Clustering Categorical Data Streams[*]

Zengyou He, Xiaofei Xu, Shengchun Deng

Department of Computer Science and Engineering Harbin Institute of Technology,

92 West Dazhi Street, P.O Box 315, P. R. China, 150001

Email: zengyouhe@yahoo.com, xiaofei@hit.edu.cn, dsc@hit.edu.cn

Abstract The data stream model is relevant to new classes of applications involving massive datasets, such as web click stream analysis and detection of network intrusions. The cluster analysis on evolving data stream becomes more difficult, because the data objects in the stream must be accessed in order and can read only once or a small number of times with limited resources. In more recently years, a few clustering algorithms have be developed for data stream problem. However, to our knowledge, there is nothing to date in the literature describing clustering algorithms for categorical data streams. This paper presents an effective categorical data stream clustering algorithm. The proposed algorithm has provably small memory footprints. We also provide empirical evidence of the algorithm's performance on real datasets and synthetic data streams.

Keywords Clustering, Categorical Data, Data Stream, Data Mining

1. Introduction

For many recent applications, the concept of *data stream* is more appropriate than a dataset. By nature, a stored dataset is an appropriate model when significant portions of the data are queried again and again, and updates are relatively infrequent. In contrast, a data stream is an appropriate model when a large volume of data is arriving continuously and it is either unnecessary or impractical to store the data in some form of memory. Data streams are also appropriate as a model of access to large data sets stored in secondary memory where performance requirements necessitate linear scans [13].

In the data stream model, data points can only be accessed in the order of their arrivals and random access is disallowed. And the space available to store information is supposed to be small relatively to the huge size of unbounded streaming data points. Thus, the data mining algorithms on data streams are restricted to be able to fulfill their works with only one pass over data sets and limited resources. It is a very challenging research field.

Clustering typically groups data into sets in such a way that the intra-cluster similarity is maximized while the inter-cluster similarity is minimized. The clustering technique has been extensively studied in many fields such as pattern recognition, customer segmentation, similarity search and trend analysis.

Most previous clustering algorithms focus on numerical data whose inherent geometric properties can be exploited naturally to define distance functions between data points. However,

[*] This work was supported by the High Technology Research and Development Program of China (No. 2002AA413310, No. 2003AA4Z2170) and the IBM SUR Research Fund.

much of the data existed in the databases is categorical, where attribute values can’t be naturally ordered as numerical values. An example of categorical attribute is *shape* whose values include *circle*, *rectangle*, *ellipse*, etc. Due to the special properties of categorical attributes, the clustering of categorical data seems more complicated than that of numerical data.

Developing clustering algorithms for categorical data streams is undoubtedly very important for real applications such as web click stream analysis and information content security. While some recent work has been done on designing clustering algorithms for data streams in which the data objects contain numeric values, to the best of our knowledge, there is no published work on how to clustering categorical data streams.

The goal of this paper is to develop an effective clustering algorithm to cluster categorical data stream. We begin by reviewing related work on stream data mining, clustering data streams and categorical data clustering. In the sequels, we describe our algorithm in detail and provide empirical evidence of the algorithm’s performance.

2. Related Work

2.1 Related Work on Mining Data Streams

More recently, there has been some initial work addressing data streams in the data mining community. These proposals tried to adapt traditional data mining technologies to the data stream model.

References [1-3] focus on efficiently constructing decision trees and the problem of ensemble classification in data stream environment. Reference [4] presents an online classification system based on info-fuzzy networks.

Reference [5] discusses the problem of frequent pattern mining in data streams. The authors in [6] proposed algorithms for regression analysis of time-series data streams.

Reference [7] considers extracting information about customers from a stream of transactions and mining it in real-time. Reference [8] proposes Hancock, which is a language for extracting signatures from data streams.

The authors in [9,10] address the problem of mining multiple data streams. Reference [9] develops algorithms for analyzing co-evolving time sequences to forecast future values and detect correlations. Reference [10] presents a collective approach to mine Bayesian networks from distributed heterogeneous web log data streams.

