum_{i=1}^{n}[(a_kx_i^k+\cdots+a_1x_i+a_0)-y_i]^2=min \tag{4.4}$$

(4) 为了解多项式的系数 a_i，将式子4.4对 a_i 求偏导数，可以得到：

$$\begin{aligned}
\frac{\partial R^2}{\partial a_0}&=2\sum_{i=1}^{n}[(a_kx_i^k+\ldots+a_1x_i+a_0)-y_i]=0\\
\frac{\partial R^2}{\partial a_1}&=2\sum_{i=1}^{n}x_i[(a_kx_i^k+\ldots+a_1x_i+a_0)-y_i]=0\\
&\cdots\\
\frac{\partial R^2}{\partial a_k}&=2\sum_{i=1}^{n}x_i^k[(a_kx_i^k+\ldots+a_1x_i+a_0)-y_i]=0
\end{aligned} \tag{4.5}$$

(5) 将式子4.5改写成矩阵相乘形式，可得：

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2*k} \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^k y_i \end{bmatrix} \tag{4.6}$$

(6) 将式子4.6进行化简，可得：

$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^k \\ 1 & x_2 & \cdots & x_2^k \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^k \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{4.7}$$

(7) 将式子4.7写作 $XA = Y$，其中，

$$\begin{aligned} X &= \begin{bmatrix} 1 & x_1 & \cdots & x_1^k \\ 1 & x_2 & \cdots & x_2^k \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^k \end{bmatrix} \\ A &= \begin{bmatrix} a_0 & a_1 & \cdots & a_k \end{bmatrix}^T \\ Y &= \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^T \end{aligned} \tag{4.8}$$

则矩阵 A 即为系数矩阵，将其求出即可得到多项式的系数。由矩阵方程的解法可知：

$$A = (X^T X)^{-1} X^T Y \tag{4.9}$$

其中，X^T 是 X 的转置矩阵，$(X^T X)^{-1}$ 是矩阵 $X^T X$ 的逆矩阵。

由式子4.9可以看出，系数矩阵 A 的计算需要经过三次矩阵相乘运算和一次矩阵求逆运算，矩阵相乘的复杂度为 $O(n^3)$，矩阵求逆复杂度可达 $O(n^3)$，因为矩阵的数据量大，乘法和求逆的复杂度很高，所以系数的计算是一个耗时的过程。

4.2 静态时序数据压缩加速方法研究

静态时序数据的压缩，是指时序数据已经全部获得，直接在静态数据集上进行压缩。时序数据是按照时间顺序来进行采样记录的同一指标的数据记录，因为带有时间的特性而区别于普通的数据。时序数据可以分为时期数据和时间点数据，时期数据是某个时间段内所积累的值，比如手机一小时的耗电量，冬季一天的降雪量等。时间点数据是某一时刻的数值，比如每天十二点的气温值，发动机的转速等。时序数据的采样时间一般分为两种，周期时间采样和非周期时间的采样，周期时间采样时指，采样的时间间隔相同，如每小时的降雨量，某产品每天的销量等等，非周期时间的采样，采样时间间隔不再相同，如股市中股票每天价格最高时的价格，每天价格最高时的时间点是不同的，所以此时序数据的采样时间是非周期的。现实应用中，时序数据周期采集的较多，如 UCI 数据集中，时序数据有 76 条，大部分都是周期时间采集的时序数据，只有少部分数据是非周期采集的。

4.2.1 周期时间采样的时序数据

周期时间采样的时序数据，其采样间隔相同。所以任意一段连续的时序数据的时间戳，是一个等差的递增数列。在现实应用与研究中，为了简化计算，根据等差递增序列的特性，常将时序数据的时刻序号 T 记作 $1, 2, 3, ...$。在对周期时间采样的时序数据进行分段拟合时，如果是平均分段，那么每段的长度一样，每个时序数据段的时刻序号也一样，则式子4.8中的 X 是一样的，$(X^TX)^{-1}X^T$ 的结果值也是一样的。那么在第一次拟合时序数据段时，将计算得到的 $(X^TX)^{-1}X^T$ 结果缓存下来，那么之后的计算中，将不必再次进行多次矩阵计算，直接使用之前计算出来的中间结果，做一次矩阵乘法即可得到多项式系数矩阵。为了验证该方法是否能有效减少计算时间，用某长度为 5,013,811 的实际生产数据，依次以不同段长来进行平均分段，测得 100 组结果，如图4.1所示。

由图4.1可知，不论分段段长为多大，缓存第一次的计算结果的计算方式，运行总时间远远小于原来的计算方式。当时序数据数据量 N 很大时，假设分段后的段长为 n，拟合多项式的次数为 k，则在原计算方式中，计算长度为 n 的数据段拟合多项式

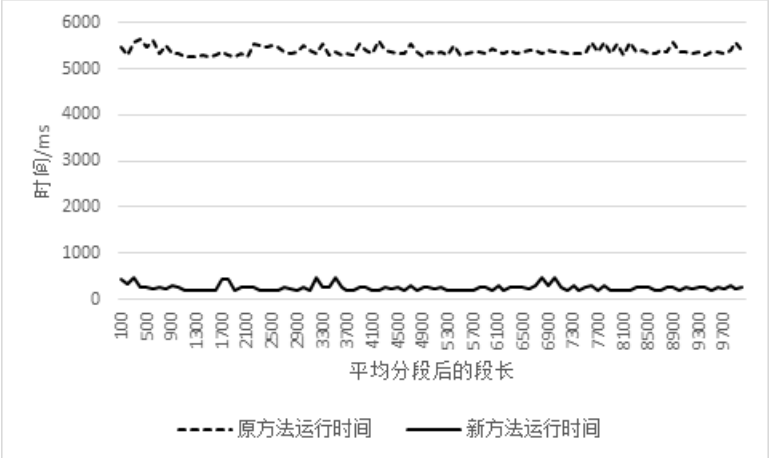


图 4.1　生产数据平均分段下两种方法的耗时情况

的系数时，$(X^TX)^{-1}X^TY$ 的计算时间为

$$\begin{aligned} T((X^TX)^{-1}X^T) &= T((X^TX)) + T(M^{-1}) + T(NX^T) + T(PY) \\ &= (k+1)^2 n + (k+1)^3 + (k+1)^2 n + (k+1) n \\ &= 2n(k+1)^2 + (k+1)^3 + n(k+1) \\ &\approx 2n(k+1)^2 + n(k+1) \end{aligned} \tag{4.10}$$

其中，矩阵 M, N, P 分别代表前面的矩阵的计算结果。因为 k 与 n 相比，是比较小的值，而且在实验过程中，发现需要进行求逆操作的矩阵规模比较小，求逆所需要的时间比较少，所以将这部分的时间忽略。段数一共有 N/n 段，则总时间为

$$\begin{aligned} T_1 &= N/n * (2n(k+1)^2 + n(k+1)) \\ &= N(2(k+1)^2 + (k+1)) \end{aligned} \tag{4.11}$$

对于将第一次的中间结果缓存下来的计算方法中，只在第一次，进行了全部的计算，之后的计算中，只进行了中间矩阵结果和 Y 矩阵的相乘计算，则其总时间为

$$\begin{aligned} T_2 &= 2n(k+1)^2 + (k+1)^3 + n(k+1) + n(k+1) * (N/n - 1) \\ &\approx N(k+1) \end{aligned} \tag{4.12}$$

由式子4.11和4.12可看出，将中间结果缓存的方法，计算复杂度更低，计算时间更短。而且，其计算时间与段长 n 没有关系，所以在图4.1中，无论段长为多少，两种方法的

运行时间基本不变。

当对周期时间采样的时序数据进行非平均分段后再进行多项式拟合时，此时的分段长度不再全部相同。此时，依然建立缓存区，缓存区存储不同长度的数据段对应的中间矩阵结果 $(X^TX)^{-1}X^T$。当计算当前段的多项式系数时，先在缓存区中进行查找当前段长对应的中间矩阵结果，若有，直接使用，否则，计算出该矩阵结果并缓存起来供下次使用。为了验证该方法的加速效果，使用实际生产数据的前 $1,000,000$ 数据进行实验，其中不平均分段算法采用自底向上算法，由不同的分段误差值可以得到不同的分段数目，实验结果如图4.2所示。从图中可以看出，在不平均分段的情况下，当分段数目很少时，原方法与新的计算方法的运行时间基本相同，没有起到加速效果，而随着分段数量的增加，新方法的运行时间相比于原方法迅速减少，分段数目越多，运行时间越少，加速效果越好，直至达到一个平稳的状态。

