The VLDB Journal (2009) 18:1241–1260
DOI 10.1007/s00778-009-0134-5

REGULAR PAPER

HE-Tree: a framework for detecting changes in clustering structure for categorical data streams

Keke Chen · **Ling Liu**

Received: 28 June 2007 / Revised: 29 January 2009 / Accepted: 29 January 2009 / Published online: 28 February 2009

Abstract Analyzing clustering structures in data streams can provide critical information for real-time decision making. Most research in this area has focused on clustering algorithms for *numerical* data streams, and very few have proposed to monitor the change of clustering structure. Most surprisingly, to our knowledge, no work has been proposed on monitoring clustering structure for *categorical* data streams. In this paper, we present a framework for detecting the change of primary clustering structure in categorical data streams, which is indicated by the change of *the best number of clusters* (Best K) in the data stream. The framework uses a *Hierarchical Entropy Tree* structure (HE-Tree) to capture the entropy characteristics of clusters in a data stream, and detects the change of Best K by combining our previously developed BKPlot method. The HE-Tree can efficiently summarize the entropy property of a categorical data stream and allow us to draw precise clustering information from the data stream for generating high-quality BKPlots. We also develop the time-decaying HE-Tree structure to make the monitoring more sensitive to recent changes of clustering structure. The experimental result shows that with the combination of the HE-Tree and the BKPlot method we are able to promptly and precisely detect the change of clustering structure in categorical data streams.

Keywords Data stream mining · Categorical data clustering · Change detection

K. Chen (✉)
Department of Computer Science and Engineering,
Wright State University, Dayton, USA
e-mail: keke.chen@wright.edu

L. Liu
College of Computing, Georgia Institute of Technology,
Atlanta, USA

1 Introduction

With the wide deployment of sensor systems and Internet-based continuous-query applications, the scalability of data mining methods is constantly being challenged by a tremendous amount of data generated at unprecedented rates. Examples of such data streams include network event logs, telephone call records, credit card transactional flows, and blog discussions, etc. As an important method in data analysis, recently clustering data streams has become a research topic of growing interest [2,27]. The initial research has shown that clustering stream data can provide important clues about emerging data patterns, so that decision makers can predict new coming events and react promptly. As many data streams also include categorical data, discretizing the numerical part and unifying both types of data under the categorical data clustering framework [5,13,34] is one convenient method. Surprisingly, very few [5] have addressed the problems of clustering categorical data streams, and none addressed the problems of monitoring the change of clustering structure in categorical data streams.

Other than its huge data volume, data streams also have the unique evolving nature. In particular, the change of clustering patterns often indicates something important is happening. For example, clustering network event streams can help us understand the normal patterns, and attack alarms can be raised if the clustering pattern changes. Unfortunately, many data stream clustering algorithms use certain fixed parameters, which do not address the evolving nature. Specifically, they often assume a fixed number of clusters in the data stream, such as the K-Median algorithm for numerical data streams [27] and the Coolcat [5] algorithm for categorical data streams, which is certainly not right for an evolving data stream. We argue that monitoring the change in the critical clustering structure, particularly, the change of best K

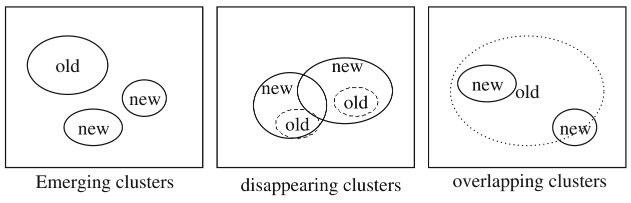


Fig. 1 Challenges in detecting the change of clustering structure

number of clusters in data streams, is one of the most important tasks in categorical data stream mining.

The change of critical clustering structure in data streams involves three major aspects: drifting of the cluster center, new emerging clusters, and disappearing clusters. Clusters often keep growing if the historical members are not discarded. The growing clusters may change cluster centers. When clusters grow to certain extent, they may merge and the previous clustering structure disappears. We notice that emerging clusters and disappearing clusters can be possibly indicated by the "Best K" number of clusters. By monitoring the change of best K number of clusters, we will know whether the underlying critical clustering structure is changing. In this paper, we design the HE-Tree based algorithms to make it possible to monitor the change of best K number of clusters in categorical data streams.