Reference [11] identifies some key aspects of stream data mining algorithms and outlines a number of possible directions for future research.

2.2 Related Work on Clustering Data Streams

References [12-15] consider clustering in the data stream model; they extend classical clustering algorithms, such as *k*-median and *k*-means to data stream literature, by assuming that the data objects arrive as chunks. Specially, in [12-14], a LOCALSEARCH subroutine is performed twice every time a new chunk arrives: first on the new chunk of point to generate cluster centers and then on the set of cluster centers of all observed chunks produced by

LOCALSEARCH to locate the overall cluster centers. It has been proved that this two-phase algorithm produce a good approximation to the optimum clustering and is memory efficient. A new algorithm, namely, VFKM is proposed in [15]. It extends the *k*-means clustering algorithm by bounding the learner's loss as a function of the number of examples used at each step.

In [16], the authors developed an efficient method, called CluStream, for clustering large evolving data streams. Instead of trying to cluster the whole stream at one time, the method view the stream as a changing process over time. The CluStream model provides a wide variety of functionality in characterizing data stream clusters over different time horizons in an evolving environment.

In [17], the author addresses the problem of clustering data stream with increasing dimensionality. That is, a data stream has *k* values (dimensions) at the *k*th snapshot while the *k*+1 values at the (*k*+1)th snapshot. A weighted distance metric between two streams is applied and an incremental clustering algorithm is developed to produce clusters of streams.

2.3 Related Work on Clustering Categorical Data

A few algorithms have been proposed in recent years for clustering categorical data [18~38]. In [18], the problem of clustering customer transactions in a market database is addressed. STIRR, an iterative algorithm based on non-linear dynamical systems is presented in [19]. The approach used in [19] can be mapped to a certain type of non-linear systems. If the dynamical system converges, the categorical databases can be clustered. Another recent research [20] shows that the known dynamical systems cannot guarantee convergence, and proposes a revised dynamical system in which convergence can be guaranteed.

K-modes, an algorithm extending the *k*-means paradigm to categorical domain is introduced in [21,22]. New dissimilarity measures to deal with categorical data is conducted to replace means with modes, and a frequency based method is used to update modes in the clustering process to minimize the clustering cost function. Based on *k*-modes algorithm, [23] proposes an adapted mixture model for categorical data, which gives a probabilistic interpretation of the criterion optimized by the *k*-modes algorithm. A fuzzy *k*-modes algorithm is presented in [24] and tabu search technique is applied in [25] to improve fuzzy *k*-modes algorithm. An iterative initial-points refinement algorithm for categorical data is presented in [26]. The work in [36] can be considered as the extensions of *k*-modes algorithm to transaction domain.

In [27], the authors introduce a novel formalization of a cluster for categorical data by generalizing a definition of cluster for numerical data. A fast summarization based algorithm, CACTUS, is presented. CACTUS consists of three phases: *summarization*, *clustering*, and *validation*.

ROCK, an adaptation of an agglomerative hierarchical clustering algorithm, is introduced in [28]. This algorithm starts by assigning each tuple to a separated cluster, and then clusters are merged repeatedly according to the closeness between clusters. The closeness between clusters is defined as the sum of the number of "links" between all pairs of tuples, where the number of "links" is computed as the number of common neighbors between two tuples.

In [29], the authors propose the notion of *large item*. An item is *large* in a cluster of transactions if it is contained in a user specified fraction of transactions in that cluster. An allocation and refinement strategy, which has been adopted in partitioning algorithms such as *k*-means, is used to cluster transactions by minimizing the criteria function defined with the notion of large item. Following the large item method in [29], a new measurement, called the small-large ratio is proposed and utilized to perform the clustering [30]. In [31], the authors consider the item taxonomy in performing cluster analysis. While the work [32] proposes a algorithm based on "caucus", which is fine-partitioned demographic groups which is based the purchase features of customers.

Squeezer, a one-pass algorithm is proposed in [33]. *Squeezer* repeatedly read tuples from dataset one by one. When the first tuple arrives, it forms a cluster alone. The consequent tuples are either put into an existing cluster or rejected by all existing clusters to form a new cluster by given similarity function.