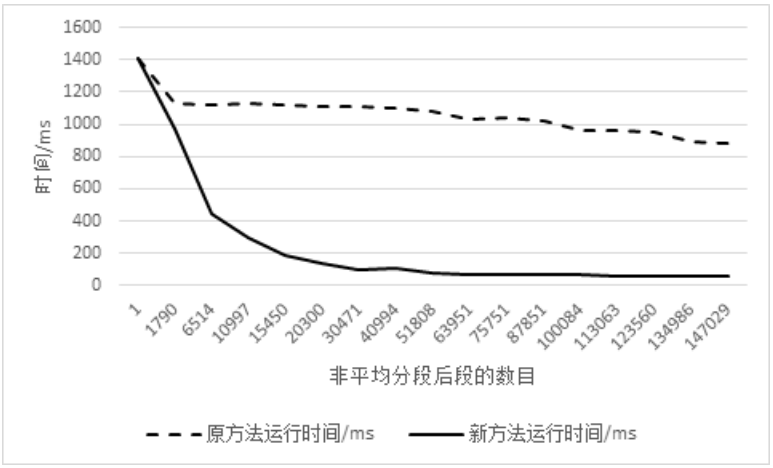


图 4.2　生产数据非平均分段下两种方法的耗时情况

4.2.2　非周期时间采样的时序数据

非周期时间采样的时序数据，其时刻序号不再是等差递增，而只是一个递增的无序序列，此时，为了简化计算，假设一段时序数据的时间戳为 $t_1, t_2, ..., t_n$，则将时刻序号中的第一项的值 x_1 记作 1，第 i 项的值 x_i 记作 $x_1 + t_i - t_1$。此时，对时序数据进行平均分段，虽然每段长度一样，但是每个时序段的时刻序号是不相同的。此时，再将第一次计算的中间矩阵结果缓存起来，下次去缓存中查找时，应该通过当前的时序数据段的时刻序号来进行查找，而不再是通过段长查找。其缓存结果如图4.4(a)所示，通过查找时刻序号 T 来查找中间矩阵结果。为了验证这种情况下的加速效果，用 UCI

数据库中一个非周期时间采集的的关于人体运动的传感器数据集[67] 来进行实验，结果如图4.3所示。由图中可知，对非周期时间采样的时序数据的压缩采用缓存的方法，虽然其加速比不如周期时间采样的时序数据的加速比，仍能起到较好加速效果，加速比在 5x～ 9x 之间。

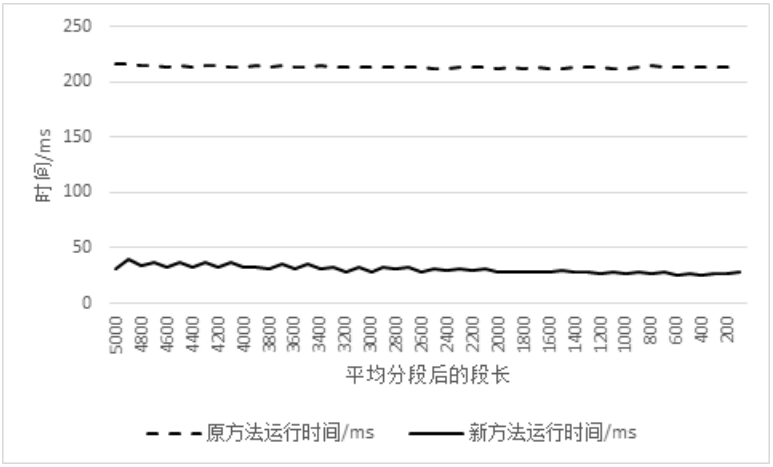
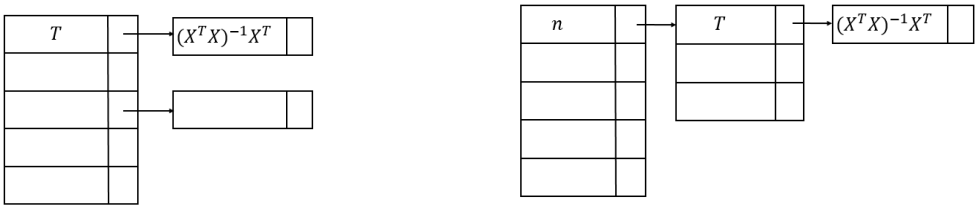


图 4.3　人体运动数据平均分段下两种方法的耗时情况

如果对非周期时间采集的时序数据进行非平均分段，其包含两个不同的因素，一是段长，二是时刻序号，相同的段长时刻序号也不一定相同。如果采用缓存的方法，缓存需要设计为二级缓存，缓存结构如图4.4(b)所示。这种情况下的加速效果不是特别理想，因此此处只给出思路。首先查找当前段长，然后再查找段长对应的时刻序号，最后得到是否已经有中间计算结果。



(a) 平均分段　　(b) 非平均分段

图 4.4　非周期时间采样的时序数据所用缓存数据结构

这一小节主要分析了静态时序数据的压缩，在已知数据整体的情况下，对于周期时间采样的时序数据，进行平均分段和不平均分段的多项式拟合时，其加速方法及其效果。对于非周期时间采样的时序数据时，进行平均分段时，其多项式拟合的加速方法及其效果，以及采用非均匀分段时，其加速想法。下一小节分析在流式计算环境下，对于流式时序数据的压缩及其加速。

4.3 流式时序数据压缩加速方法研究

本节对流式时序数据进行压缩分析。在进行分段时，不均匀分段比均匀分段更能体现数据本身的特性，有利于保留原始序列的主体特征。所以，本文选择对原始数据进行不均匀分段的多项式拟合操作进行数据的压缩。

因为流式数据的特点，压缩过程是动态的，需要进行在线的压缩，所以选择滑动窗口算法来进行分段。该过程如下。假设给定误差值为 ε，当前窗口内的已有点为 $\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$，拟合后的多项式函数为 $f_m(x)$，则必有

$$\sum_{i=1}^{m}\left(f_m\left(x_i\right)-y_i\right)^2 \leq m*\varepsilon \tag{4.13}$$

当新的数据点 (x_{m+1},y_{m+1}) 到达窗口内后，首先计算当前拟合函数 $f_m(x)$ 是否可以在误差值 ε 下拟合点 $\{(x_1,y_1),(x_2,y_2),...,(x_{m+1},y_{m+1})\}$。若

$$\sum_{i=1}^{m+1}\left(f_m\left(x_i\right)-y_i\right)^2 \leq (m+1)*\varepsilon \tag{4.14}$$

则

$$f_{m+1}(x)=f_m(x) \tag{4.15}$$

否则，需要对当前窗口内 $m+1$ 个点进行新的多项式拟合。重复此过程直至当前段已达到最大数量，再多一个数据点后拟合后的误差值大于 ε，此时，窗口内的数据点分为一个数据段，窗口重新开始接收新的数据作为新的一段。该过程表达如算法3所示。

用多项式拟合来压缩时序数据的过程，就是得到多项式系数的过程。由以上内容可知，每来一个数据点时基本都要进行一次系数矩阵的计算，而且计算过程需要进行矩阵的乘法和求逆的运算，矩阵的规模可以达到当前数据段段长的平方。当窗口内的数据量很大时，这将是一笔巨大的计算时间开销。时序数据分为周期时间采集的与非周期时间采集的数据。针对这两类不同时刻序号的时序数据，对其压缩过程的加速也是不同的。

Algorithm 3: 流式时序数据的在线多项式拟合算法

Input: Streaming time-series data
Output: the polynomial coefficient after fitting
Let k be the degree of polynomials;
Let ε be the predefined error bound;
$temp\_seg = \phi$;
while *input(x, y)* **do**
  $temp\_seg = temp\_seg \cup (x, y)$;
  if *Length(temp_seg)*$= k + 1$ **then**
    $temp\_A = calculatePolynomial(temp\_seg, k)$;
    $A = temp\_A$;
    $seg = temp\_seg$;
  else
    if *Length(temp_seg)* $> k + 1$ **then**
      if *calculateError(*$temp\_seg, temp\_A$*)* $> \varepsilon$ **then**
        $temp\_A = calculatePolynomial(temp\_seg, k)$;
        if *calculateError(*$temp\_seg$, $temp\_A$*)* $> \varepsilon$ **then**
          $save(A, seg)$;
          $temp\_seg = \{(x, y)\}$;
        else
          $A = temp\_A$;
          $seg = temp\_seg$;

