一种基于网格方法的高维数据流子空间聚类算法*)

孙玉芬　卢炎生
(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　基于对网格聚类方法的分析，结合由底向上的网格方法和自顶向下的网格方法，设计了一个能在线处理高维数据流的子空间聚类算法。通过利用由底向上网格方法对数据的压缩能力和自顶向下网格方法处理高维数据的能力，算法能基于对数据流的一次扫描，快速识别数据中位于不同子空间内的簇。理论分析以及在多个数据集上的实验表明算法具有较高的计算精度与计算效率。

关键词　网格，子空间聚类，数据流，高维数据

A Grid-based Subspace Clustering Algorithm for High-dimensional Data Streams

SUN Yu-Fen　LU Yan-Sheng
(Computer Department of Huazhong University of Science and Technology, Wuhan 430074)

Abstract　Based on the analysis of grid-based clustering algorithms, we propose a subspace clustering algorithm that can find clusters in different subspaces for high-dimensional data streams. The algorithm combines the advantages of bottom-up grid-based method and top-down grid-based method. A uniformly partitioned grid data structure is used to summarize the data stream online. A top-down grid partition method is used o find the subspaces in which clusters locate. Theory analysis and performance study with real datasets and synthetic dataset demonstrate the efficiency and effectiveness of our proposed algorithm.

Keywords　Grid, Subspace clustering, Data stream, High-dimensional data

1　概述

目前，大量应用如网络监控、通讯数据管理、证券市场分析、传感器网络等每天都在产生着大量的数据流数据。数据流是一个以一定速度连续到达的数据项序列 $x_1,\cdots,x_i,\cdots,x_n,\cdots$，这个数据项序列只能按下标 i 的递增顺序读取一次[1]。数据流的体积潜在无限，处理系统很难或无法存储整个数据流[2]。而且数据流通常要求在线处理，这使得流数据处理算法无法进行代价昂贵的磁盘存取，因此算法只能使用有限的内存对流数据进行分析处理。

聚类是一项被广泛用来探索数据内部结构的技术。它将一个数据集划分为多个簇，使得同一簇内的数据对象之间具有较高的相似度，而属于不同簇的对象之间相似度低。其中，两个数据对象之间的相似度通常由它们之间的距离来度量[3]。最近，聚类分析已被应用到数据流领域[2,4~7]。

由于只能对流数据作一次扫描，而且有限的内存无法存储整个数据流数据，流算法通常需要在内存中保存流数据的概要信息，以支持后续的计算。直方图是一种常用的概要数据结构，能有效保存数据分布信息。网格数据结构可看作是一种多维直方图，它保存了多维空间内的数据分布信息。聚类分析就是对数据分布特征的一种探索。因此，基于网格数据结构的压缩可用在流数据聚类分析中。但是，单纯的基于这种网格数据结构的算法无法有效处理高维数据，子空间聚类算法是一类特别设计用来处理高维数据的算法[8,9,11]。

本文提出一个基于网格的子空间聚类算法 GSCDS(Grid-based Subspace Clustering algorithm for high-dimensional Data Streams)。此算法结合了自底向上网格方法和自顶向下网格方法的优点，能快速、有效地处理高维数据流。算法首先使用自底向上的网格方法将数据空间均匀划分得到一个网格数据结构，数据流数据的分布信息被保存在这个网格数据结构中。然后使用自顶向下的网格划分方法将此网格数据结构中的簇分隔开，并将每个簇与相应的子空间相关联。最后，算法再次使用自底向上的网格方法，得到精确的簇的描述，并消除自顶向下网格划分方法所产生的错误。我们对算法的复杂度分析以及在多个数据集上所做的性能分析实验都验证了算法的效率和有效性。

本文第 2 小节介绍了与算法相关的工作，第 3 小节讨论了基于网格方法的聚类方法，并详细分析了两类不同的网格方法的优缺点。我们的子空间识别算法、主算法 GSCDS 及其复杂度分析在第 4 小节给出。第 5 小节报告了我们的实验及结果。最后总结全文。

2　相关工作

在高维数据空间，所有数据点之间的距离趋于相等[5]。这给基于数据点之间的距离定义它们之间相似度的聚类算法带来了困难。降低数据空间的维度是处理高维问题的一个有效方法。传统的降维方法有维选择、维转换等技术，即通过选择聚类属性较好的维，或通过对原数据空间的维作 PCA (principle component analysis)分析，得到一个比原数据空间维度低的子空间[13]。由于不同的簇可能位于不同的子空间，学者们提出了子空间聚类算法[8,9,11]。CLIQUE 是第一个子空间聚类算法[8]。它首先将数据空间均匀划分为等大小的网格单元，然后从一维空间开始，使用 APRIORI 技术逐步找到所有子空间内满足某个密度阈值的高密度网格单元，同一子

*)本文得到湖北省自然科学基金项目“时空数据库的关键技术研究与实验”(ABA048)的资助。**孙玉芬**　博士生，研究方向为流数据挖掘和聚类分析；**卢炎生**　教授，博导，研究方向为特种数据库、数据挖掘和软件测试。