COOLCAT, an entropy-based algorithm for categorical clustering, is proposed in [34]. Starting from a heuristic method of increasing the height-to-width ratio of the cluster histogram, the authors in [35] develop the CLOPE algorithm. [37] introduce a distance measure between partitions based on the notion of generalized conditional entropy and a genetic algorithm approach is utilized for discovering the median partition. In [38], the authors formally define the categorical data clustering problem as an optimization problem from the viewpoint of cluster ensemble, and apply cluster ensemble approach for clustering categorical data.

3. Notations

Let $A_1, \ldots, A_m$ be a set of categorical attributes with domains $D_1, \ldots, D_m$ respectively. Let the dataset $\boldsymbol{D} = \{X_1, X_2, \ldots, X_n\}$ be a set of objects described by m categorical attributes, $A_1, \ldots, A_m$. The value set V_i of A_i is set of values of A_i that are present in $\boldsymbol{D}$. For each $v \in V_i$, the frequency f(v), denoted as f_v, is number of objects $O \in \boldsymbol{X}$ with $O.\ A_i = v$. Suppose the number of distinct attribute values of A_i is p_i, we define the histogram of A_i as the set of pairs: $h_i = \{(v_1, f_1), (v_2, f_2), \ldots, (v_{p_i}, f_{p_i})\}$. The histogram of the data set $\boldsymbol{D}$ is defined as: $H = \{h_1, h_2, \ldots, h_m\}$.

Let X, Y be two categorical objects described by m categorical attributes. The dissimilarity measure between X and Y can be defined by the total mismatches of the corresponding attribute values of the two objects. The smaller the number of mismatches is, the more similar the two objects. Formally,

$$d_1(X,Y) = \sum_{j=1}^{m} \delta(x_j, y_j) \tag{1}$$

where

$$\delta(x_j, y_j) = \begin{cases} 0 & (x_j = y_j) \\ 1 & (x_j \neq y_j) \end{cases} \tag{2}$$

Given the dataset $\boldsymbol{D} = \{X_1, X_2, \ldots, X_n\}$ and an object Y, The dissimilarity measure between $\boldsymbol{X}$ and Y can be defined by the average of the sum of the distances between X_i and Y.

$$d_2(D,Y) = \frac{\sum_{j=1}^{n} d_1(X_j, Y)}{n} \tag{3}$$

If we take the histogram $H = \{h_1, h_2, \ldots, h_m\}$ as the compact representation of the data set ***D***, formula (3) can be redefined as (4).

$$d_3(H,Y) = \frac{\sum_{j=1}^{m} \phi(h_j, y_j)}{n} \tag{4}$$

where

$$\phi(h_j, y_j) = \sum_{l=1}^{p_j} f_l * \delta(v_l, y_j) \tag{5}$$

From a viewpoint of implementation efficiency, formula (4) can be presented in form of (6).

$$d_4(H,Y) = \frac{\sum_{j=1}^{m} \psi(h_j, y_j)}{n} \tag{6}$$

where

$$\psi(h_j, y_j) = \sum_{l=1}^{p_j} f_l * (1 - \delta(v_l, y_j)) \tag{7}$$

Formula (6) can be computed more efficiently for which requires only the frequencies of matched attribute value pairs.

4. A Stream Clustering Algorithm for Categorical Data

Squeezer, a one-pass clustering algorithm for categorical data is presented in [33]. The algorithm has *n* tuples as input and produce clusters as final results. Initially, the first tuple in the database is read in and a new histogram is created. The consequent tuples are read iteratively.

For every tuple, according to (6), we compute its similarities with all existing clusters, which are represented as the corresponding histograms. The largest value of similarity is selected out. If it is larger than the given threshold, donated as *s*, the tuple will be put into the cluster that has the largest value of similarity. The histogram is also updated with the new tuple. If the above condition does not hold, a new cluster (represented by a histogram) must be created with this tuple. The algorithm continues until we have traversed all the tuples in the dataset.