Briefly, we want to design a summarization structure, which captures the clustering characteristics of the data stream. There are two outstanding challenges in this design: (1) In order to make time-critical detection of the changes, this structure must be able to swiftly adapt to the change of the clustering structure, i.e., emerging and disappearing of clusters; (2) To capture up-to-date or transient clustering patterns in the stream, this structure must also be able to efficiently discount the effects of historical examples. Without this discounting function, a so-called cluster overlapping problem (Fig. 1) may prevent prompt detection of new cluster patterns that happen on the historical large cluster.

To address the first challenge, we design a summarization tree structure, called the Hierarchical Entropy Tree (HE-Tree for short). The HE-Tree is able to utilize a small amount of memory to efficiently summarize the cluster entropy characteristics of the data stream and group the data records into a bunch of coarse clusters, which are saved at HE-Tree leaf nodes. The working mechanism makes it possible to swiftly adapt to the change of the clustering structure. The subclusters (often a few hundreds) at HE-Tree leaf nodes can be easily dumped as a snapshot. An extended ACE algorithm [13] is designed to work on the snapshot and generate an approximate BKPlot at certain time interval. The difference between clustering structures can be conveniently identified by comparing these snapshot BKPlots.

To avoid stocked historical data records interfering the detection of transient changes, we also design a time-decaying HE-Tree. To use the time-decaying HE-Tree, the data stream has to be processed in windows, and each window consists of time units that is the time interval a BKPlot can be generated. The old HE-Tree from the last window is inherited and proportionally discounted to adapt to the possible changes happening in this window. Experimental results show that the HE-Tree can effectively detect the change of clustering structure and the decaying structure can efficiently discount the effect of the historical data.

The rest of the paper is organized as follows. Section 2 sets down notations and basic concepts in entropy-based categorical clustering. Section 3 briefly introduces the BKPlot method for finding the best K in static categorical datasets. In Sect. 4, we develop the HE-Tree structure and describe its working mechanism. In Sect. 5, we propose the framework for detecting the change of clustering structure in categorical data streams. Experimental results are shown in Sect. 6 to address several questions. Finally, we review some related work of categorical clustering and data stream mining.

2 Entropy-based categorical clustering

Clustering techniques for categorical data are very different from those for numerical data, mainly because of the definition of similarity measure. Most numerical clustering techniques have been using distance functions, for example, Euclidean distance, to define the similarity measure. However, there is no such inherent distance meaning between categorical values.

In contrast to distance-based similarity measures for pairs of data records, similarity measures based on the "purity" of a bulk of records seem more intuitive for categorical data. Entropy [18] is a well defined concept measuring the purity of a dataset. Originated from information theory, entropy has been applied in various areas, such as pattern discovery [8], numerical clustering [17] and information retrieval [40]. Due to the lack of intuitive distance definition for categorical values, recently entropy has been applied in clustering categorical data [5,11,13,20,34]. Initial results have shown that the entropy criterion can be very effective in clustering categorical data. Paper [34] also shows that the entropy criterion can be formally derived in the framework of probabilistic clustering models, which strongly supports that the entropy criterion is a meaningful and reliable similarity measure, particularly good for categorical data.

In entropy-based categorical clustering, the quality of clustering is essentially evaluated by the entropy criterion, namely, the *Expected Entropy* of clusters [5,34]. Other variants, such as minimum description length (MDL) [11] or mutual information [3,20], can also be unified under the probabilistic clustering framework [34]. We categorize all these

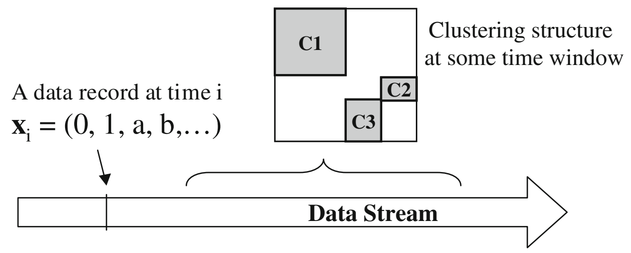

Clustering structure at some time window
C1
C2
C3
A data record at time i
$\mathbf{x}_i$ = (0, 1, a, b,…)
Data Stream

Fig. 2 A multi-dimensional categorical data record in the data stream and a clustering pattern at some time window

approaches as entropy-based categorical clustering. The main goal of these algorithms is to find a partition that minimizes the expected entropy for K clusters, if K is known. However, this problem, similar to the pairwise distance-based clustering problem, is computationally intractable even for a dataset of median size. A common approach to solving this problem is approximation. Typically, in approximation algorithms we have to sacrifice some optimality to obtain reasonable computational cost. Apparently, stream data makes the balance between the optimality and the computational efficiency even harder.