4.3.1 周期时间采样的时序数据

由上节内容可知，周期时间采集的时序数据的时刻序号 T 为 $1, 2, 3...$，此时，式子4.8中的 $x_i = i$。此时，由式子4.8和4.9可知，如果两个时序数据的序列的段长相同，则其时刻序号相同，矩阵 X 是相同的，从而 $(X^TX)^{-1}X^T$ 的矩阵结果是相同的。而在对时序数据进行在线压缩时，滑动窗口内的数据是依次增加的，所以，压缩后任意不同两段的时序数据在压缩的过程中，前期长度一样的时候，其 X 是一样的，则中间矩阵计算结果 $(X^TX)^{-1}X^T$ 也是一样的。而且，假设经过压缩后有任意两段时序数据，则长度较短的一段的时刻序号必定是长度较长的一段的时刻序号的前缀。则在计算较短段长的时序数据段的过程中，其计算得到的 $(X^TX)^{-1}X^T$ 中间结果必定是计算较长段长时序数据段的过程中需要的中间结果的一部分。而矩阵的计算是十分消耗时间的，假如建立一个缓存区域，将已经计算过的矩阵结果缓存起来，在下一次计算的时候，在缓存区域查找是否已经有计算结果，如果有，可以避免进一步的计算，可以减少计算时间，进行加速。所以，对于周期时间采样的时序数据，可以通过空间换时

间的思想，重复利用矩阵的中间计算结果，来减少计算时间。

4.3.1.1 建立缓存

段长为 n 的时序数据的时刻序号 $x_1, x_2, x_3, ..., x_n$ 为 $1, 2, 3, ..., n$，则段长为 n 的时序数据进行多项式拟合，求解拟合多项式系数矩阵时，式子4.9中的 $(X^TX)^{-1}X^T$ 的结果值是一样的。所以，对于唯一的段长值 n，可以唯一确定值 $(X^TX)^{-1}X^T$。因此，如果在计算过程中，对于当前长度为 n 的时序数据段，首先去缓存查找是否有 n 对应的 $(X^TX)^{-1}X^T$ 计算结果，如果有，直接使用该结果；如果没有，计算出矩阵结果，并且将该结果存储在缓存区域，下一次即可不需计算直接使用。此时，在对流式时序数据进行多项式拟合时，对于不同段长的数据段的 $(X^TX)^{-1}X^T$ 计算只需要进行一次，而在之后的运算中，由原来三次的矩阵乘法运算和一次矩阵求逆运算减少到只需要一次矩阵乘法计算，大大地减少了计算步骤，提高了计算效率。

$(X^TX)^{-1}X^T$ 的计算结果是一个 $(k+1)*n$ 的矩阵，其中，k 是拟合多项式的次数，则 $k+1$ 是多项式的系数的个数，n 是被拟合的时序数据段的段长。假设 $k=6, n=i$，元素数据类型为 $double$ 型，每个元素占 8 字节，则一个矩阵占用空间为 $(6+1)*i*8 = 56iB$，则假设最长时序数据段段长为 N，则所占的内存空间为

$$\sum_{i=1}^{N} 56i = 28N(N+1)B \tag{4.16}$$

如果缓存空间大小为 $128MB$，则 $N \approx 2189$，所以可以缓存的矩阵计算结果的时序数据的长度可达 2189, 所以使用存储矩阵的方法来减少计算时间进行拟合加速是可行的。

4.3.1.2 缓存结构

由上节内容可知，确定的时序数据段段长 n 对应的中间结果是唯一的，所以缓存可以采用哈希表表示，数据结构如图4.5所示。

其中，拟合的时序数据的段长 n 为关键字 key，该段长对应的中间结果矩阵值 $(X^TX)^{-1}X^T$ 为 key 对应的 $value$。key 的存储以数组形式存储，所以查找速度很快，相比于计算时间几乎不消耗多余时间。

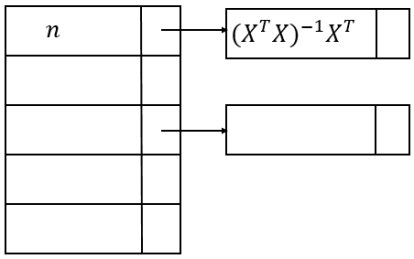


图 4.5　缓存区域的数据结构

4.3.2　非周期时间采样的时序数据

非周期时间采样的时序数据，如果建立缓存，因为段长和时刻序号都不相同，通过实验发现加速效果很差，而且还浪费缓存内存与查找时间，所以，这种情况下不再使用缓存的方法。

假设当前窗口内的数据段为 $\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$，则其多项式系数矩阵 $A=(X^TX)^{-1}X^TY$。可以得到，

$$X^TX=\begin{bmatrix} n & \sum_{i=1}^{n}x_i & \cdots & \sum_{i=1}^{n}x_i^k \\ \sum_{i=1}^{n}x_i & \sum_{i=1}^{n}x_i^2 & \cdots & \sum_{i=1}^{n}x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n}x_i^k & \sum_{i=1}^{n}x_i^{k+1} & \cdots & \sum_{i=1}^{n}x_i^{2*k} \end{bmatrix} \tag{4.17}$$

$$X^TY=\begin{bmatrix} \sum_{i=1}^{n}y_i \\ \sum_{i=1}^{n}x_iy_i \\ \vdots \\ \sum_{i=1}^{n}x_i^ky_i \end{bmatrix} \tag{4.18}$$

当窗口内新来一个数据点 (x_{n+1},y_{n+1}) 时，当前被拟合的数据段为 $\{(x_1,y_1),(x_2,y_2),...,$

$(x_n, y_n), (x_{n+1}, y_{n+1})\}$，多项式的系数矩阵 $A' = (X'^T X')^{-1} X'^T Y'$，在这个式子中，有

$$
\begin{aligned}
X'^T X' &= \begin{bmatrix} n+1 & \sum_{i=1}^{n+1} x_i & \cdots & \sum_{i=1}^{n+1} x_i^k \\ \sum_{i=1}^{n+1} x_i & \sum_{i=1}^{n+1} x_i^2 & \cdots & \sum_{i=1}^{n+1} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n+1} x_i^k & \sum_{i=1}^{n+1} x_i^{k+1} & \cdots & \sum_{i=1}^{n+1} x_i^{2*k} \end{bmatrix} \\
&= \begin{bmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2*k} \end{bmatrix} + \begin{bmatrix} 1 & x_{n+1} & \cdots & x_{n+1}^k \\ x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n+1}^k & x_{n+1}^{k+1} & \cdots & x_{n+1}^{2*k} \end{bmatrix} \\
&= X^T X + \begin{bmatrix} 1 & x_{n+1} & \cdots & x_{n+1}^k \\ x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n+1}^k & x_{n+1}^{k+1} & \cdots & x_{n+1}^{2*k} \end{bmatrix}
\end{aligned} \tag{4.19}
$$

由式子4.19可知，$X'^T X'$ 的结果由两部分组成，一部分是之前计算过的结果 $X^T X$，是不变的部分，一部分只与新来的数据点的时刻值 X_{n+1} 有关，是可变的部分。所以每次做计算时，将 $X^T X$ 的结果临时存储一下，那么进行下次拟合计算的时候，只需要计算可变部分的结果，然后将不变部分与可变部分做矩阵相加即可得到 $X'^T X'$ 的结果。这次的结果依旧可以作为临时变量存起来，作为下次计算结果的一部分。该方法采取了增量计算的思想，每一批计算的结果是由之前数据的结果与增量的数据的结果得到的，即每次新来一个数据点后，只计算该数据点带来的影响变化，并将这个变化应用到之前的结果上。这种方法可以减少一部分的数据计算，进而进行计算过程的加

速。同理，在计算 $X'^{T}Y'$ 时，有

$$
\begin{aligned}
X'^{T}Y' &= \begin{bmatrix} \sum_{i=1}^{n+1} y_i \\ \sum_{i=1}^{n+1} x_i y_i \\ \vdots \\ \sum_{i=1}^{n+1} x_i^k y_i \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^k y_i \end{bmatrix} + \begin{bmatrix} y_{n+1} \\ x_{n+1}y_{n+1} \\ \vdots \\ x_{n+1}^k y_{n+1} \end{bmatrix} \\
&= X^T Y + \begin{bmatrix} y_{n+1} \\ x_{n+1}y_{n+1} \\ \vdots \\ x_{n+1}^k y_{n+1} \end{bmatrix}
\end{aligned}
\tag{4.20}
$$

因此，$X'^{T}Y'$ 部分的计算，依旧可以采用增量计算的思想，通过将 X^TY 的结果与计算新来的数据点 (x_{n+1}, y_{y+1}) 得到的可变部分的结果相加得到。从上面这些分析可知，在流式环境下，当对非周期时间采样的时序数据进行压缩时，可以在计算过程中采用增量计算的方法减少计算量。等新来一个数据点时，上一次计算结果可以重复使用，而这次的结算结果，又可以作为下一次计算的部分结果使用。每一次只需要计算新来数据点带来的可变部分的结果即可。而且由式子可知，这种方法不需要缓存滑动窗口内的数据点为 $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$，只需要存储每次的 X^TX 与 X^TY 的结果即可。