During the clustering process of *Squeezer*, our main task is to maintain and update histograms. However, data stream algorithms must not have large space requirements and, so, our first goal will be to make modifications so that the clustering can be carried out in a small space. Therefore, instead of maintaining the all the value frequencies in each histogram, we apply approximation counts technique over data streams developed by Manku and Motwani [5] to keep only value frequencies that are relatively "large". In the sequel, we will briefly introduce their method (Lossy Counting Algorithm).

The lossy counting algorithm [5] accepts two user-specified parameters: a support threshold $s \in (0,1)$ and an error parameter $\varepsilon \in (0,1)$ such that ε <<*s*. Let *N* donate the current length of the stream, i.e., the number of tuples seen so far.

The incoming stream is conceptually divided into *buckets* of width $\left\lceil \frac{1}{\varepsilon} \right\rceil$ transactions each. Buckets are labeled with *bucket ids*, starting from 1. The *current bucket id* is donated by $b_{current}$, whose value is $\left\lceil \frac{N}{\varepsilon} \right\rceil$. For an element *e*, its true frequency in the stream seen so far is donated by f_e. The data structure *D* is a set of entries of the form (*e*, *f*, Δ), where *e* is an element in the stream, *f* is an integer representing its estimated frequency, and Δ is the maximum possible error in *f*.

Initially, *D* is empty. Whenever a new element *e* arrives, we first look up *D* to see weather an entry for *e* already exists or not. If the lookup succeeds, the entry is updated by incrementing its frequency *f* by one. Otherwise, a new entry of the form (*e*, 1, $b_{current}$-1) is created. Also, *D* is pruned by deleting some of its entries at bucket boundaries, i.e., whenever $N \equiv 0$ mod *w*. The rule for deletion is: an entry (*e*, *f*, Δ) is deleted if $f + \Delta \le b_{current}$. When a user requests a list of items with threshold *s*, outputting those entries with $f \ge (s-\varepsilon)N$.

Theoretical analysis [5] shows that answers produced by the lossy counting algorithm will have the following guarantees:

1. All itemsets whose true frequency exceeds *sN* are output. There are *no false negatives*.
2. No itemset whose true frequency is less than $(s-\varepsilon)N$ is output.
3. Estimated frequencies are less than the true frequencies by at most εN.

The new streaming clustering algorithm, namely StreamCluCD is described in Fig. 1. The key steps in this algorithm are Step (04) and Step (11). In Step (04), according to (6), we compute the similarity between a tuple and a cluster (represented by a histogram). Notice that the histogram maybe has been pruned in Step (11). In Step (11), the histogram is pruned by deleting some of its entries at bucket boundaries according to [5].

Next, we will show that the StreamCluCD algorithm has provable space guarantees.

Theorem 1 [5]: Lossy Counting computes an ε-deficient synopsis using at most $\frac{1}{\varepsilon}\log(\varepsilon N)$ entries, where *N* donate the current length of the stream, i.e., the number of tuples seen so far.

In our literature, theorem 1 implies that each histogram will be space-bounded.

Straightforwardly, we can get theorem 2.

```
Algorithm StreamCluCD

Input:   DS              // the categorical data stream
         minisupport     // user defined threshold for the permissible minimal support
         $\varepsilon$               // user defined error parameter for frequent pattern
         s               // user defined threshold value
Output:  The cluster descriptions in form of histograms so far

Begin
foreach   tuple t in DS do begin
     foreach existed cluster C
          computing the similarity between t and C
     end
     getting the max value of similarity: sim_max
     getting the corresponding cluster : c
     if sim_max >= s then
        update entries of the form (e, f, maximal error ) in cluster c
        if t pruning condition is satisfied then
             deleting some of its entries at bucket boundaries
        end
     end
     else
          creating a new cluster with t (creating a new histogram)
     end
end
```

Fig. 1 The streaming clustering algorithm for categorical data

Theorem 2: StreamCluCD algorithm uses at most $\frac{1}{\varepsilon}\log(\varepsilon^k \prod_{i=1}^{k} N_i)$ entries, where N_i donates the current length of the cluster *i* and *k* is the current number of clusters.