Figure 2 shows the basic setting of clustering categorical a data stream. Below, we first give the notations and definitions used in this paper, and then describe an important measure, namely *Incremental Entropy* (IE), which is a key measure used to approximately describe clustering relationships in our algorithms.

2.1 Notations and definitions

Consider that a dataset (data stream) $\mathbb{S}$ with N records and d columns, is a sample set of the discrete random vector $X = (x_1, x_2, \ldots, x_d)$. For each component x_j, $1 \le j \le d$, x_j takes a value from the domain A_j. There are a finite number of distinct categorical values in domain A_j and we denote the number of distinct values as $|A_j|$. The *total cardinality* $m = \sum_{j=1}^{d} |A_j|$ will be an important factor in this paper. Let $p(x_j = v)$, $v \in A_j$, represent the probability of $x_j = v$. We introduce the classical entropy definition [18].

$$H(X) = \sum_{j=1}^{d} H(x_j) = -\sum_{j=1}^{d} \sum_{v \in A_j} p(x_j = v) \log_2 p(x_j = v)$$

Here, entropy $H(X)$ is based on the column entropy $H(x_j)$, while the correlations between columns are ignored for easy manipulation without affecting the results [5,34]. $H(X)$ is often estimated with the sample set $\mathbb{S}$, we define the estimated entropy as $H(X \mid \mathbb{S})$.

$$H(X \mid \mathbb{S}) = -\sum_{j=1}^{d} \sum_{v \in A_j} p(v \mid \mathbb{S}) \log_2 p(v \mid \mathbb{S})$$

where $p(v \mid \mathbb{S})$ is the empirical probability estimated on $\mathbb{S}$. To further simplify the notation, we also define the *column entropy* of A_j as

$$H(A_j \mid \mathbb{S}) = -\sum_{v \in A_j} p(v \mid \mathbb{S}) \log_2 p(v \mid \mathbb{S})$$

The estimated entropy on $\mathbb{S}$ is simply the sum of the column entropies.

Now we can define the concept of cluster entropy. Suppose the dataset $\mathbb{S}$ is partitioned into K clusters. Let $C^K = \{C_1, \ldots, C_K\}$ represent the partition, where C_k is a cluster and n_k represent the number of records in C_k. Thus, the *cluster entropy* of C_k is the dataset entropy $H(X \mid C_k)$. For simpler presentation, we use $H(\mathbb{S})$ and $H(C_k)$ to represent the dataset entropy and cluster entropy, respectively.

The classical entropy-based clustering criterion tries to find the optimal partition, C^K, which maximizes the following entropy criterion [7,9,34].

$$\text{Opt}(C^K) = \frac{1}{d}\left(H(\mathbb{S}) - \frac{1}{n}\sum_{k=1}^{K} n_k H(C_k)\right)$$

Since $H(\mathbb{S})$ is fixed for the given dataset, maximizing $\text{Opt}(C^K)$ is equivalent to minimizing the item $\frac{1}{n}\sum_{k=1}^{K} n_k H(C_k)$, which is named as the *expected entropy* of partition C^K. Let us denote it as $\bar{H}(X \mid C^K)$, or simply $\bar{H}(C^K)$. For convenience, we also name $n_k H(C_k)$ as the *weighted entropy* of cluster C_k.

Intuitively, the expected entropy describes the overall disorderliness in each cluster. The purer the clusters are, the higher quality the clustering result is. While the entropy criterion can also be applied to numerical data [17] if the numerical data is appropriately discretized, it loses the ability of describing most of the nice geometric features for numerical data [12].