4.4 本章小结

本章介绍了对流式时序数据进行在线压缩的过程。针对数据的在线压缩选择了滑动窗口的算法，并且给出了多项式拟合的伪代码和计算其系数的过程。针对周期时间采样的时序数据的时刻序号的特点，设计了一种缓存的方式，以空间换时间，来减少计算过程中关于矩阵的计算，减少了计算时间，提高了计算效率，保证了流式数据计

算的实时性。而对于非周期时间采样的时序数据，分析了其计算过程，提出了一种增量计算的方法，减少了数据的计算量，对拟合进行了加速。

第 5 章　实验

第三章针对流式数据的特点，提出了一种利用网格的改进的增量 LOF 快速异常数据检测算法，不但可以减少存储空间，也可以减少计算量，加快检测速度。第四章针对流式数据中时序数据的特点，分别对周期时间采样的时序数据和非周期时间采样的时序数据提出加速的计算方式，分别建立缓存和使用增量计算，减少矩阵的计算次数，实现快速压缩。为了验证所提算法的有效性，本章设计了多组对照实验，进行了多方面的性能比较。

5.1　实验环境

所有的实验的运行环境为 Windows 7，64 位操作系统，CPU 为 3.40GHz，安装内存为 8.00GB。使用 Java 语言在 Eclipse 上编程实现，并且用 Python 实现结果的可视化。

5.2　异常数据检测对比实验

在本节内容中, 进行异常数据检测的对比实验。测试实验采用的数据集一共有两种，一种是真实数据集 KDD CUP-99[68], 这个数据集是由 MIT 林肯实验室收集的网络入侵检测数据集。另外，为了可以更好地将算法运行的结果可视化，另一种测试的数据集是人工数据集，称其为 RandomSet。

5.2.1　人工数据集的对比实验

人工数据集 RandomSet 共有 11,000 条数据记录，是一个二维数据集，每一维数据类型都为实数，第一维数据的范围在 0 ～ 520，第二维数据范围为 0 ～ 260。RandomSet 的分布如图5.1所示，数据共可以聚为 4 类，其中两类数据，分布比较均匀，但是数据量较少，分布比较稀疏。另外两类，中心密度较高，周围数据点分布比较松散。在整个数据集中，有些游离在类外的数据点。该数据集主要用来进行增量的 LOF 算法与本算法的对比实验，包括异常点检测的精确度对比，运行时间对比和运行内存对比。

数据集 RandomSet 是随机生成的，由图5.1可以看出，虽然四个类的密度不同，但是根据异常数据的定义，异常数据点主要是在类外面的游离的数据点和分布在圆形数

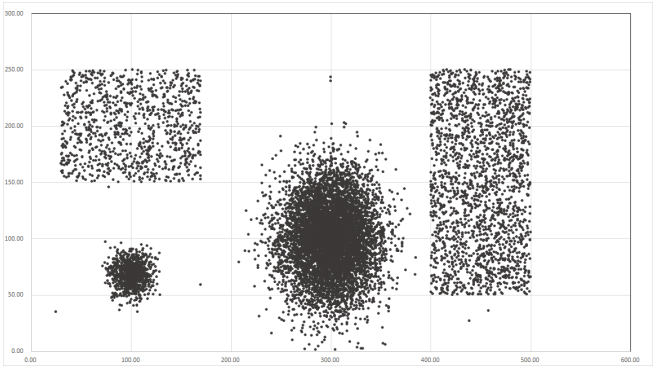
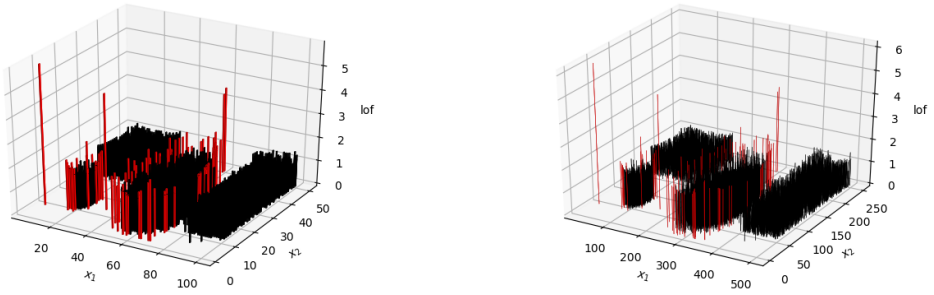


图 5.1　人工数据集 RandomSet

据类的外侧边缘的数据点。在实验中，各类的数据点以及异常数据点交替出现，特别是异常数据点，随机地出现在数据流中，次序是随机的。在实验中，k 取值为 180。每个维度平均分段，段的长度 len 取值为 5。则本算法的异常数据检测的结果与增量的 LOF 算法的结果如图5.2所示。其中 X_1 表示第一维，X_2 代表第二维，LOF 代表检测出来的因子。LOF 越大，是异常值的可能性就越大。

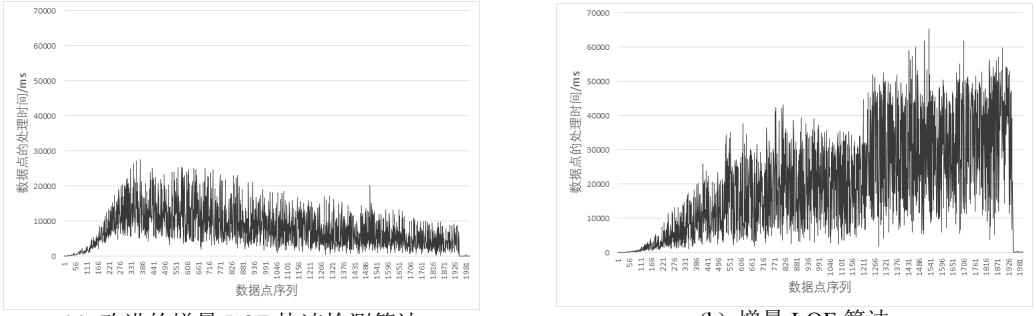


(a) 改进的增量 LOF 快速检测算法　　(b) 增量 LOF 算法

图 5.2　在 RandomSet 上两种方法的异常数据检测结果

其中，图5.2(a)是本文提出的算法的检测结果，图5.2(b)是增量的 LOF 算法的检测结果。由两图对比可知，本算法检测出来的异常网格与增量的 LOF 算法检测出来的异常数据的局部异常因子基本一致，也符合图5.1的直观感受到的异常数据。

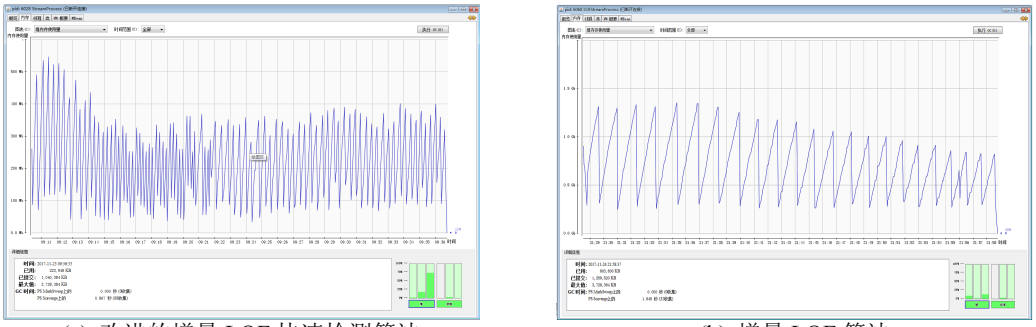
为了验证本文提出算法的加速效果，选择了连续的 2000 个数据点，记录每个数据点到来后进行检测的时间，其结果如图5.3所示。由5.3的结果可知，当数据量较少时，两种算法的运行时间基本相同，但随着数据量的增长，增量 LOF 算法的对每个数据点的检测时间大致在增长，而本算法对每个数据点的检测时间基本不变，并且小于增量 LOF 算法，所以当数据量越来越多时，本算法对每个数据点的检测时间更短，



(a) 改进的增量 LOF 快速检测算法　　(b) 增量 LOF 算法

图 5.3　在 RandomSet 上两种方法的运行时间对比

速度更快。实验的运行时内存对比如图5.4所示。在实验中，两种算法的实现程序在 Eclipse 中运行，运行的 JVM 内存参数设置为“$-Xmx4096m\ -Xms1024m$”，使用工具 JConsole 来查看运行时内存变化，其结果如图5.4所示，从图中可以看出，本算法的运行过程中，大部分时间的内存使用量在 400MB 以下，而增量的 LOF 算法的运行中，内存的使用量可达到 1.3GB，大部分可达 1.0GB 以上。所以，本算法的运行过程中，内存使用量更少。