Proof: From theorem 1, we know for each cluster *i*, the space needed is at most $\frac{1}{\varepsilon}\log(\varepsilon N_i)$. Thus, StreamCluCD algorithm uses at most

$$\sum_{i=1}^{k}\frac{1}{\varepsilon}\log(\varepsilon N_i) = \frac{1}{\varepsilon}\sum_{i=1}^{k}\log(\varepsilon N_i) = \frac{1}{\varepsilon}\log(\varepsilon^k \prod_{i=1}^{k} N_i) \qquad \square$$

5. Experimental Results

A comprehensive performance study has been conducted to evaluate our method. In this section, we describe those experiments and their results. The use of real-life datasets is to compare the quality of the clustering results produced by our streaming clustering algorithm, and the synthetic datasets are used to primarily demonstrate the scalability of our algorithm.

Our algorithms were implemented in Java. All experiments were conducted on a Pentium4−2. 4G machine with 512 M of RAM and running Windows 2000.

5.1 Real Life Dataset

We experimented with a real-life dataset: the Mushroom dataset, which was obtained from the UCI Machine Learning Repository [39]. Now we will give a brief introduction about it.

The mushroom dataset has 22 attributes with 8124 tuples. Each tuple records physical characteristics of a single mushroom. A classification label of poisonous or edible is provided with each tuple. The numbers of edible and poisonous mushrooms in the dataset are 4208 and 3916, respectively.

In addition, the clustering accuracy for measuring the clustering results is computed as the follows. Suppose the final number of clusters is *k*, clustering accuracy *r* is defined as: $r = \frac{\sum_{i=1}^{k} a_i}{n}$, where *n* is number of instances in the dataset, a_i is number of instances occurring in both cluster *i* and its corresponding class, which has the maximal value. In other words, a_i is the number of instances with class label that dominate cluster *i*. Consequently, the clustering error is defined as $e = 1-r$.

5.2 Clustering Results

It has been demonstrated that the *Squeezer* algorithm [33] can produce better clustering output than other algorithms in categorical dataset with respect to clustering accuracies. In addition, it is a one-pass clustering algorithm for categorical data. Thus, this section is devoted to demonstrate the effectiveness of the streaming clustering algorithm-StreamCluCD compared to *Squeezer* algorithm.

We used the StreamCluCD and *Squeezer* algorithms to cluster the *mushroom* dataset into different numbers of clusters with fixed threshold value of similarity, varying from 7 to 16. For each fixed threshold value of similarity, the clustering errors of different algorithms were compared. In this experiment, the error parameter ε is set to 0.001.

Fig.2 shows the results on the mushroom dataset of different clustering algorithms. From Fig.2, we can summarize the relative performance of our algorithms as follows (See Table 1):

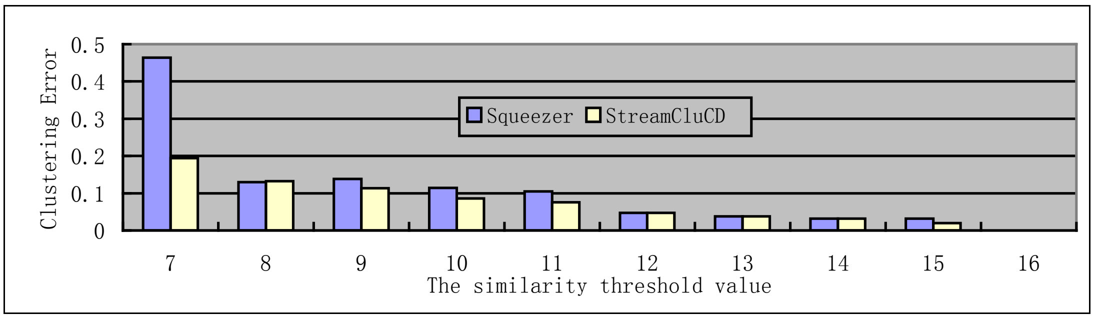

Fig.2. Clustering results on mushroom dataset

Table 1: Relative performance of different clustering algorithms

Ranking	**1**	**2**	**Average Clustering Error**
Squeezer	5	**5**	0.110
StreamCluCD	**9**	1	**0.074**

That is, comparing to the *Squeezer* algorithm, the StreamCluCD algorithm performs the best for 9 cases and the worst in 1 case. Furthermore, the average clustering error of the StreamCluCD algorithm is significantly smaller than that of the *Squeezer* algorithm.