2.2 Incremental entropy

Individually, cluster entropy cannot determine the structural difference between clusters. However, we observe that the structural difference can be observed by mixing (merging) two clusters. By the entropy definition, the structural characteristic of a dataset is determined by the value frequencies in each column. Intuitively, mixing two clusters that are similar in the inherent structure will not change the value frequencies, thus, will not change the expected entropy of the partition as well. However, merging dissimilar ones will inevitably change the value frequencies, increasing the expected entropy. Therefore, the increase of the expected entropy in merging clusters has some correlation with the similarity between clusters.

By the definition of expected entropy, after merging two clusters in a partition the difference in expected entropy can be equivalently evaluated by the difference between the weighted entropies, i.e., $(n_p + n_q)H(C_p \cup C_q)$ and $n_p H(C_p) + n_q H(C_q)$. We have the following first result about weighted entropies.

Proposition 1 $(n_p+n_q)H(C_p \cup C_q) \geq n_p H(C_p)+n_q H(C_q)$

Proof Sketch This proposition formally states that mixing two clusters will not reduce the weighted entropy. The first step of the proof is to expand both sides of the formula with the entropy definition. Let $p(x_j = v|C_p)$ be the estimated probability of $x_j = v$ in the column A_j within the cluster C_p.

$$\begin{aligned} &-\sum_{j=1}^{d}\sum_{v\in A_j}(n_p+n_q)p(x_j=v|C_p\cup C_q)\\ &\quad\cdot\log_2 p(x_j=v|C_p\cup C_q)\\ &\geq -\sum_{j=1}^{d}\sum_{v\in A_j} n_p p(x_j=v|C_p)\log_2 p(x_j=v|C_p)\\ &-\sum_{j=1}^{d}\sum_{v\in A_j} n_q p(x_j=v|C_q)\log_2 p(x_j=v|C_q) \qquad (1)\end{aligned}$$

It is straightforward to prove that the above formula is true if the following relation is satisfied for each value v in each column A_j. Namely, if we can prove that for each categorical value in each column the following formula is true, then the proposition is established.

$$\begin{aligned} &n_p p(x_j=v|C_p)\log_2 p(x_j=v|C_p)\\ &\quad+n_q p(x_j=v|C_q)\log_2 p(x_j=v|C_q)\\ &\geq(n_p+n_q)p(x_j=v|C_p\cup C_q)\cdot\log_2 p(x_j=v|C_p\cup C_q) \qquad (2)\end{aligned}$$

Without loss of generality, suppose C_p having x rows and C_q having y rows with value v at jth attribute, $x, y > 0$ (if $x = 0$ or $y = 0$, the inequality is trivially satisfied), i.e., $p(x_j = v|C_p) = \frac{x}{n_p}$, $p(x_j = v|C_q) = \frac{x}{n_q}$, and $p(x_j = v|C_p \cup C_q) = \frac{x+y}{n_p+n_q}$. Then, the inequality 2 can be transformed to $x \log_2 \frac{x}{n_p} + y \log_2 \frac{y}{n_q} \geq (x+y)\log_2 \frac{x+y}{n_p+n_q}$, which is exactly the "log-sum inequality" [18]. □

We name $M(C_p, C_q) = (n_p + n_q)\hat{H}(C_p \cup C_q) - (n_p \hat{H}(C_p) + n_q \hat{H}(C_q)) \geq 0$ as the "*Incremental Entropy* (IE)" of merging two clusters C_p and C_q. Note that $M(C_p, C_q) = 0$ suggests that the two clusters have the identical structure—for every categorical value v in any arbitrary attribute x_j, $1 \leq j \leq d$, we have $p(x_j = v|C_p) = p(x_j = v|C_q)$. The larger the $M(C_p, C_q)$ is, the more different the two clusters are. IE also plays an important role in constructing a hierarchical clustering scheme, because of the following corollary.

Corollary 2 *Minimizing the IE measure is equivalent to minimizing the expected entropy criterion in hierarchical clustering.*

Proof Sketch Assume C_p and C_q are selected to merge, which minimizes the expected entropy from a C^{K+1} clustering scheme to a C^K scheme. Then, we want to find

$$\min_{C_p,C_q}\left\{\frac{1}{n}\sum_{k=1}^{K} n_k H(C_k) - \frac{1}{n}\sum_{k=1}^{K+1} n_k H(C_k)\right\}$$

Since in the $K + 1$ clusters only two clusters are selected to merge, while the remaining $K - 1$ clusters are not changed, the above objective function equals to

$$\begin{aligned}&\min_{C_p,C_q}\{(n_p+n_q)H(C_p\cup C_q)-(n_pH(C_p)+n_qH(C_q))\}\\ &\quad=\min_{C_p,C_q} M(C_p,C_q)\end{aligned}$$

□

This property significantly differentiates the IE measure from other entropy-related measures, such as KL divergence [18].