空间内相连的高密度单元被作为聚类输出。CLIQUE 最大的缺点在于其运行时间与高密度网格单元的最高维度呈指数关系。另外，由于不同维度子空间内的数据分布密度无法进行比较，CLIQUE 对所有子空间使用同一个密度阈值也不是十分合理。CLIQUE 输出的簇之间可能会有重叠，即同一数据对象可能属于多个簇，这与某些应用的要求是矛盾的。MAFIA 试图通过将数据空间按照数据分布进行不均匀划分来改进算法 CLIQUE[9]。但是，由于基本方法与 CLIQUE 一样，它具有 CLIQUE 的所有缺点。SURFING 试图通过检查某个子空间内数据的投影是否均匀分布来判断此自空间是否包含簇[11]。它需要对每个子空间内的每个数据点计算其 k 个最近邻居，因此计算量巨大。

最近，学者们针对数据流提出了几种聚类算法[2, 4~7]。算法 STREAM[2] 采用分而制之(Divide-and- Conquer)的思想，将数据流划分为多个段，对每段分别聚类，得到第一层簇中心。当第一层簇中心达到一定数目后，对其进行聚类，得到第二层簇中心。这样的过程伴随数据的流入一直进行，在每个时刻，系统最多维持 m 个第 i 层中心点。算法 CluStream[4] 考虑了数据流的变化特性，将数据流数据在线压缩至微聚类(micro-clusters)内，并按金字塔时间结构(Pyramidal Time Frame)定期保存 micro-clusters 的快照，从而能够以较小的误差计算任意时间段内的聚类，并且能分析随时间的推进聚类所发生的改变。算法 HPStream[5] 是一个高维数据流子空间聚类算法，它同样使用微聚类压缩数据流信息。对每个得到的簇，HPStream 选择使簇的分布范围较小的维与其相关。在 HPStream 中，用户需要指定与各个簇相关的维的数目的平均值，即各个子空间的平均维度。CluStream 与 HPStream 不适于发现非凸面形状的簇，并且对噪音和数据的输入顺序敏感。Park 等人实现了一个基于网格的流数据聚类算法[6]。此算法需要假设网格单元内的数据分布类型，以便对初始的网格单元进行划分。它完全没有考虑高维数据的情况。算法 GCHDS 是作者早先提出的一个基于网格的高维数据流聚类算法[7]。它通过分析数据在每维投影的分布，选择对聚类分析有用的维。此算法所识别的所有簇都位于同一子空间中。但是对于真实数据集，不同的簇可能位于不同的子空间中。

与上述算法相比，本文提出的算法 GSCDS 具有如下优点：

1. 能在线识别不同子空间内任意形状的簇。

2. 不需要为子空间的识别指定任何参数(如密度阈值、子空间维度等)。

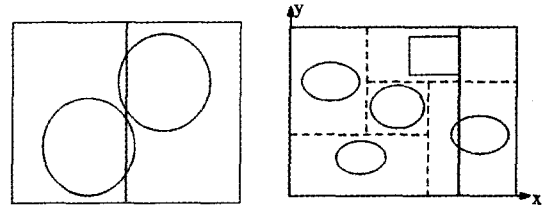
3. 计算复杂度低，能快速处理高维数据。

3 基于网格的聚类算法

对于聚类分析任务，基于网格方法的聚类算法具有很多理想的特性，如发现任意形状、任意大小的簇、计算结果与数据输入顺序无关、计算时间与数据量无关等。算法 GSCDS 充分利用了网格方法的多种优良特性。在介绍算法之前，我们首先讨论各类网格方法。

使用在聚类算法中的网格方法可被划分为两类，即由底向上的网格方法和自顶向下的网格方法。由底向上的网格方法基于用户输入的划分参数将数据空间均匀划分为等大小的网格单元，包含一定数目数据点的网格单元被称为高密度网格单元。聚类算法将所有相连的高密度网格单元识别为簇。此类算法假设落入同一网格单元内的所有数据点都属于同一个簇，每个网格单元仅保存落入其内数据的统计信息，如数据点个数、数据点之和等。其优点在于，它能通过对数据的一次扫描，将数据压缩到一个网格数据结构内，并基于此网格数据结构，发现任意形状的簇。簇的精度取决于网格单元的大小，如果网格单元的粒度较小，则聚类的精度较高，但是算法的计算复杂度较大。由底向上的网格方法不适合处理高维数据。在高维空间，数据的分布非常稀疏，网格方法失去其压缩作用，而且属于同一个簇的高密度网格单元也可能不相连，导致算法无法发现合理数目的簇。WaveCluster[12] 与 CLIQUE[8] 是采用由底向上网格方法的代表性算法。WaveCluster 处理低维空间数据，它的性能超越了 BIRCH、CLARANS 以及 DBSCAN 等优秀的聚类算法[3]。CLIQUE 主要考虑了高维子空间聚类。但正如前面所提到的，它的时间复杂度较高，而且需要用户指定全局密度阈值。