One important observation from the Fig.2 and Table 1 is that the clustering accuracy of StreamCluCD algorithm is relatively good when we apply approximation counts technique over data streams. It indicates that our algorithm can produce good clustering output with limited memory in data stream environment.

Furthermore, Ref [33] shows that the choice of *s* (the similarity threshold) can affect both the quality of clustering and execution time of *Squeezer*-like algorithms. As shown in Fig.3, with the increase of *s*, the number of clusters produced by both algorithms increases. While the variance of the numbers of clusters in StreamCluCD algorithm relatively is smaller, which means the clusters formed in the clustering process of StreamCluCD algorithm are more stable and less affected by parameters.

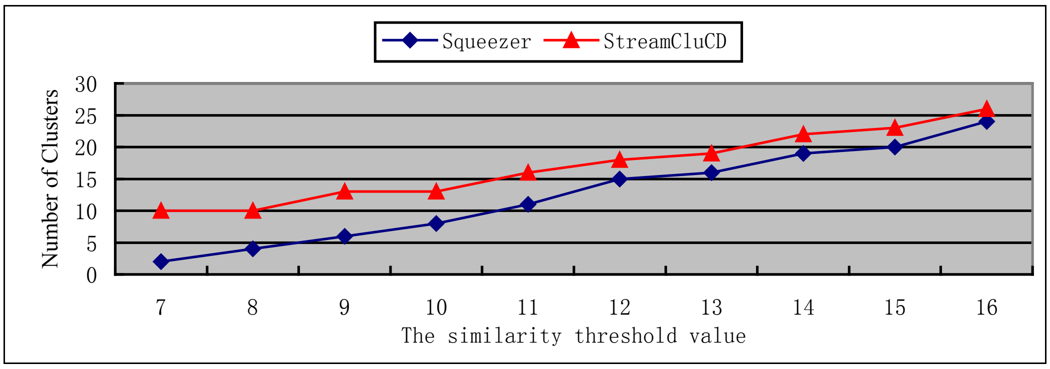

Fig.3. Number of clusters vs. Different similarity threshold values

5.3 Scalability Test

For the running time, experiments were carried out with synthetic datasets. To find out how the number of tuples affects the 4 algorithms, we ran a series of experiments with increasing number of tuples. The datasets were generated using a data generator, in which all possible values are produced with (approximately) equal probability. We set the number of tuples to 1 million, the number of attributes to 10 and the number of attribute values for each attribute to 10. Due to sparseness of generated datasets, we set *s* to 1.

Fig.4 shows the scalability of our algorithm while increasing the number of tuples from 1 to 10 million.

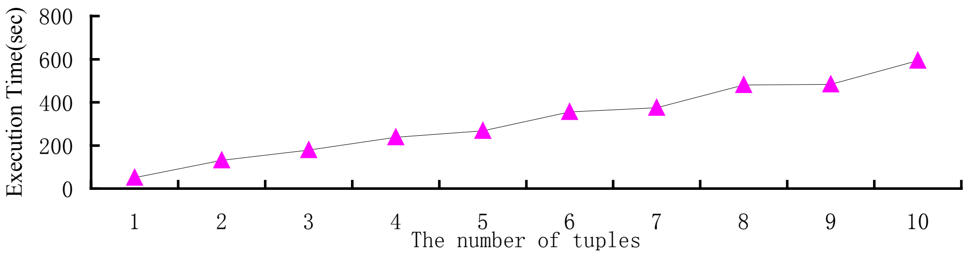

Fig.4: The Execution Time with Different Number of Tuples

To find out how the number of attributes affects these algorithms, we ran a series of experiments with increasing number of attributes. In Fig.5, the number of attributes is increased from 10 to 100 and the number of tuples is fixed to 100,000.