(a) 改进的增量 LOF 快速检测算法　　(b) 增量 LOF 算法

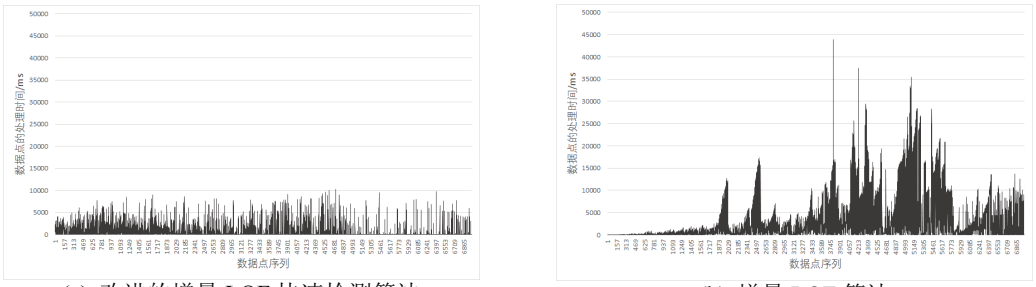
图 5.4　在 RandomSet 上两种方法的运行内存变化

5.2.2　KDD CUP-99 数据集的对比实验

KDD CUP-99 数据集共有 41 个属性，其中，有 34 个属性为连续属性，有 7 个为离散属性，在实验中，采用 34 个连续的属性，并且，为了数据处理的准确性，防止属性值很大的维度占据主要的比重而忽略属性值小的维度，对数据集的各个维度做 0-1 标准化处理，转换函数如下：

$$x' = \frac{x - min}{max - min} \tag{5.1}$$

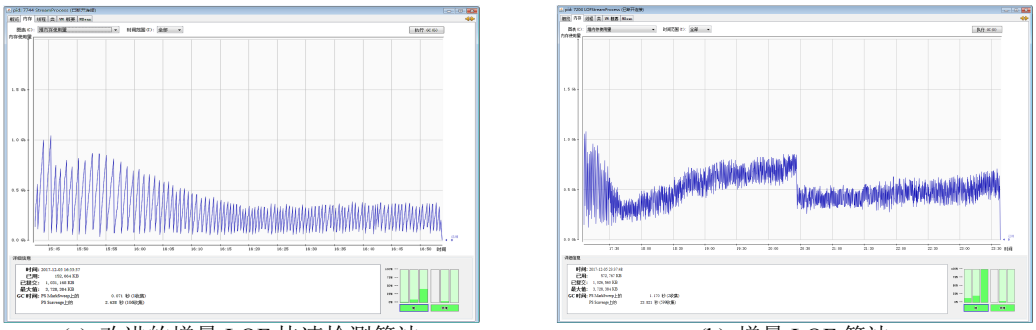
处理后的各个维度的值均在 $[0,1]$ 之间。在实验中，各个维度的划分长度为 0.05，k 的取值为 180。该实验的实验数据集是真实数据，主要进行性能测试的比较，一共选取了连续的 10,000 条记录来进行实验，并在其中，选择了连续的 7,000 条数据记录，记录了每条数据的处理时间，对比结果如图5.5所示。



(a) 改进的增量 LOF 快速检测算法　　(b) 增量 LOF 算法

图 5.5　在 KDD CUP-99 数据集上两种方法的异常数据检测结果

从图5.5中可以看出，在开始的一段时间内，原来的增量 LOF 算法运行速度要更快，但是一段时间后，随着数据量的增加，本文提出的改进的增量 LOF 快速检测算法的处理速度更快，时间更少。而且从全局时间来看，本文提出的改进的增量 LOF 快速检测算法的对每个数据的处理时间维持在一个较为稳定的水平，时间波动不大，而原增量 LOF 算法的运行之间呈现增长趋势，这也与本文前面的分析保持一致。两种算法的程序也在 Eclipse 中运行，运行的 JVM 内存参数设置为“$-Xmx4096m\ -Xms1024m$”，使用工具 JConsole 来查看运行时内存变化，其结果如图5.6所示。在图5.6(b)和5.6(a)中，



(a) 改进的增量 LOF 快速检测算法　　(b) 增量 LOF 算法

图 5.6　在 KDD CUP-99 数据集上两种方法的运行内存变化

横坐标为运行的时间，纵坐标为内存的使用量。从图中可以看出，增量的 LOF 算法的运行时间长于本文提出的改进算法，在运行前期，两种算法的内存使用量基本一致，但是随着时间的增长，数据量越来越多，本文提出的算法的内存使用量保持在 0.5MB

以下，呈现稳定的状态，而增量的 LOF 算法的内存使用量越来越高，并且明显高于 0.5MB。所以从对于 KDD CUP-99 数据集的实验中可以看出，在流式数据环境中，当数据量越来越多的时候，本算法的运行时间更少，且在运行过程中，内存使用量更少。

5.3 压缩算法对比实验

在本节内容中，进行流式时序数据的多项式压缩的对比实验。通过以下几组对比实验，来验证本文在上一章提出的加速方法的效果，是否可以很好地提高计算速度，有效地减少多项式拟合时间。实验采用三个时序数据集，其中两个数据集是周期时间采样的数据集，包括某实际生产数据集，从 UCI 数据库中下载得到的气体数据集[69]。剩余一个是非周期时间采样的数据集，从 UCI 数据库中下载，是测量人体移动的传感器数据集[67]。本节实验将用这些时序数据集模拟流式时序数据。

实验分别验证周期时间采样的时序数据与非周期时间采样的时序数据的加速方法的加速效果，在实验中，定义加速比为原计算方法的运行时间与加速方法的运行时间的比值。为了得到较为精确的对比，分别选取不同的拟合误差 (ε) 与不同的多项式次数 (k) 来进行实验对比。

5.3.1 周期时间采集的流式时序数据对比实验

周期时间采集的时序数据包括实际生产数据集和从 UCI 数据库中下载的气体数据集[69]，其中，实际生产数据集包含 $5,013,811$ 条数据记录，UCI 气体数据包含 $4,178,504$ 条数据记录。

当测试数据集是实际生产数据集，多项式的次数 k 依次为 $5,6,7,8$，并且拟合误差 ε 分别为 $100,125,...,250$ 时，其运行结果如图5.7所示。

由图中可知，当拟合多项式次数 k 越大时，两种方法的运行时间越长。而且，当 k 不变时，随着拟合误差 ε 的增大，运行时间先变大，之后基本保持不变，这是因为拟合误差越大，每个分段后的数据段长度越长，所以每段的拟合时间会变长，但是因为因为数据量总量不变，所以段长变长后，段数会变少，所以，全体数据的拟合时间基本不变。在生产数据集上多项式拟合的加速比如图5.8所示。

由图5.8可知，本文提出的加速方法与原计算方法相比，具有较好的加速效果，加速比最大可达 2.4x，而且整体来看，多项式次数 k 越大，加速效果越好。关于 UCI 气

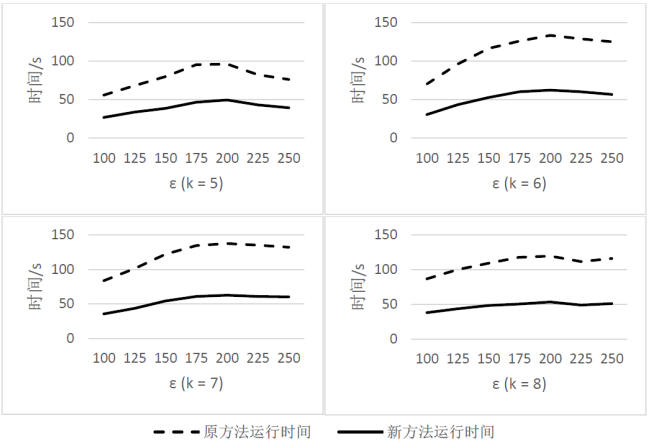


图 5.7　周期时间采样的生产数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差

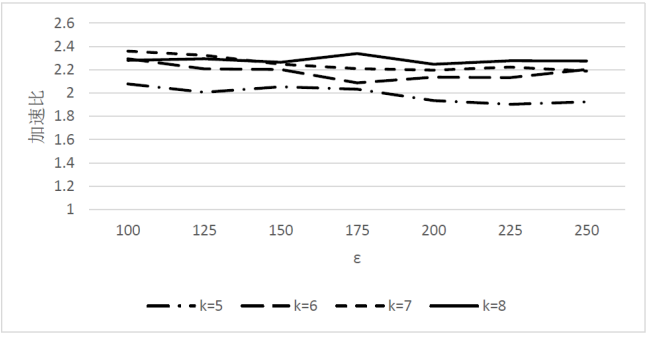


图 5.8　周期时间采样的生产数据上的加速比，其中 k 是多项式次数，ε 是拟合误差

体数据集，将多项式次数 k 设置为 $5, 6, 7, 8$，因为其数据属性原因，所以将其拟合误差 ε 设为 $5, 6, 7, .., 10$，则运行时间结果如图5.9所示，加速比如图5.10所示。