自顶向下的网格方法不需要用户指定划分参数，它根据数据的分布对空间进行划分。此方法采取分而治之的思想，将问题规模不断减小。原数据空间首先被划分为几个较大的区域。对于每个得到区域，划分过程被反复执行，直到每个区域仅包含属于同一个簇的数据点，这些大小不一的区域被称为网格单元。基于自顶向下网格方法的聚类算法直接将高密度网格单元识别为一个簇，或是将相连的高密度网格单元识别为簇。这种方法受数据空间的维度的影响较小，可快速将高维数据中的簇分隔开。由于划分方法使用的是数据的总体分布信息，而噪音通常被认为是在整个空间均匀分布的，因此它对噪音不敏感。但是，由于自顶向下的划分方法得到的网格单元的体积远大于由底向上网格方法中的网格单元体积，因此其簇的描述精度比由底向上的网格方法得到的簇的描述精度低。而且在自顶向下的划分过程中，同一个簇可能被划分到多个区域中，最终得到的同一区域也可能包含不同的簇，这些错误进一步降低了算法的精度。这类划分方法的另一个缺点是它在划分过程中，需要对数据集进行多次扫描。OptiGrid[10] 与 CLTree[14] 是两个典型的基于自顶向下网格方法的聚类算法。其中，CLTree 使用划分的信息增益来选取最优划分，而 OptiGrid 则是基于空间数据分布的密度信息来选择最优划分。下面详细讨论算法 OptiGrid。



(a) 每维都重叠的簇　(b) 可被平行于轴的划分线分隔开的簇

图 1　自顶向下的划分所引入的错误

OptiGrid 是一个典型的基于自顶向下网格方法的聚类算法，它按数据在每维上的投影的分布来划分空间。算法试图寻找空间中数据密度最低的位置将空间中的簇分隔开。对当前区域的每个数据维，算法选择数据投影密度的最低点来确定划分平面的位置，经过此点且垂直于此数据维的划分平面将区域划分为两个小的区域。算法最终得到多个由平行于轴的平面所构成的区域，每个区域至多包含一个簇。簇由包含它的区域所代表。OptiGrid 很容易实现，但是它也具有自顶向下网格方法的缺点，如需要对数据集进行多次扫描，对簇的

描述精确不高等。这里详细分析 OptiGrid 在划分过程中产生的错误，即一个簇被切割为多个簇，以及同一区域包含属于多个不同簇的数据。正如文[10]中所指出的，这些错误出现在两个簇在每维都有重叠的情况下。图 1(a)给出了一个二维空间的例子。在这种情况下，只要划分平面(划分线)是平行于数据轴的，错误的产生就无法避免。但是，我们发现，即使两个簇不是在每个数据维都有重叠，算法仍可能产生错误。图 1(b)描述了这种情况。图中的粗线条代表 OptiGrid 所选择的划分线。文[10]中没有考虑这类错误，但是它可能会大大降低算法的精度。特别是当数据空间中存在多个簇时，这个问题会非常明显。

正如上面所分析的，当划分参数选择恰当时，基于由底向上网格方法的聚类算法的聚类精度较高，但是它们无法处理高维数据。基于自顶向下网格方法的聚类算法能快速处理高维数据，但是其聚类结果精度较低。如果我们能结合这两类算法的优点，就能得到一个具有高精度、高效率的聚类算法。这就是我们设计本文算法的动机。下面介绍我们的高维数据流子空间聚类算法。

4 基于网格的高维数据流子空间聚类算法

本文仿照 CLIQUE[8]算法，将问题形式化如下。设 $S=A_1\times A_2\times\Lambda\times A_d$ 代表一个 d 维数据空间，其中，$A_1,A_2,\cdots,A_d$ 为 S 的 d 个数据维。流数据是此 d 维空间中的点 $X=X_1,X_2,X_3,\cdots$，每个点为 $X_i=(x_{i1},x_{i2},\cdots,x_{id})$。$X_i$ 的第 j 个属性值 $x_{i,j}$ 是它在维 A_j 上的取值，$X_{i,j}$ 的取值范围为 $[\min_j,\max_j)$。

我们首先将数据空间 S 均匀划分，得到一个网格数据结构 GS。S 的每一维被均匀划分为 k 段。来自各维的段相交形成了网格单元。网格单元 C_i 为 $(c_{i1},c_{i2},\cdots,c_{id})$，其中 $c_{ij}=[l_{mj},h_{mj}),1\leqslant m\leqslant k$ 是维 A_j 上的一个半开区间。落入每个网格单元的数据点数目由网格单元的 *count* 属性记录下来，*count* 也被称作网格单元的密度。网格 GS 中一共有 k^d 个网格单元。当数据空间的维度 d 较大时，k^d 将会很大。为了能在内存中存储整个网格数据结构，我们仅保存 $count_i>0$ 的网格单元。