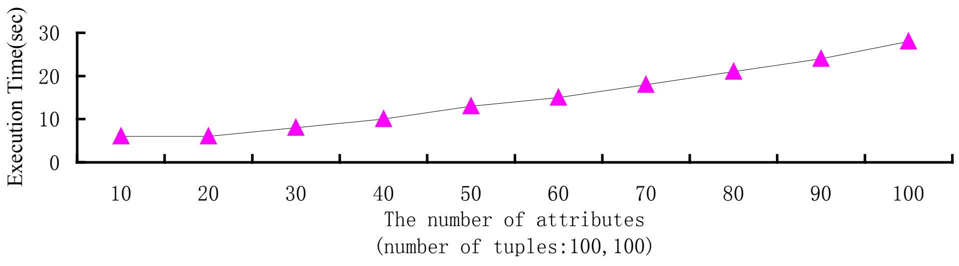

Fig.5: The Execution Time with Different Number of Attributes

The above experimental results demonstrate the scalability of StreamCluCD algorithm with respect to both the size of dataset and the number of dimensions. It qualifies the StreamCluCD algorithm for clustering high-speed evolving data streams.

6. Conclusions

A wide spectrum of clustering methods has been developed in data mining, statistics, and machine learning with many applications. Although very few have been examined in the context of stream data clustering, to our knowledge, we are the first to consider the problem of clustering categorical data streams. We present an effective categorical data stream-clustering algorithm, namely StreamCluCD. The proposed algorithm has provably small memory footprints. We also provide empirical evidence of the algorithm's performance on real datasets and synthetic data streams.

As future work, we are going to apply the StreamCluCD algorithm for detecting cluster based local outliers[40] and semantic outliers[41] in data stream environment.

References

[1] P. Domingos and G. Hulton. Mining high-speed data streams. In: Proc of KDD'00, 2000: 71~80.

[2] G. Hulton, L. Spencer and P. Domingos. Mining time-changing data streams. In: Proc of KDD'01, 2001: 97~106.

[3] W. N. Street, Y. S. Kim. A streaming ensemble algorithm (SEA) for large scale classification. In: Proc of KDD'01, 2001: 377-382.

[4] M. Last. Online classification of non-stationary data streams. Intelligent Data Analysis, 2002, 6(2): 129-147.

[5] G. S. Manku, R. Motwani. Approximate frequency counts over data streams. In: Proc of VLDB'02, 2002:346-357.

[6] Y. Chen, G. Dong, J. Han, B. W. Wah, J. Wang. Multi-dimensional regression analysis of time-series data streams. In: Proc of VLDB'02, 2002: 323-334.

[7] D. Lambert, J. C. Pinheiro. Mining a stream of transactions for customer patterns. In: Proc of KDD'01, 2001:305-310.

[8] C. Cortes, K. Fisher, D. Pregibon, A. Rogers. Hancock: a language for extracting signatures from data streams. In: Proc of KDD'00, 2000:9-17.

[9] B. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, A. Biliris. Online data mining for co-evolving time sequences. In: Proc ICDE'00, 2000: 13-22.

[10] R. Chen, K. Sivakumar, H. Kargupta. An approach to online Bayesian learning from multiple data streams. In: 2001 PKDD Workshop on Ubiquitous Data Mining for Mobile and Distributed Environments, 2001.

[11] P. Domingos, G. Hulton. Catching up with the data: research issues in mining data streams. In: 2001 SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2001.

[12] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan. Clustering Data Streams. In: Proc of FOCS'00, 2000: 359-366.

[13] L. O. Chalaghan, N. Mishra, A. Meyerson, S. Guha. Streaming-data algorithms for high-quality clustering. In: Proc of ICDE'02, 2002: 685-694.

[14] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan. Clustering data streams: theory and practice. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(3): 515-528.

[15] P. Domingos, G. Hulton. A general method for scaling up machine learning algorithms and its application to clustering. In: Proc of ICML'01, 2001:106-113.