3 BKPlot for determining the "Best K" for categorical clustering

In order to better understand the entire framework for detecting changes in the clustering structure for categorical data streams, we briefly describe the BKPlot method for determining the candidate best K for static datasets. For detailed description and analysis, please refer to the paper [13].

Traditionally, cluster analysis of numerical data uses statistical validity indices that are based on geometry and density distribution to validate the clustering result [29]. A typical index curve consists of the index values for different K number of clusters. Those Ks at the peaks, valleys, or distinguishing "knees" on the index curve, are regarded as candidates of the optimal number of clusters (the best K). The BKPlot method tries to find such kind of index for categorical data clustering.

Let "neighboring partitions" be two clustering results having K and $K+1$ clusters, respectively. The basic idea of the BKPlot is to investigate the entropy difference between any two *optimal* neighboring partitions. Let the expected entropy of the optimal partition be $\bar{H}_{\text{opt}}(C^K) = \min\{\bar{H}_i(C^K)\}$, where i is the index of all possible K-cluster partitions. By definition, the curve of $\bar{H}_{\text{opt}}(C^K)$ is non-increasing, i.e., $\bar{H}_{\text{opt}}(C^K) \geq \bar{H}_{\text{opt}}(C^L)$, for $K < L$,

The first heuristic is to look at the shape of the $\bar{H}_{\text{opt}}(C^K)$ curve (Fig. 3). However, experiments show that it usually has no distinguishing peaks, valley, or knees. Therefore, from this curve we cannot effectively identify the best K.

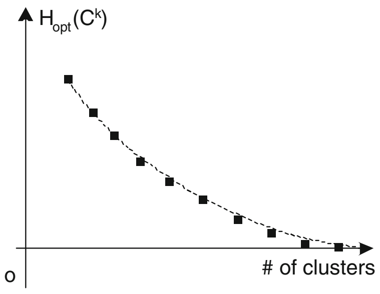


Fig. 3 Sketch of expected entropy curve

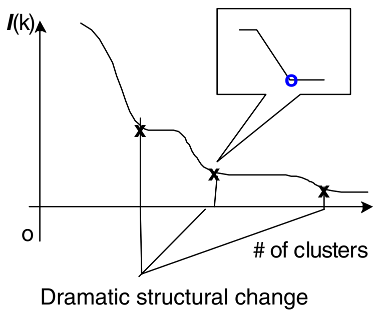


Fig. 4 Sketch of $I(K)$ graph

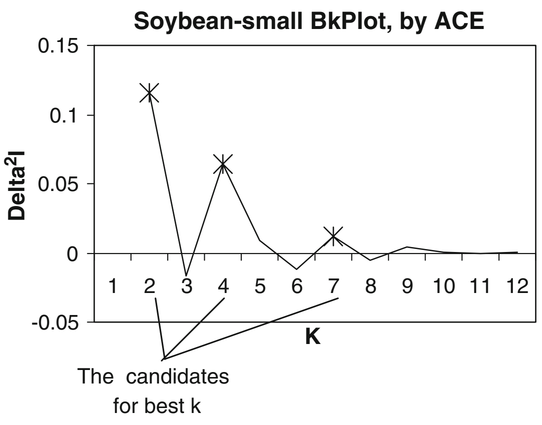


Fig. 5 Finding the best k with BKPlot (for soybean-small dataset)

But we are able to find the special meaning behind the entropy difference of neighboring partitions, which can direct us to the best K. Since the expected entropy is non-decreasing with the decrease in the number of clusters, the increasing rate of expected entropy might be interesting. Let the increasing rate of entropy between the optimal neighboring partitions defined as $I(K) = \bar{H}_{\mathrm{opt}}(C^K) - \bar{H}_{\mathrm{opt}}(C^{K+1})$. We can interpret $I(K)$ with two levels of the difference between neighboring partitions.