4.1 子空间识别算法

子空间聚类算法的关键步骤就是识别簇所在的子空间。我们称簇所在的子空间的维与此簇是相关的。受 OptiGrid 算法的启发，我们将相关维定义为：

定义 1 对数据空间中的每一维，如果此维上的某个点能将属于某个簇的数据点投影与其它簇的数据点投影分隔开，这个数据维称为是与这个簇相关的。

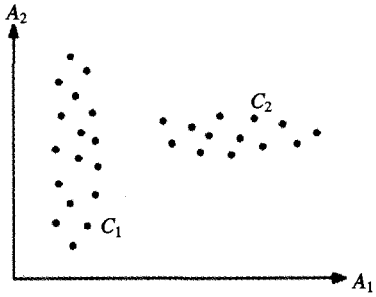


图 2 簇的相关维定义

我们的相关维的定义与其他子空间聚类算法所隐含的相关维的含义不太相同。其它子空间聚类算法如 CLIQUE 与 PROCLUS[15]是将簇在其上分布范围较小的维作为与簇相关的维，而我们的定义则完全是从簇的投影是否能与其他簇的投影分隔开来考虑的。图 2 说明了这种不同。CLIQUE 与 PROCLUS 将把维 A_1 与簇 C_1 相关联，维 A_2 与簇 C_2 相关联。而我们的定义将 A_1 与 C_1、C_2 相关联。

算法 OptiGrid 并没有涉及到子空间聚类问题，但是它所使用的自顶向下的划分方法却可以帮助我们快速找到与每个簇相关的维。例如，对于划分得到的区域 $R_i:(r_{i1},r_{i2},\cdots,r_{id})$，$r_{ij}\subseteq[\min_j,\max_j)$是一个半开区间，如果 r_{ij} 小于数据在维 j 上的取值范围$[\min_j,\max_j)$，则区域 R_i 的维 j 曾被某个划分面切割。这意味着维 j 上存在某个划分面能将区域 R_i 内的簇与其他簇分隔开，即维 j 是与 R_i 内的簇相关的。反之，如果区域 R_i 的某个维的范围与数据的取值范围相同，则表明即使不使用数据在此维上的值，R_i 内的簇也能与其它簇分隔开。因此，对每个包含簇的区域，我们可以找到其取值范围小于数据取值范围的维。这些维与区域内的簇相关，并且提供了足够的信息将此区域内的簇与其他簇分隔开。它们构成了这个簇所在的子空间，因此我们可以通过自顶向下的划分过程自然地找到与每个簇相关联的子空间。

自顶向下的划分方法需要多次扫描整个数据集，而流数据只能被扫描一次。为了解决这个问题，我们首先将数据流数据压缩到一个均匀划分的网格数据结构 GS 中，然后在此网格数据结构上进行处理。在下面的子空间识别算法中，划分程序对空间进行递归划分直到没有区域需要划分或是划分程序已经执行了 n 次，n 为控制划分次数的参数。

算法 1 子空间识别算法

输入：d 维网格 GS，执行次数 n，噪音水平 nl

输出：区域集 RR，$|RR|$ 个维子集 DI

1. 令 $CR=\{GS\}$代表候选区域集，置 $iteration\_num=0$。

2. 对 CR 中的每个区域 R_l：

2.1. 对每个数据维 $j,1\leqslant j\leqslant d$：

2.1.1. 将 R_l 内的网格单元投影到维 j。对于维 j 上的区间 $i,[l_{ij},h_{ij})\subseteq r_{ij}$，其密度为 $D_{ij}=\{\sum_p count_p\mid C_p\in R_l,c_{pj}=[l_{ij},h_{ij})\}$。

2.1.2. 设 $D_{i,j}$ 为密度的局部最小点，若 $\max(D_{m_j,j},D_{m_j+1,j},\cdots,D_{i-1,j})>N_{R_l}/(nl\times k_j)$，且 $\max(D_{i+1,j},D_{i+2,j},\cdots,D_{m_j+k_j-1,j})>N_{R_l}/(nl\times k_j)$，则将区间$[l_{ij},h_{ij})$中的 h_{ij} 插入候选划分点 CCP 集中。其中 N_{R_l} 为落入区域 R_l 的数据点数目，k_j 为 R_l 在维 j 上的区间数目，m_j 为 R_l 在维 j 上的最小区间，m_j+k_j-1 为 R_l 在维 j 上的最大区间。D_{ij} 与 $\left|\sum_{p=m_j}^{i-1}D_{pj}-\sum_{p=i+1}^{m_j+k_j-1}D_{pj}\right|$ 与 h_{ij} 一起被存储。

2.2. 若 CCP 非空，我们从中选择密度 D_{ij} 最小的点作为划分点。如果几个划分点具有相同的最小密度，我们从中选择具有最小 $\left|\sum_{p=m_j}^{i-1}D_{pj}-\sum_{p=i+1}^{m_j+k_j-1}D_{pj}\right|$ 值的点作为划分点。

