An Incremental Data Stream Clustering Algorithm Based on Dense Units Detection

Jing Gao[1], Jianzhong Li[2], Zhaogong Zhang[2], and Pang-Ning Tan[1]

[1] Dept. of Computer Science & Engineering, Michigan State University
East Lansing, MI 48824-1226 USA
gaojing2,ptan@cse.msu.edu
[2] Dept. of Computer Science & Technology, Harbin Institute of Technology
Harbin, 150001 China
lijz,zhangzhaogong@mail.banner.com.cn

Abstract. The data stream model of computation is often used for analyzing huge volumes of continuously arriving data. In this paper, we present a novel algorithm called DUCstream for clustering data streams. Our work is motivated by the needs to develop a single-pass algorithm that is capable of detecting evolving clusters, and yet requires little memory and computation time. To that end, we propose an incremental clustering method based on dense units detection. Evolving clusters are identified on the basis of the dense units, which contain relatively large number of points. For efficiency reasons, a bitwise dense unit representation is introduced. Our experimental results demonstrate DUCstream's efficiency and efficacy.

1 Introduction

In recent years, data stream model is motivated by many applications that continuously generate huge amount of data at unprecedented rate [1]. In this paper, we will focus on the stream clustering problem, which is a central task of data stream mining.

Recently this problem has attracted much attention. O'Callaghan et. al. [2] study the k-median problem over data streams. Aggarwal et. al. [3] present a framework of clustering evolving data streams, which analyzes the clusters over different portions of the stream. However this framework can not give online response of queries of macro clusters. Nasrouni et. al. [7] design an immune system learning model to find evolving clusters in data streams. But this algorithm is not space and time efficient due to the use of AIS model.

In static data environment, many clustering algorithms have been designed [4–6]among which grid-based clustering is an efficient method. This approach partitions the data space into many units and perform clustering on these units [6]. Recently, Park et.al. [8] propose a statistical grid-based method which identifies evolving clusters as a group of adjacent dense units in data stream environments. But their work is focusing on partitioning dense units and maintaining their distributions.

In this paper, we propose an efficient data stream clustering algorithm DUC-stream. We partition the data space into units and only keep those units which contain relatively large number of points. An incremental clustering algorithm is presented based on these dense units. The clustering results are represented by bits to reduce the memory requirements. Extensive experiments indicate that our framework can obtain high-quality clustering with little time and space.

2 Problem Statement

We begin by defining the stream clustering problem in a formal way.

Suppose S is a d-dimensional numerical space. For each dimension, we partition it into non-overlapping rectangular units. The density of a unit u is defined as the number of points that belong to it, i.e. $\text{den}(u) = |v_i|v_i \in u|$. The relative density of u is defined as follows: $\text{rel\_den}(u) = \text{den}(u)/|D|$, where $\text{den}(u)$ is the density and D is the data set we observe. If u's relative density is greater than the density threshold γ, then u is referred to as a dense unit.As defined in [6], a cluster is a maximal set of connected dense units in d-dimensions.

A data stream is a set of points from data space S that continuously arrives. We assume that data arrives in chunks $X_1, X_2, \ldots, X_n, \ldots$, at time stamps $t_1, t_2, \ldots, t_n, \ldots$. Each of these chunks fits in main memory. Suppose that each chunk contains m points, and the current time stamp is t. We use $\text{den}(u)$ to denote the overall density of u with respect to the t chunks that has been seen so far. The density of u with respect to the i-th chunk is denoted as $\text{den}^i(u)$. The relative density of a unit u is $\text{rel\_den}(u) = \text{den}(u)/(mt)$. If u's relative density is greater than the density threshold γ, then u is referred to as a dense unit at time t. At time t, the clustering result R is all the clusters found in the t chunks of data visited so far. Our goal is to compute the clustering results when the data stream continuously arrives, i.e. obtain $R_1, R_2, \ldots, R_n, \ldots$, where R_i represents the result of clustering $X_1, X_2, \ldots, X_i$.

3 Algorithm Description

3.1 Basic Idea

In brief, we will find the dense units and cluster these units. First, we consider what units should be maintained thus introduce the concept of local dense units.

Suppose that each chunk contains m points, and the current time stamp is t. If unit u begins to be maintained at time i, the local relative density of u is $\text{loc\_den}(u) = \text{den}(u)/(m(t-i+1))$, where $\text{den}(u)$ is the density of u. If u's local relative density is greater than the density threshold γ, then u is referred to as a local dense unit at time t. The following proposition holds on.