- $I(K)$ is the level of difference between two neighboring schemes. The larger the difference is, the more significant the clustering structure is changed by reducing the number of clusters by 1.
- Consider $I(K)$ as the amount of impurity introduced from $K+1$-cluster scheme to K-cluster scheme. If $I(K) \approx I(K+1)$, i.e., the K-cluster scheme introduces a similar amount of impurity as the K+1-cluster scheme does, the change of clustering structure follows a "similar pattern". Thus, we can also consider there is no significant difference from the K+2-cluster partition to the K-cluster partition.

As the above heuristics suggest, we should look at the changing rate, i.e., the differential of the expected entropy curve – the $I(K)$ curve (Fig. 4). At the $I(K)$ curve, we expect that the similar neighboring schemes with different K are at the same "plateau". From plateau to plateau, there are critical points implying significant changes of clustering structure, which could be candidates for the best K.

3.1 Definition of BKPlot

A common way to automatically identify such critical knees on the $I(K)$ curve is to find the peaks/valleys at the second-order difference of the curve. Since an $I(K)$ curve consists of a set of discrete points, we define the second-order difference as $\delta^2 I(K)$:$\delta I(K) = I(K) - I(K+1)$ and $\delta^2 I(K) = \delta I(K-1) - \delta I(K)$ to make K aligned with critical points. These critical points are highlighted at the peak of the second-order difference curve of $I(K)$ (Fig. 5), which is named as "Best-K Plot (BKPlot)".

Exact BKPlots cannot be achieved in practice, since $I(K)$ is based on the optimal K-cluster scheme which involves minimization of the expected entropy. However, since we need only to identify these peak/valley points, approximate BKPlots, which accurately identify the peaks/valleys, are as useful as an exact BKPlot. A hierarchical clustering algorithm ACE in [13] is proposed to generate such high-quality approximate BKPlots, and we have shown in experiments that ACE is a robust method for generating high-quality BKPlots. ACE also has a nice property that we only need to look at the peaks in BKPlots generated by ACE to determine best Ks. In next section, we will give a brief description of ACE algorithm. For further details, please refer to the paper [13].

3.2 A brief description of the ACE algorithm

The ACE algorithm is based on IE. While a traditional hierarchical algorithm needs to explicitly use the inter-cluster similarity like "single-link", "multi-link" or "complete-link" methods [32], IE is a natural inter-cluster similarity measure, ready for constructing a hierarchical clustering algorithm.

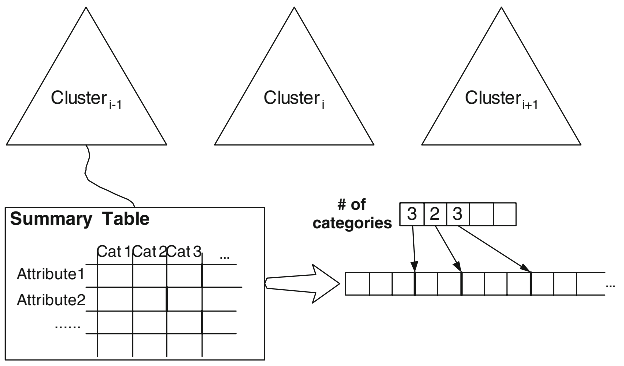


Fig. 6 The summary table and physical structure

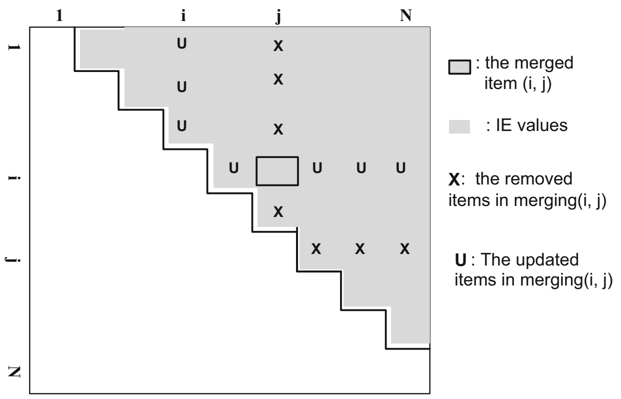

1
i
j
N
U
X
: the merged item (i, j)
: IE values
X: the removed items in merging(i, j)
U : The updated items in merging(i, j)

Fig. 7 The operation schedule after a merging operation

The ACE algorithm is a bottom-up process to construct a clustering tree. It begins with the scenario where each record is a cluster. Then, an iterative process is followed—in each step, the algorithm finds a pair of clusters C_p and C_q that are most similar, i.e., $M(C_p, C_q)$ is minimum among all possible pairs of clusters. We use $M^{(K)}$ to denote the M value in forming the K-cluster partition from the K+1-cluster partition.