2.3. 设 h_{ij} 是从 CCP 中选择的划分点，则区域 R_l 被经过 h_{ij} 点，垂直于维 j 的超平面切割为两个区域 R_l' 和 R_l''。更新候选区域集 $CR=CR+R_l'+R_l''-R_l$。若 CCP 为空，则直接将 R_l 从候选区域集 CR 移至结果区域 RR 集中。

3. 令 $iteration\_num=iteration\_num+1$。若 $iteration\_$

$num<n$ 且 CR 非空，转到第二步。

4. 若 CR 非空，将 CR 中的区域都移至 RR 中。

5. 对 RR 中的每个区域 R_l，

5.1. 对每一维 j，$1\leqslant j\leqslant d$，若 $r_{lj}=[l,h)$ 满足条件 $l>\min_j$ 或 $h<\max_j$，则将 j 插入与 R_l 相关联的维子集中。

在上述算法中，DI_i 中的维构成 R_i 中的簇所关联的子空间。需要指出的是，如果原数据空间中的所有点都投影到 DI_i 所张的子空间内，则 R_l 中的簇可能会无法识别，这是因为算法所计算的划分面可能会对簇产生切割。例如，图 1(b) 中最右方的簇所关联的 DI 仅包含 x 轴，但是若将所有数据都投影到 x 轴，这个簇是无法被识别的。因此，为了识别簇，除了识别与簇相关的子空间，还必须记录包含簇的区域的范围。在下一小节的主算法中，我们将对被切割的簇进行还原。

算法中的参数 nl 表示数据中的噪音水平。在实验部分，nl 的缺省值被设置为，即数据中的噪音量不超过整个数据量的。这对大部分数据集来说是合理的。用户可以根据对数据集的先验知识设置 nl 的值。算法第 2.2.1 步对候选划分点左右密度最大值的要求是为了保证此候选点左右两边都存在簇。算法会选择具有最小密度的候选点作为划分点，若多个候选点具有相同的最小局部密度，则算法会选择将区域内数据均匀分割的点作为划分点。这是因为对于一个分而治之的方法，每次将任务均匀划分是最有效的。

4.2 算法 GSCDS

算法 GSCDS 在内存中维持一个均匀划分的网格数据结构，以增量式地压缩数据流数据。当聚类请求到来时，子空间识别算法和聚类过程在此网格数据结构上执行。正如所有基于由底向上网格方法的聚类算法一样，GSCDS 的性能受此网格数据结构的划分粒度所影响。在实验部分，我们将讨论网格划分参数的选取问题。

由于数据流数据是在不断变化的，历史数据中的簇可能会与当前数据中的簇不同。而大部分应用只关心当前数据内存在的结构。为避免历史数据对当前计算造成影响，每当有新的数据到来时，算法将每个网格单元的 count 乘以一个衰减参数 $\varepsilon<1$，只有 $count>\xi$ 的网格单元才被保存。其中 $\xi<1$ 是密度阈值。衰减参数 ε 与密度阈值 ξ 一起作用，消除历史数据对当前计算的影响。

算法 2 GSCDS 算法

输入：数据流 DS，参数 k,ε,ξ

输出：带簇标识的网格 GS

1. 将数据空间的每一维均匀划分为 k 段以构建网格 GS。

2. 对于一个新到达的数据点 $X_i=(x_{i1},x_{i2},\cdots,x_{id})$，找到满足条件 $x_{ij}\in c_{mj}$，$1\leqslant j\leqslant d$ 的网格单元 C_m，更新 $count_m$ 为 $count_m+1$。

3. 对每个网格单元，更新其 $count$，$count=count\times\varepsilon$。删除 $count<\xi$ 的网格单元。

4. 如有聚类请求，转入步骤 5；否则，转入步骤 2。

5. 运行子空间识别算法得到区域集 RR，以及与每个区域相关联的维子集 DI_i，$1\leqslant i\leqslant|RR|$。

6. 对 RR 中的每个区域 R_l，

6.1. DI_l 中的维构建了一个子空间 SS_l，将 R_l 中的网格单元 $C_i\in R_l$ 都投影到 SS_l 中。

6.2. 在 SS_l 中，将所有相连的网格单元标志为一个簇。

6.3. 原数据空间中的网格单元得到它们在 SS_l 中的投影的簇标识。

6.4. 对区域 R_l 中的每个簇 Clu_i，

6.4.1. 对 DI_l 中的每一维 j，

6.4.1.1. 设 R_l 在维 j 上的范围为 $[l_{mj},h_{m'j})$，$1\leqslant m,m'\leqslant k$。若 $l_{mj}>\min_j$，找到集合 $CL_{ijm}=\{C_p\mid C_p\in Clu_i,c_{pj}=[l_{mj},h_{mj})\}$；若 $h_{m'j}<\max_j$，找到集合 $CR_{ijm'}=\{C_p\mid C_p\in Clu_i,c_{pj}=[l_{m'j},h_{m'j})\}$。

6.4.1.2 若 CL_{ijm} 非空，计算 CR_{ijm} 在除了维 j 外所有维上的范围；若 $CR_{ijm'}$ 非空，计算 $CR_{ijm'}$ 在除了维 j 外所有维上的范围。