[16] C.C. Aggarwal, J. Han, J. Wang, P. S. Yu. A framework for clustering evolving data streams. In: Proc of VLDB'03, 2003.

[17] J. Yang. Dynamic clustering of evolving streams with a single pass. In: Proc of ICDE'03, 2003.

[18] E.H. Han, G. Karypis, V. Kumar, B. Mobasher: Clustering based on association rule hypergraphs. In: SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, 1997.

[19] D. Gibson, J. Kleiberg, P. Raghavan. Clustering categorical data: an approach based on dynamic systems. In: Proc of VLDB'98, pp. 311-323, 1998.

[20] Y. Zhang, A. W. Fu, C. H. Cai, P. A. Heng. Clustering Categorical Data. In: Proc of ICDE'00, 2000.

[21] Z. Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. In: SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, 1997.

[22] Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Mining and Knowledge Discovery, 1998, 2(3): 283-304.

[23] F. Jollois, M. Nadif. Clustering large categorical data. In: Proc of PAKDD'02, pp. 257-263, 2002

[24] Z. Huang, M. K. Ng. A fuzzy k-modes algorithm for clustering categorical data. IEEE Transaction on Fuzzy Systems, 1999, 7(4): 446-452.

[25] M. K. Ng, J. C. Wong. Clustering categorical data sets using tabu search techniques. Pattern Recognition, 2002, 35(12): 2783-2790.

[26] Y. Sun, Q. Zhu, Z. Chen. An iterative initial-points refinement algorithm for categorical data clustering. Pattern Recognition Letters, 2002, 23(7): 875-884.

[27] V. Ganti, J. Gehrke, R. Ramakrishnan. CACTUS-clustering categorical data using summaries. In: Proc of KDD'99, pp. 73-83, 1999.

[28] S. Guha, R. Rastogi, K. Shim. ROCK: a robust clustering algorithm for categorical attributes. In: Proc of ICDE'99, pp. 512-521, 1999.

[29] K. Wang, C. Xu, B. Liu. Clustering transactions using large items. In: Proc of CIKM'99, pp. 483-490, 1999.

[30] C. H. Yun, K. T. Chuang, M. S. Chen. An efficient clustering algorithm for market basket data based on small large ratios. In: Proc of COMPSAC'01, pp. 505-510, 2001.

[31] C. H. Yun, K. T. Chuang, M. S. Chen. Using category based adherence to cluster market-basket data. In: Proc of ICDM'02, 2002.

[32] J. Xu, S. Y. Sung. Caucus-based transaction clustering. In: Proc of DASFAA'03, pp. 81-88, 2003.

[33] Z. He, X. Xu, S. Deng: *Squeezer:* an efficient algorithm for clustering categorical data. Journal of Computer Science & Technology, 2002,17(5): 611-624.

[34] D. Barbara, Y. Li, J. Couto. COOLCAT: an entropy-based algorithm for categorical clustering. In: Proc of CIKM'02, pp. 582-589, 2002.

[35] Y. Yang, S. Guan, J. You. CLOPE: a fast and effective clustering algorithm for transactional data. In: Proc of KDD'02, 2002.

[36] F. Giannotti, G. Gozzi, G. Manco. Clustering transactional data. In: Proc of PKDD'02, pp. 175-187, 2002.

[37] D. Cristofor, D. Simovici. Finding median partitions using information-theoretical-based genetic algorithms. Journal of Universal Computer Science, 2002, 8(2): 153-172.

[38] Z. He, X. Xu, S. Deng. Clustering categorical data: an cluster ensemble method. In: Proc of ICYCS'03, 2003.

[39] C. J. Merz, P. Merphy. UCI Repository of Machine Learning Databases, 1996. (Http://www.ics.uci.edu/~mlearn/MLRRepository.html).

[40] Z. He, X. Xu, S. Deng. Discovering cluster based local outliers. Pattern Recognition Letters, 2003, 24(9-10): 1651-1660.

[41] Z. He, S. Deng, X. Xu. Outlier detection integrating semantic knowledge. In: *Proc. of WAIM'02*, 2002:126-131.