Proposition 1 *For any dense unit u at time t, it must be recorded as a local dense unit at time $i(1 \leq i \leq t)$.*

Proof. Suppose that a dense unit u is not recorded as a local dense unit at time $1, 2, \ldots, t$ and each chunk contains m points. We recall that the number of points that belong to u in the i-th chunk is $\text{den}^i(u)$. Then $\text{den}^i(u) < \gamma m$. Therefore at time t, $\text{den}(u) = \sum_{i=1}^{t} \text{den}^i(u) < \gamma m t$. so u is not a dense unit at current time, contrary to the hypothesis. The conclusion is accordingly established.

In other words, local dense units are candidate dense units, which may become dense in the future. Therefore we maintain all the local dense units and pick up dense units among them to do clustering. We call this process dense units detection. The following proposition analyzes the error of our algorithm.

Proposition 2 *Assume that a certain unit u's density gradually increases so that its density with respect to the i-th chunk is ipm where m is the number of points belonging to each chunk, p is a constant from 0 to 1 that indicates the amount of increase. At the time from $(1 + \sqrt{1 + 8\gamma/p})/2$ to γ/p, this unit can not be successfully detected as a dense unit.*

Proof. According to the definition of local dense units, we will not keep unit u as long as $ipm < \gamma m$, i.e., $i < \gamma/p$. However, when its density reaches γm, u becomes a dense unit at that time. Suppose at time k, u's density is equal to γm. Then $\sum_{i=1}^{k} ipm = \gamma m$,i.e.,$\frac{k(k-1)}{2}p = \gamma$. It can be derived that $k = (1 + \sqrt{1 + 8\gamma/p})/2$. Therefore the time range when error occurs is as stated.

Another issue is how to get the right results with little time and memory. To lighten the computational and storage burden, we propose to represent the clustering results in bits. Suppose that the dense units are sorted by their density and each of them is assigned a unique id. The Clustering Bits (CB) of a cluster r is a $0-1$ bit string $a_n, \ldots, a_1$, where a_i is a bit and n is the number of dense units. $a_i = 1$ if and only if the i-th dense unit is in cluster r, otherwise $a_i = 0$. We can benefit from the use of Clustering Bits in both the time and space usage.

3.2 Stream Clustering Framework

Based on the above two points, we summarize our stream clustering algorithm in Figure (1). We refer to this algorithm as DUCstream (**D**ense **U**nits **C**lustering for data **stream**). The data structures used in the algorithm include: L, the local dense units table; Q_a, the added dense units id list; Q_d, the deleted dense units id list; R_i, the clustering result $\{c_1, \ldots, c_s\}$ at time stamp i.

The important components in this framework entail:

1. map_and_maintain(X_i, L): This procedure maps each data point in X_i into the corresponding unit. For one of these units u, if it is in L, update the corresponding item, otherwise if u is a local dense unit, insert it into L. After that, scan L once and decide Q_a and Q_d.

2. create_clusters(Q): We use a depth-first search algorithm to create clusters as described in [6]. They identify the clusters as the connected components of the graph whose vertices represent dense units and whose edges correspond to the common faces between two vertices.

DUCstream Algorithm:
Input: Data chunks $X_1, X_2, \ldots, X_n, \ldots$
Output: Clustering results $R_1, R_2, \ldots, R_n, \ldots$
Method:
1. Create a new empty table L;
2. (L, Q_a, Q_d)=map_and_maintain(X_1, L);
3. R_1=create_clusters(Q_a);
4. i=2;
5.Repeat until the end of the data stream
5.1 (L, Q_a, Q_d)=map_and_maintain(X_i, L);
5.2 R_i=update_clusters(R_{i-1}, Q_a, Q_d);
5.3 $i = i + 1$;

Fig. 1. SemiSOD algorithm framework

3. update_clusters(R_{i-1}, Q_a, Q_d): We get the clustering result R_i in an incremental manner stated as follows.

For each added dense unit u, one of following occurs: **Creation:** If u has no common face with any old dense units, a new cluster is created containing u; **Absorption:** There exits one old dense unit u' such that u has common face with u', then absorb u into the cluster u' is in; **Mergence:** There exist multiple old dense units $w_1, w_2, \ldots, w_k (k > 1)$ that have common faces with u, then merge the clusters these dense units belong to. Absorb u into the new cluster.

For each deleted dense unit u, suppose it is contained in cluster c, we can distinguish the following cases: **Removal:** If there are no other dense units in c, i.e. the cluster becomes empty after deleting u, we remove this cluster; **Reduction:** All other dense units in c are connected to each other, then simply delete u from c; **Split:** All other dense units in c are not connected to each other, this leads to the split of cluster c.