7. 对任意两个簇 Clu_i 与 Clu_l，若存在非空集 CL_{ijm} 与 CR_{ljm} 满足 $m=m'+1$，并且它们在除维 j 外的所有维上的范围都是重叠的，合并这两个簇。

8. 转到第 2 步。

在子空间中，数据点的分布比原数据空间要密集得多，因此我们能将相连的高密度网格单元作为簇。由于区域之间是相互不重叠的，原数据空间中的网格单元至多只会得到一个簇标识。

正如第 3 小节所指出的，自顶向下的网格划分方法可能会将一个簇分割到几个区域中去。为判断两个簇是否本应属于同一个簇，算法首先检查它们是否与同一划分面相邻，若是，则进一步检查它们与此划分面相邻的范围是否相互重叠。在算法中，第 6.4 步负责找出每个簇与划分面相邻的范围，步骤 7 将满足条件的两个簇合并。

4.3 算法时间复杂度分析

由于区域个数与数据中簇的数目成正比，子空间识别算法的时间复杂度为 O(簇的数目$\times d\times$高密度网络单元个数)。其中，高密度网格单元指的是 $count\geqslant\xi$ 的网格单元。

算法 GSCDS 处理每个数据点的时间为 $O(d\times$高密度网络单元个数)，聚类时间为 $O(d\times$高密度网络单元个数$^2)$。如果对寻找相连高密度网格单元的步骤进行优化，算法的时间复杂度还能被进一步降低。

5 实验结果

在本小节，我们通过在多个数据集上的实验验证算法 GSCDS 的计算精度与运行效率。实验所使用的两个真实数据集曾被文[5]用来评估算法 HPStream。我们将算法 GSCDS 与 GCHDS 进行了比较。GCHDS 是我们的一个较早的算法，它利用数据在每个数据维上的投影的分布选择对聚类有用的维[7]。在这个算法中，所有的结果簇都位于同一子空间中。对于文[7]中所使用的数据集，GCHDS 在计算精度与效率上都超越了算法 HPStream。关于此算法更详细的描述请参见文[7]。我们还对算法 GSCDS 进行了参数敏感度分析。所有的实验都在一个 CPU 为 AMD 2500+，内存为 512 MB，操作系统为 Windows XP 的 PC 机上进行。算法是用 Microsoft Visual C++ 实现的。

我们使用了两个具有不同数据特征的真实数据集来评估我们的算法。第一个数据集是 KDD-CUP'99 网络入侵检测流数据集。此数据集一共有 494020 条记录，每条记录有 34 个连续属性。第二个数据集是 Forest CoverType 数据集。这个数据集是从 UCI 的机器学习数据集网站(HTUhttp://www.ics.uci.edu/~mlearnUTH)下载的。它包含 581012 条记录，每条记录有 10 个数字属性参与聚类分析。这两个数据集的每条记录都带有类标识。除了这两个真实数据集外，我们还自己生成了一个简单的二维数据集来验证合并簇的方法

是否有效。

与文[7]中一样，我们使用滑动时间窗口内簇的平均纯度来衡量算法计算的精度。数据流流速为一个时间单位100个数据点，滑动时间窗口的长度为1000个数据点。一个簇的纯度表示簇中属于簇的主体类别的数据点在簇中所占的比例。纯度的计算使用了数据的类别信息。

为了避免数据衰减得太快，我们仅在每1000个新数据到达后才对网格单元的count值进行衰减。在我们的实验中，ε的缺省值为0.9，ξ为0.8。对于子空间辨识算法，参数n的缺省值为20，nl为10。对于网络入侵检测数据集，算法GCHDS选择20个维构建子空间。对于Forest CoverType数据集，GCHDS选择8个维构建子空间。

图3与图4是算法GSCDS与GCHDS的计算精度的比较。其中，图3给出的是两个算法对入侵检测数据集计算精度的比较，图4给出的是它们对Forest CoverType数据集计算精度的比较。在这两个试验中，k都被置为20。我们可以看到，GSCDS的计算精度比GCHDS高。而且，如图4所示，当数据的分布较复杂时，GCHDS的计算精度很差，因为它所使用的维选择方法过于简单。

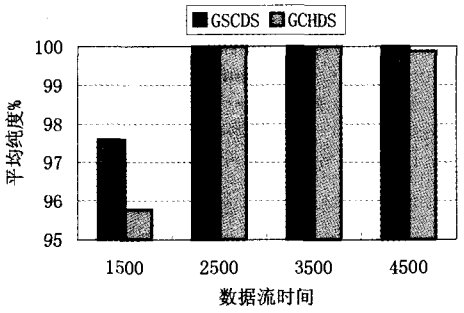


图3 计算精度比较(入侵检测数据集)