Therefore, maintaining minimum IE in each step is one of the major tasks of the algorithm. In order to efficiently implement ACE algorithm, we maintain three data structures: a *summary table* for conveniently counting occurrences of values, an M-*table* for bookkeeping $M(C_p, C_q)$ of any pair of clusters C_p and C_q, and an M-*heap* for maintaining the minimum M value in each step.

The *Summary Table* is used to maintain the fast calculation of the cluster entropy $\hat{H}(C_k)$ and each cluster has one summary table (Fig. 6). Since computing the cluster entropy is based on counting occurrences of each categorical value in each column, we need the summary table to keep counters for each cluster, totally m counters for d columns. Such a summary table enables fast merging operation—when merging two clusters, the two summary tables are added up to form a new summary table for the merged cluster.

The M-*table* is used to keep track of the IE between any pair of clusters, which is then used to maintain the min M in each round of merging. The M-table is a symmetric table (thus, only a half of the entries are used in practice), where the cell (i, j) keeps the value of $M(C_i, C_j)$ (see Fig. 7).

The M-*heap* is used to keep track of the globally minimum IE. We define the most similar cluster to cluster u as $u.similar = \arg\min_v\{M(u, v), v \neq u\}$. Let $u.M$ represent the corresponding IE of merging u and $u.similar$, we define $<u, u.M, u.similar>$ as the *feature vector* of cluster u. Feature vectors are inserted into the heap, sorted by $u.M$, for quickly locating the most similar pair of clusters.

Algorithm 1 shows the sketch of the main procedure. When merging u and $u.similar$, their summary tables are summed up to form the new summary table. Consider u as the main cluster, i.e., $u.similar$ is merged to cluster u, we need to find a new $u.similar$ and insert the new feature vector $<u, u.M, u.similar>$ into the heap. This accounts for the important procedure of updating the bookkeeping information after the merging operation, illustrated by Fig. 7 and detailed in Algorithm 2.

Algorithm 1 ACE.main()

```
$T_s[] \leftarrow$ initialize summary tables
$T_M \leftarrow$ initialize $M$ table
$h \leftarrow$ heap
for Each record $u$ do
    $h$.push ($<u, u.M, u.similar>$)
end for
while not empty($h$) do
    $<u, u.M, u.similar> \leftarrow h.top()$
    $T_s[u] \leftarrow T_s[u] + T_s[u.similar]$
    update $<u, u.M, u.similar>$
    $h$.push ($<u, u.M, u.similar>$)
    updating_after_merging() //Algorithm 2
end while
```

Algorithm 2 ACE.update_after_merging()

```
$C_i \leftarrow$ master cluster, $C_j \leftarrow$ merged cluster
release $T_s[C_j]$
invalidate $M$ table entries $(C_j, *)$
update $M$ table entries $(*, C_i)$ and $(*, C_j)$
for Each valid cluster $u$, if $u.similar == C_i$ or $C_j$ do
    update $<u, u.M, u.similar>$;
    relocate $<u, u.M, u.similar>$ in $h$
end for
```

ACE is initially designed for static datasets and its $O(N^2 \log N)$ complexity prevents it from working directly on large datasets or data streams. It has been shown that ACE can run on samples of static large datasets so that the generated sample BKPlots are consistent with the one on the entire dataset. However, the sampling approach cannot be directly applied to the data stream since the clustering structure may change over time. In the next section, we will design a data summarization tree structure, again, with the help of IE. Combining this tree structure and an extended ACE algorithm, we can continuously generate high-quality BKPlots for data streams.