After processing all the units in Q_a, Q_d, we can obtain the new clustering result R_i.

4 Empirical Results

The data set is KDD'99 Intrusion Detection Data, which is partitioned into chunks each consisting of $1K$ points. We first examine the time complexity of DUCstream compared with the baseline methods STREAM [2] and CluStream [3]. To make the comparison fair, we make the number of clusters all five in these algorithms.Figure (2) shows that DUCstream is about four to six times faster than STREAM and CluStream. This is attributed to our use of dense units detection, Clustering Bits and good design of incremental update algorithm.

DUCstream maintains the local dense units and current clustering results in main memory. Since the clustering results, represented by Clustering Bits, cost very little space, we only keep track of the number of local dense units to monitor the memory usage. Figure (3) demonstrates that after a certain time, a steady state is reached as for the number of local dense units. In general, the algorithm

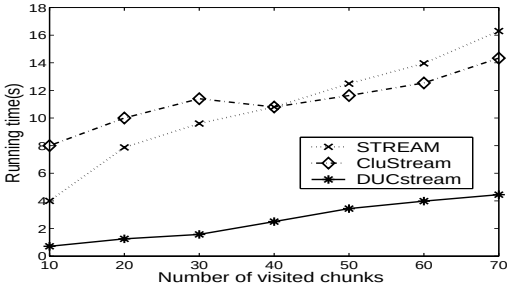

Fig. 2. Running time

only requires a negligible amount of memory even when the data stream size becomes sufficiently large.

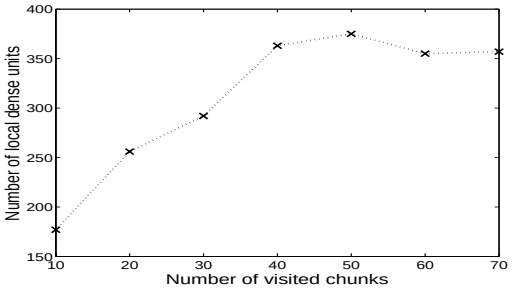

Fig. 3. Memory usage

We then compare DUCstream with STREAM and CluStream using the measurement SSQ, the sum of square distance. Figure (4) shows that the clustering quality of DUCstream is always better than that of STREAM because we capture the characteristics of clusters more precisely using the dense units compared with only maintaining k centers. For CluStream, it performs better when the horizon is small but the accuracy tends to be lower when the horizon becomes larger.

5 Conclusion

In this paper, we propose an efficient data stream clustering algorithm based on dense units detection. This is an incremental, one-pass density-based algorithm, which finds high-quality clusters with considerably little time and memory in the data stream environment. It discards noisy and obsolete units through dense units detection. The clustering result is updated using the changed dense units. We also introduce a bitwise clustering representation to update and store away the clustering results efficiently. Empirical results prove that this algorithm has

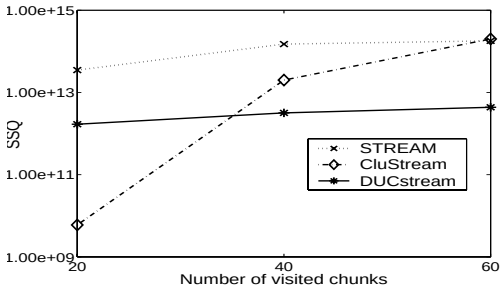

Fig. 4. Memory usage

good quality while cost surprisingly little time. The problem of finding arbitrary-shaped clusters is an interesting future work.

6 Acknowledgment

The work was partially supported by IRGP grant #71-4823 from the Michigan State University.

References

1. B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.
2. L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra and S. Guha. Streaming-data algorithms for high-quality clustering. In *Proceedings of IEEE International Conference on Data Engineering*, pages 685–696, 2002.
3. C. Aggarwal, J. Han, J. Wang and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the International Conference on Very Large Data Bases*, pages 81–92, 2003.
4. M. Ester, H.P. Kriegel, J. Sander and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–231, 1996.
5. M. Ester, H.P. Kriegel, J. Sander, M. Wimmer and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the International Conference on Very Large Data Bases*, pages 323–333, 1998.
6. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering for high dimensional data for data mining applications. In *Proceedings of the ACM International Conference on Management of Data*, pages 94–105, 1998.
7. O. Nasraoui, C. Cardona, C. Rojas and F. Gonzlez. TECNO-STREAMS: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proceedings of the IEEE International Conference on Data Mining*, pages 235–242, 2003.
8. N.H. Park and W.S. Lee. Statistical grid-based clustering over data streams. *ACM SIGMOD Record*, 33(1):32–37, 2003.