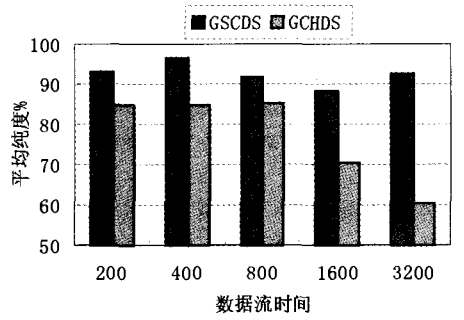


图4 计算精度比较(Forest CoverType数据集)

图5是GSCDS与GCHDS的聚类时间比较。由于两个算法维持网格数据结构的时间是一样的，因此图中的时间仅包含在网格数据结构上执行聚类分析的时间。GSCDS的子空间识别方法比GCHDS复杂，它所需要的时间也更多。

对于算法GSCDS，参数n，nl，ε与ξ对算法质量的影响不大。但是划分参数k的取值对算法的计算精度影响很大。图6给出的是k取不同值时，算法对Forest CoverType数据集的聚类精度。显然，当k很小时，算法的计算精度很低。根据我们在多个数据集上的实验结果可以发现，只要k不小于数据集中簇的数目的10倍，算法的计算精度都比较高。但是这个参数选取原则只是针对本文的算法，不能直接用于其他由底向上的网格聚类算法。

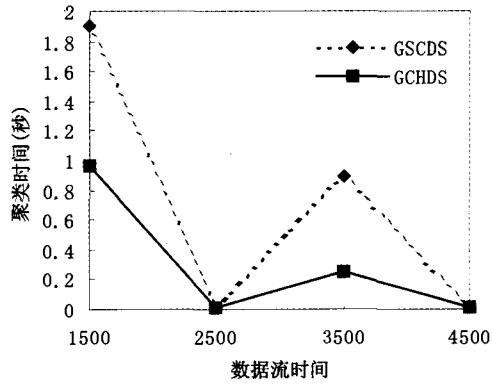


图5 聚类时间的比较(入侵检测数据集)

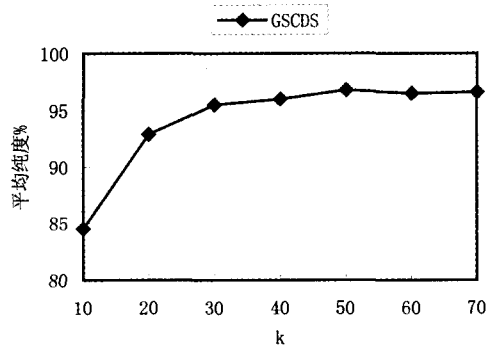


图6 k取不同值时聚类精度的变化(Forest CoverType数据集)

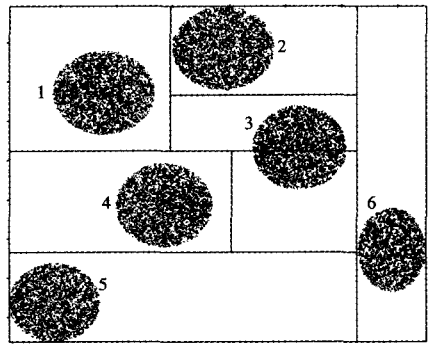


图7 融合功能的验证

我们生成了一个简单的二维数据集来检验算法对于被划分面切割的簇的还原能力。图7表示此数据集以及算法生成的区域和簇标识。从图中可以看出，标识为3的簇被划分面切割到两个区域中，但是算法仍然能成功识别出这个簇。

结论 本文给出了一个基于网格方法的子空间聚类算法GSCDS，以在线聚类高维数据流。此算法充分利用了自顶向下网格方法和由底向上网格方法的优点，实现对高维数据流快速、精确的聚类。实验结果表明算法的计算精度超过了另一个相似的算法GCHDS。

参考文献

1 Henzinger M R, Raghavan P, Rajagopalan S. Computing on data streams. SRC Technical Note 1998-011. Digital systems research center: Palo Alto, California, 1998

2 O'Callaghan L, et al. Streaming-Data Algorithms for High-Quality Clustering. In: Proc. of the 18th Intl. Conf. on Data Engineering (ICDE'02), 2002. 685～694

3 Han J, Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, 2001

(下转第221页)

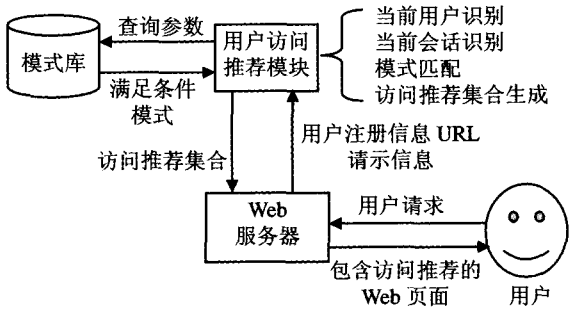


图 7　用户访问推荐示意图

4 Pierrakos D, Paliouras G. Web Usage Mining as a Tool for Personalization: A Survey [J]. Kluwer Academic Publishers, 2003. 311～372