由图5.9可知，本文提出的加速方法的运行时间少于原方法的运行时间，当 k 和 ε 变化时，总运行时间基本不变。由图5.10可知，加速比在 2x $\sim$ 2.5x 之间，而且 k 越大，加速效果越好。由上面两组实验结果可知，在流式环境中，本文提出的方法计算速度更快，具有较好的加速效果，而且对于周期性采样的流式时序数据，加速比可达到 $2x$。

5.3.2　非周期时间采样的流式时序数据对比实验

在关于非周期时间采样的流式时序数据的对比实验中，采用从 UCI 数据库中下载的非周期时间采集的测量人体移动的传感器数据集[67]，这个数据集有 $195,737$ 条数据记录。因为非周期时间采集的时序数据集较少，所以为了得到更加精确的结论，将周期时间采集的 UCI 气体数据集[69] 当做非周期时间采集的数据集，来进行此组实验。

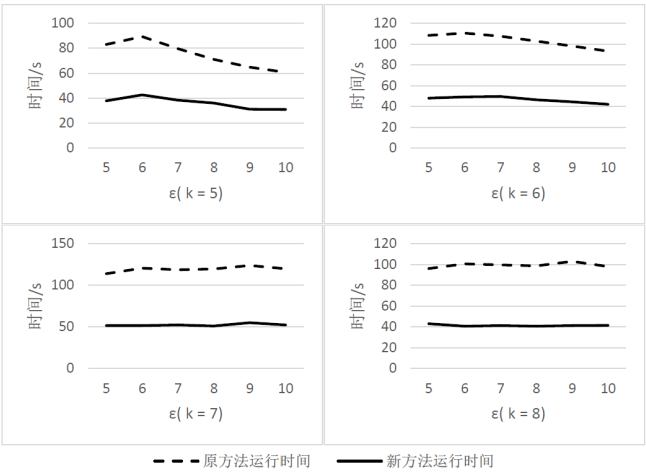


图 5.9　周期时间采样的 UCI 气体数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差

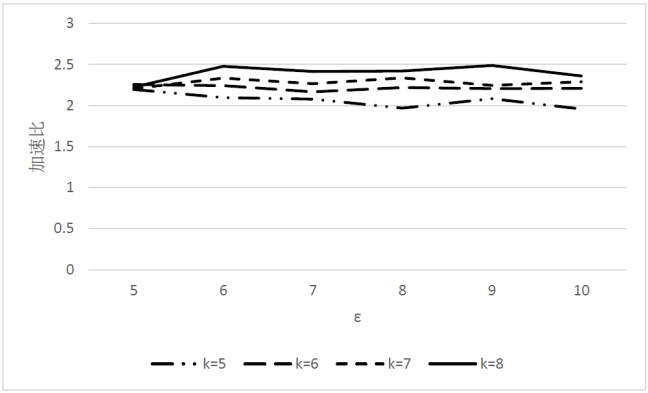


图 5.10　周期时间采样的 UCI 气体数据上的加速比，其中 k 是多项式次数，ε 是拟合误差

所以，本组实验的实验数据包括 UCI 人体移动传感器数据集和 UCI 气体数据集。

如图5.11所示，是在 UCI 气体数据集上的运行结果，由图中可知，当 k 不变时，拟合误差 ε 越大，运行时间呈大致先增大后减小的趋势。而当拟合误差 ε 不变时，k 越大，运行时间越长。所以运行时间随着多项式次数 k 的增大而增大。而且当 k 与 ε 相同时，本文提出的加速方法，运行时间要少于原方法。其加速比如图5.12所示。

由图可知，其加速比在 1.4x $\sim$ 2.2x 之间，最大可达 2.2x，而且 k 越大，加速比越大，但其变化不大，甚至 $k = 8$ 时的加速比要整体小于 $k = 7$ 时的加速比。

在关于人体移动传感器数据集的实验中，因为采集时间间隔是不确定的，时刻序号是不确定的，而且相对来说某些时间间隔很大，即计算中 x_i 的值很大。所以，在计算过程中，会遇到矩阵 X 中的元素很大，以至于 X^TX 中的元素值很大，而对其求逆后，逆矩阵 $(X^TX)^{-1}$ 中的元素值很小，接近于 0，最终会出现逆矩阵为零矩阵的问题。为了解决这个问题，实验中对时刻序号进行处理，全部将其缩小十倍。最后，实

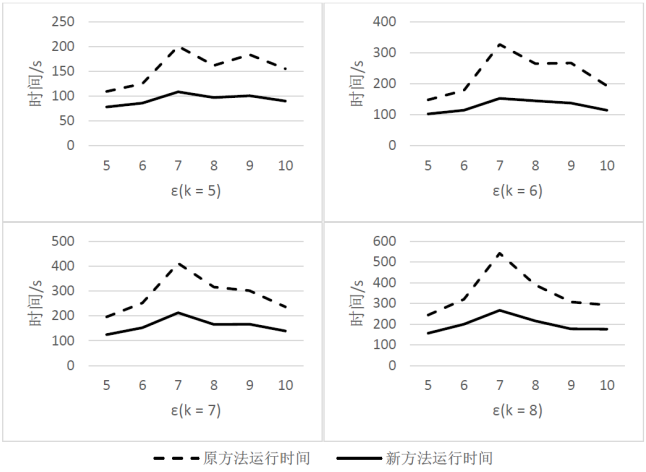


图 5.11　模拟非周期时间采样的 UCI 气体数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差

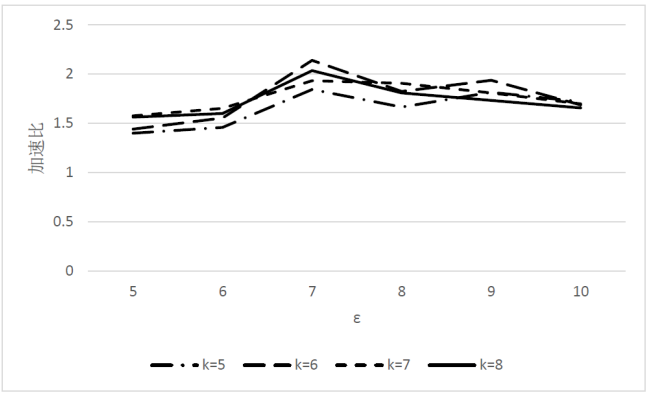


图 5.12　模拟非周期时间采样的 UCI 气体数据上的加速比，其中 k 是多项式次数，ε 是拟合误差

验的运行时间结果如图5.13所示，其加速比如图5.14所示。

由图5.13可知，当多项式次数 k 不变时，运行时间基本不变，但是当 k 增大时，运行时间都会增大。由图5.14可以看出，加速比基本在 1.4x ～ 1.7x 之间。从这两组实验数据可知，无论多项式次数 k 和拟合误差 ε 如何取值。在非周期时间采样时序数据上的加速方法可以达到加速效果。

5.4　本章小结

本章内容对本文提出的流式计算中的快速异常数据检测算法与数据压缩的加速方法进行了实验验证。在异常数据检测的实验中，分别选择了数据集，进行了增量的 LOF 算法与本文提出的加速算法的对比实验，实验结果发现，本算法可以较快地发现异常数据，而且使用内存容量更少，更能符合流式数据数据量大以及处理速度快的特

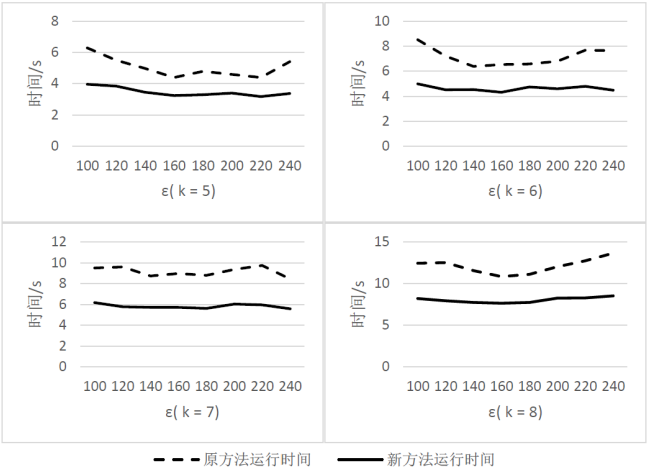


图 5.13　非周期时间采样的 UCI 人体移动传感器数据上的运行时间，其中 k 是多项式次数，ε 是拟合误差

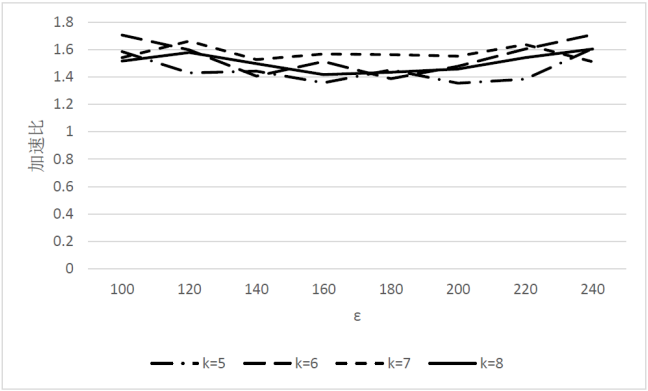


图 5.14　非周期时间采样的 UCI 人体移动传感器数据集上的加速比，其中 k 是多项式次数，ε 是拟合误差

点。在对流式时序数据的压缩实验中，由实验结果可知，本文提出的方法，能够在周期时间采样的时序数据的压缩上达到很好的加速效果。而对于非周期时间采样的时序数据，虽然加速效果不如周期时间采样的时序数据，但是也可以达到 1.5x 倍的加速效果。