5 张慧颖,焦霖楠. 用户访问模式聚类分析在网页推荐中的应用[J]. 计算机工程,2006,32(15):64～66

6 Han Jiawei, Kamber M. Data Mining: Concepts and Technlques [M]. Morgan Kaufmann Publishers. inc, 2001

7 Wu Song, Jin Hai, Tan Guang. Symmetrical Declustering: A Load Balancing and Fault To lerant Strategy for Clustered Video Servers [C]. In: International Conference on Computational Science and Its Applications, 2003. 199～208

8 詹宇斌,殷建平,张玲,等. 一种基于有向树挖掘 Web 日志中最大频繁访问模式的方法[J]. 计算机应用,2006,26(7):1662～1665

9 Han J, Kamber M. Data mining: concepts and techniques [M]. San Francisco: Morgan Kaufmann Publishers Inc, 2001

10 Scime A. Web mining: application and techniques [M]. Hershey: Idea Group Publishing, 2004

(上接第 203 页)

4 Aggarwal C C, et al. A Framework for Clustering Evolving Data Streams. In: Proc. of the 29th VLDB Conf., 2003. 81～92

5 Aggarwal C C, et al. A Framework for Projected Clustering of High Dimensional Data Streams. In: Proc. of the 30th VLDB Conf., 2004. 852～863

6 Park N H, Lee W S. Statistical Grid-Based Clustering over Data Streams. ACM SIGMOD Record, 2004, 33(1): 32～37

7 Lu Y, et al. A Grid-Based Clustering Algorithm for High- Dimensional Data Streams. In: Proc. of the 1st International Conference on Advanced Data Mining and Applications, 2005. In Lecture Notes in Computer Science, 2005, 3584: 824～831

8 Agrawal R, et al. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'98), 1998. 94～105

9 Goil S, et al. MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets: [Technical Report, No. CPDC-TR-9906-010]. Center for Parallel and Distributed Computing, Department of Electrical & Computer Engineering, Northwestern University, 1999

10 Hinneburg A, Keim D A. Optimal Grid-Clustring: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In: Proc. of the 25th VLDB Conference, 1999. 506～517

11 Baumgartner C, et al. Subspace Selection for Clustering High-Dimensional Data. In: Proc. 4th IEEE Int Conf On Data Mining (ICDM'04), 2004. 11～18

12 Sheikholeslami G, Chatterjee S, Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In: Proc. of the 24th VLDB Conference, 1998. 428～439

13 Parsons L, Haque E, Liu H. Subspace Clustering for High Dimensional Data: A Review. ACM SIGKDD Explorations Newsletter, 2004, 6(1): 90～105

14 Liu B, Xia Y, Yu P S. Clustering Through Decision Tree Construction. In: Proc. of the Ninth International Conference on Information and Knowledge Management, 2000. 20～29

15 Aggarwal C C, et al. Fast Algorithms for Projected Clustering. In: Proc. of the 1999 ACM SIGMOD International Conference on Management of Data, 1999. 61～72

(上接第 212 页)

大,即意味提高对模式重复程度的限制,则抽取的准确性即查准率可以得到提高,但有可能丢失掉一些重复程度并不十分频繁的数据记录,使得查全率有所下降。而 MinLen 值增大,意味着具备相当结构特征的数据记录才会被抽取,提高了查准率,但查全率会因此受到一定影响。我们可以根据实际需要,通过适当调整参数来取得较好的抽取效果。

结束语　本文描述了一种基于重复模式的自动抽取网页信息的方法。根据页面的 HTML tag 序列构造后缀树,利用后缀树的优越性质取得序列中所有的重复模式以及对应的实例;应用相关规则,对候选模式进行过滤和精化,最后保留合理的模式,并从其实例中抽取网页数据记录的信息。实验结果表明,该方法是有效可行的。目前,我们正致力于进一步改善抽取的准确性,同时对系统的性能进行优化。

参 考 文 献

1 Laender A, Ribeiro-Neto B, Silva A, et al. A brief survey of Web data extraction tools. SIGMOD Record, 2002, 31(2)

2 Arasu A, Garcia-Molina H. Extracting Structured Data from Web Pages. SIGMOD-03, 2003

3 Chang C H, Lui S L. IEPAD: Information extraction based on pattern discovery. WWW-10, 2001

4 Embley D W, Jiang Y, Ng Y K. Record-Boundary Discovery in Web Documents. In: Proc. SIGMOD'99, 1999

5 McCreight E. A space-economical suffix tree construction algorithm. Journal of the ACM, 1976, 23: 262～272

6 Ukkonen E. On-line construction of suffix trees. Algorithmica, 1995, 14: 249～60

7 Muslea I, Minton S, Knoblock C. A Hierarchical Approach to Wrapper Induction. In: Proceedings of the 3rd International Conference on Autonomous Agents, 1999

8 Kushmerick N, Weld D, Doorenbos B. Wrapper induction for information extraction. In: Proc. Int Joint Conf. Artificial Intelligence, 1997

9 Soderland S. Learning Information Extraction Rules for Semi-structured and Free Text. Machine Learning, 1999

10 http://blogs.law.harvard.edu/tech/rss