结论

随着大数据时代的到来，信息技术与互联网的快速发展，数据量呈爆炸式增长。数据中包含着巨大的价值，而如何从海量的数据中挖掘出对人类有价值的信息，是现代社会的一个巨大的挑战与机遇。数据的压缩与异常数据检测，是在数据的分析与挖掘模型中重要的两个环节，在工业与生活中，也具有广泛的应用。流式数据是大数据中重要的一种形式。在对流式数据的处理与分析中，存在着以下几个问题。首先，是数据的存储问题，流式数据的数据量巨大，已有的存储介质不能存储所有的原始数据，所以，一些适用于静态数据集的挖掘方法无法直接使用在流式数据上。然后，是计算的实时性问题，流式数据是快速而且持续地产生的，计算的实时性尤其重要，如果数据处理不及时，很有可能发生数据的堆积，从而造成数据丢失。最后，流式数据往往是动态变化的，数据的分布总在变化，这使得很多静态数据集上研究的算法无法适应，效率与性能大大降低。上述的这些问题，给流式数据的研究带来了困难与挑战。基于上述问题，本文主要对流式数据的数据压缩与异常数据检测的加速进行了研究，使其能够快速计算，保证流式数据处理的实时性。本文的主要研究成果可以概括如下：

一，对数据的异常数据检测与数据的压缩进行了研究与总结。总结介绍了异常数据的定义，异常数据是不符合正常行为模式定义的数据模式，异常数据分为点异常，上下文异常和集体异常。而且异常数据检测的输出有分数型和标记型两种形式。然后又介绍了异常数据检测的相关算法，分析了各种算法的优缺点与使用范围。介绍了数据压缩的意义和现有的数据压缩算法，包括分段表示法、频域法、奇异值分解法与符号表示法。分析了各种算法的优缺点与使用范围，并且阐述了分段表示法最常使用的原因。

二，在对流式数据进行异常数据检测时，针对流式数据数据量巨大的特点，提出了改进的增量 LOF 算法。第一，将其空间划分为多个网格，将流式数据的数据点映射到网格中，可以解决流式数据数据量大，无法全部存储的问题。第二，设计网格的特征向量，将网格中的有权值的中心点代替映射到网格中的所有数据点，来进行增量的 LOF 算法的检测，可以减少计算量，加速检测异常数据速度，保证流式计算的实时性。第三，实验表明，该算法不但可以有效检测出异常数据，而且检测异常数据的速度更快，效率更高。

三，在对流式数据进行压缩时，选择简单直观常用的分段多项式拟合算法，提出

了加速算法。第一，针对分段多项式拟合，给出了最小二乘法的解决过程。第二，针对静态时序数据的压缩，分别给出了平均分段与不平均分段的加速方法，通过建立缓存，来直接使用之前的计算结果，减少矩阵计算，加快了计算的速度。并且分别针对周期时间采集的与非周期时间采集的时序数据，给出了不同的加速方式。第三，针对流式时序数据，给出了使用滑动窗口算法的压缩过程，而且对于周期时间采样的时序数据，根据其时刻序号的特点，给出了使用缓存减少计算量的方法，加速压缩过程。针对非周期时间采样的时序数据，因为采样间隔不确定，所以不再使用空间替换时间的方法，而提出一种增量计算的方法，减少计算量和窗口内的数据点的存储量，提高了计算效率。

参考文献

[1] Lynch C. Big data: How do your data grow? [J]. Nature, 2008, 455 (7209): 28.

[2] Li G, Cheng X. Research status and scientific thinking of big data [J]. Bulletin of Chinese Academy of Sciences, 2012.

[3] Wang Y Z. Network Big Data:Present and Future [J]. Chinese Journal of Computers, 2013, 36 (6): 1125–1138.

[4] James Manyika M C. Big Data: The Next Frontier for Innovation, Comptetition, and Productivity. https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation.

[5] White T, Cutting D. Hadoop : the definitive guide [J]. O'reilly Media Inc Gravenstein Highway North, 2009, 215 (11): 1 – 4.

[6] 李圣, 黄永忠, 陈海勇. 大数据流式计算系统研究综述 [J]. 信息工程大学学报, 2016, 17 (1): 88–92.

[7] Apache Kafka, A high-throughput distributed messaging system. http://kafka.apache.org/design.html.

[8] Auradkar A, Botev C, Das S, et al. Data Infrastructure at LinkedIn [C]. In IEEE International Conference on Data Engineering, 2012: 1370–1381.

[9] Chauhan J, Chowdhury S A, Makaroff D. Performance Evaluation of Yahoo! S4: A First Look [C]. In Seventh International Conference on P2p, Parallel, Grid, Cloud and Internet Computing, 2012: 58–65.

[10] Xhafa F, Naranjo V, Caballe S. Processing and Analytics of Big Data Streams with Yahoo!S4 [C]. In IEEE International Conference on Advanced Information NETWORKING and Applications, 2015: 263–270.

[11] Neumeyer L, Robbins B, Nair A, et al. S4: Distributed Stream Computing Platform [C]. In IEEE International Conference on Data Mining Workshops, 2011: 170–177.

[12] Patrizio A. StreamBase's Real-Time Database Gets More Real [J]. Software Development Times, 2006 (May).

[13] Abadi D J, Ahmad Y, Balazinska M, et al. The Design of the Borealis Stream Processing Engine [J]. Cidr, 2005: 277–289.

[14] 孙玉芬, 卢炎生. 流数据挖掘综述 [J]. 计算机科学, 2007, 34 (1): 1–5.

[15] 于戈，周傲英，张天成，秦首科. 数据流分析与挖掘技术现状与发展 [J]. 中国计算机学会通讯,

2006, 7.

[16] Croux C, Garciaescudero L A, Gordaliza A, et al. Robust principal component analysis based on trimming around affine subspaces [J]. Statistica Sinica, 2018.

[17] Chandola V, Banerjee A, Kumar V. Anomaly detection:A survey [J]. Acm Computing Surveys, 2009, 41 (3): 1–58.

[18] Song X, Wu M, Jermaine C, et al. Conditional anomaly detection [J]. IEEE Transactions on Knowledge and Data Engineering, 2007, 19 (5): 631–645.

[19] Zheng Y, H, Yu Y. Detecting collective anomalies from multiple spatio-temporal datasets across different domains [C]. In Sigspatial International Conference on Advances in Geographic Information Systems, 2015: 2.

[20] Görnitz N, Kloft M, Rieck K, et al. Toward Supervised Anomaly Detection [J]. Journal of Artificial Intelligence Research, 2013, 45 (1): 235–262.

[21] Gonbadi A M, Tabatabaei S H, Carranza E J M. Supervised geochemical anomaly detection by pattern recognition [J]. Journal of Geochemical Exploration, 2015, 157: 81–91.

[22] Thakran Y, Toshniwal D. Unsupervised outlier detection in streaming data using weighted clustering [J]. World Academy of Science Engineering and Technology, 2012: 947–952.

[23] Bremer R. Outliers In Statistical Data [J]. Technometrics, 1995, 37 (1): 117–118.

[24] Ozkan H, Ozkan F, Kozat S S. Online Anomaly Detection Under Markov Statistics With Controllable Type-I Error [J]. IEEE Transactions on Signal Processing, 2016, 64 (6): 1435–1445.

[25] Rousseeuw P J, Hubert M. Anomaly detection by robust statistics [J]. Wiley Interdisciplinary Reviews Data Mining and Knowledge Discovery, 2017.

[26] Grubbs F. Procedures for Detecting Outlying Observations in Samples [J]. Technometrics, 1969, 11 (1): 1–21.

[27] Piepel G F. Robust Regression and Outlier Detection [J], 1987, 31 (2): 260–261.

[28] Barnett V, Lewis T, Abeles F. Outliers in Statistical Data [M]. Wiley, 1978.

[29] 金伟. 基于统计方法的异常数据检测及其修复 [D]. [S. l.]: 南京邮电大学, 2016.

[30] Rajasegarar S, Leckie C, Palaniswami M. Hyperspherical cluster based distributed anomaly detection in wireless sensor networks [J]. Journal of Parallel and Distributed Computing, 2014, 74 (1): 1833–1847.

[31] Jain A K, Murty M N, Flynn P J. Data clustering: a review [J]. ACM Computing Surveys, 1999, 31 (3): 264–323.

[32] Zimmer V, Valles A. Cluster anomaly detection using function interposition [J], 2017.

[33] Knorr E M, Ng R T. Algorithms for Mining Distance-Based Outliers in Large Datasets [C]. In International Conference on Very Large Data Bases, 1998: 392–403.

[34] Knorr E M, Ng R T. Finding Intensional Knowledge of Distance-Based Outliers [J]. Vldb, 1999: 211–222.

[35] Bentley J L. Multidimensional binary search trees used for associative searching [M]. ACM, 1975.

[36] Wald I, Havran V. On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N) [C]. In Interactive Ray Tracing 2006, IEEE Symposium on, 2006: 61–69.

[37] Guttman A. R-trees: A dynamic index structure for sparial searching [J]. Acm Sigmod Record, 1984, 14 (2): 47–57.

[38] Hadjieleftheriou M, Manolopoulos Y, Theodoridis Y, et al. R-Trees: A Dynamic Index Structure for Spatial Searching [J], 2016.

[39] Zhang Y, Du B, Zhang L, et al. A Low-Rank and Sparse Matrix Decomposition-Based Mahalanobis Distance Method for Hyperspectral Anomaly Detection [J]. IEEE Transactions on Geoscience and Remote Sensing, 2016, 54 (3): 1376–1389.

[40] 王斌. 面向不确定感知数据的异常数据检测技术 [D]. [S. l.]: 东北大学, 2009.

[41] Zhao C, Wang X, Yan Y. Density background refinement-based anomaly detection algorithm for hyperspectral images [J]. Journal of Harbin Engineering University, 2016.

[42] Liu F T, Kai M T, Zhou Z H. Isolation Forest [C]. In Eighth IEEE International Conference on Data Mining, 2009: 413–422.

[43] Liu F T, Kai M T, Zhou Z H. Isolation-Based Anomaly Detection [M]. ACM, 2012.

[44] Iijima M, Takagi K, Takeuchi R, et al. A piecewise-linear regression on the ASHRAE time-series data [J], 1994.

[45] Wang J, Yuan H, Wu Q, et al. A piecewise linear representation based on compression ratio [C]. In Prognostics and System Health Management Conference, 2016: 1–5.

[46] 张海涛, 李志华, 孙雅, et al. 时间序列的层次分段及相似性度量 [J]. 计算机工程与应用, 2015, 51 (10): 147–151.

[47] 戴爱明, 高学东. 时间序列三角极值点线性分段算法 [J]. 南昌航空大学学报 (自然科学版), 2009, 23 (2): 25–28.

[48] 刘贺红, 张毅坤. 确定时间序列分段点的方法研究 [J]. 计算机工程与应用, 2010, 46 (13): 44–46.

[49] Zhu Y, Wu D, Li S. A Piecewise Linear Representation Method of Time Series Based on Feature Points [C]. In International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, 2007: 1066–1072.

[50] Grinsted A, Moore J C, Jevrejeva S. Application of the cross wavelet transform and wavelet coherence to geophysical time series [J]. Nonlinear Processes in Geophysics, 2004, 11 (5/6): 561–566.

[51] Cooley J W, Lewis P A, Welch P D. The fast fourier transform and its application to time series analysis [J]. Statistical Methods for Digital Computers, 1977.

[52] Zhang L Q. An Infrared Image Background Removing Dynamic Compression Algorithm Based on Frequency Domain [J]. Optics and Optoelectronic Technology, 2016.

[53] Dwivedi Y, Subba Rao S. A test for second□order stationarity of a time series based on the discrete Fourier transform [J]. Social Science Electronic Publishing, 2015, 32 (1): 68–91.

[54] Zhang J, Tsui F, Wagner M M, et al. Detection of outbreaks from time series data using wavelet transform. [J]. AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium, 2003, 2003: 748–752.

[55] Laisné A, Gillard R, Citerne J, et al. A discrete wavelet transform (DWT)□based compression technique for the computation of Kirchhoff integrals in MR/FDTD [J]. Microwave and Optical Technology Letters, 2015, 27 (5): 312–316.

[56] Lin J, Keogh E, Li W, et al. Experiencing SAX: a novel symbolic representation of time series [J]. Data Mining and Knowledge Discovery, 2007, 15 (2): 107–144.

[57] Cadzow J A, Baseghi B, Hsu T. Singular-value decomposition approach to time series modelling [J]. Communications Radar and Signal Processing Iee Proceedings F, 1983, 130 (3): 202–210.

[58] He H, Chen S, Li K, et al. Incremental Learning From Stream Data [J]. IEEE Transactions on Neural Networks, 2011, 22 (12): 1901–14.

[59] Bifet A, Holmes G, Pfahringer B. Improving Adaptive Bagging Methods for Evolving Data Streams [C]. In Asian Conference on Machine Learning Advances in Machine Learning, 2009: 23–37.

[60] Ando S, Thanomphongphan T, Seki Y, et al. Ensemble anomaly detection from multi-resolution trajectory features [J]. Data Mining and Knowledge Discovery, 2015, 29 (1): 39–83.

[61] 屠莉. 流数据的频繁项挖掘及聚类的关键技术研究 [D]. [S. l.]: 南京航空航天大学, 2009.

[62] Breunig M M. LOF: identifying density-based local outliers [J]. Acm Sigmod Record, 2000, 29 (2): 93–104.

[63] Pokrajac D, Lazarevic A, Latecki L J. Incremental Local Outlier Detection for Data Streams [C]. In Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on, 2007: 504–515.

[64] Domingos P, Hulten G. A general framework for mining massive data streams [J]. Journal of Com-

putational and Graphical Statistics, 2003, 12 (4): 945–949.

[65] Roussopoulos N, Kelley S, Vincent F. Nearest Neighbor Queries [C]. In ACM SIGMOD International Conference on Management of Data, 1995: 71–79.

[66] Achtert E, Kunath P, Pryakhin A, et al. Efficient reverse k-nearest neighbor search in arbitrary metric spaces [C]. In ACM SIGMOD International Conference on Management of Data, 2006: 515–526.

[67] Daphnet Freezing of Gait Data Set. https://archive.ics.uci.edu/ml/datasets/Daphnet+Freezing+of+Gait#.

[68] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[69] Gas sensor array under dynamic gas mixtures Data Set. https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+under+dynamic+gas+mixtures.

攻读学位期间发表论文与研究成果清单

[1] 计卫星，张露露. 一种时序数据多项式拟合加速方法 [J]. 北京理工大学学报.（EI）

致谢

马上就要毕业了，数十年的求学生涯，也即将画上句号。想到这里，不免思绪万千。

回想短短两年半的研究生生活，我的导师，计卫星老师，给了我很大的帮助，从一开始对研究领域的一无所知，到定下研究方向，再到完成论文的期间，计老师与我一起讨论了无数次，给了我非常多的指导与启发。计老师是一位学风严谨，认真负责的老师。每周，计老师都会安排组会，组会上，同学们会分享自己近期研究的内容，计老师会指导每一位同学的学习进度。如果同学有问题找计老师，计老师一定会很认真负责地跟同学一起讨论，来解决问题。在计老师的帮助下，我才能顺利度过研究生生涯，顺利完成这篇论文，在这里，对我的导师计卫星老师表示衷心的感谢与深深的敬意。

然后，我要感谢实验室的各位老师，包括石峰老师，高玉金老师，王一拙老师和魏晋老师，各位老师在我开题答辩的时候，给了我很好的意见与建议，使我能在写论文初期，意识到论文的一些问题，及时改正。要感谢实验室的石剑君师姐，张晶晶同学，廖心怡同学，高志伟同学，罗辉师弟，田泽明师弟，高建花师妹，杨恬师弟，李安民师弟与谈兆年师弟，以及我的舍友们，我们一起去打球，一起出去游玩，一起去吃大餐，一起度过了很多快乐的时光。感谢来了暖气的实验室，让我可以好好写论文。

最后，我要特别感谢我的父母，我的家人，他们虽然不懂我的学业上的问题，但是很多很难熬的时候，我的父母都在我的身边关心我，开导我，帮我度过难关，是我坚强的后盾，是我的港湾。我要感谢我的朋友们，特别是一起在北京的高中同学们，我们一起度过了很多快乐的时光，他们给了我很多很好的建议，给了我很多鼓励。