4 HE-Tree: capturing cluster entropy of the categorical data stream

Data stream processing has an important feature that the stream processor has only one chance to see any individual record. It is impossible to retrieve historical data more than once due to the huge amount of data, although some of the latest records can be possibly buffered. Often, the strategy is to preserve only the aggregate information over periods of the stream with a focus on the recently processed data. In this section, we design a summarization structure—Hierarchical Entropy Tree (HE-Tree), to capture entropy characteristics of the categorical data stream in aggregates and achieve low cost and high precision in detecting the change of clustering structure.

The basic idea of the HE-Tree is to coarsely and rapidly assign the records from the data stream onto hundreds of sub-clusters. Applying an algorithm similar to ACE to these sub-clusters will give us a precise estimation of the clustering structure. The HE-Tree determines which subcluster a new record should be assigned to, only based on previously processed data records, and the entire clustering structure keeps evolving with aggregated new data records. In order to better understand the structure, we will give a simple version first, and a time-decaying structure will be discussed later.

A tree structure is good for efficient search. Therefore, we organize subclusters in a tree, i.e., HE-Tree, for convenient search and assigning a new coming record to a subcluster. An HE-Tree consists of two key components:

1. An *HE-node structure*, which summarizes entropy characteristics of a group of records and facilitate fast processing of stream data items;
2. An *Incremental-Entropy based lookup/assigning algorithm*, which helps adapt the changing clustering structure.

Given fixed tree height h and fanout f, an HE-Tree is constructed in two stages:

1. a *growing stage*, which happens only at the beginning of processing the data stream when the tree is not full;
2. an *absorbing stage*, which absorbs the new coming items to the subclusters at the leaf nodes, when the tree is full.

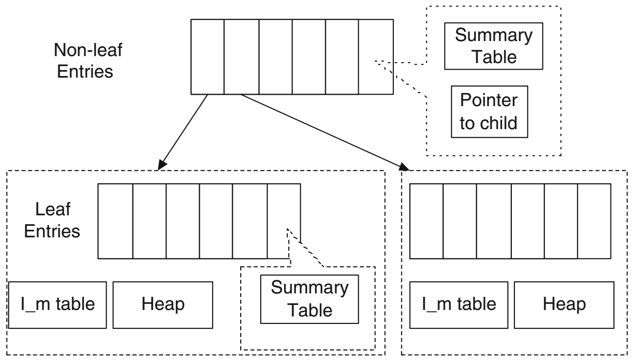


Fig. 8 Structure of the HE-Tree

We first describe the structure of the HE-Tree node, which includes a few data structures for fast entropy calculation. After that, we will focus on the algorithms for generating and updating the HE-Tree.

4.1 Structure of the HE-Tree

Summary Table. The summary table is used to maintain fast calculation of the entropy $\hat{H}(C_k)$. Each node in the HE-Tree maintains one summary table, which has the same structure as shown before in Fig. 6.

Nodes in the HE-Tree. The HE-Tree is a balanced tree similar to the B-tree, where each node has f entries. Each entry in the leaf nodes represents a subcluster and each in the internal nodes represents the summary of its subtree. As shown in Fig. 8, each entry in the leaf node contains a summary table, and a leaf node also has an M-table with $(f+1)^2$ entries and an M-heap for fast locating or merging entries. The structure of the M-table is shown before in Fig. 7. An internal node (non-leaf) in the tree contains only the aggregation information of its child nodes.

Concretely, the aggregation information in an entry of an internal node is also stored in the summary table. This can be easily maintained based on the subtree's summary tables, because of the property of entropy aggregation. Let vector $\vec{s}$ represent a summary table and the entropy characteristic of any internal node C_i denoted as $\mathrm{EC}_i(n_i, \vec{s_i})$, where n_i is the number of records summarized by its subtree. Let C_{ij}, $1 \le j \le f$ represent the child nodes of C_i. The HE-Tree maintains the following property.

$$\mathrm{EC}_i(n, \vec{s}) = \sum_{j=1}^{f} \mathrm{EC}_{ij}(n_{ij}, \vec{s_{ij}}) = \mathrm{EC}_i\left(\sum_{j=1}^{f} n_{ij}, \sum_{j=1}^{f} \vec{s_{ij}}\right) \quad (3)$$

i.e., the parent node represents the merge of the child nodes. This is recursively done from root to leaves.

The major benefit of the HE-Tree structure is to approximately minimize the overall expected entropy by locally minimizing the expected entropy of the selected branch $\bar{H}(C_i^f)$, when a new record is inserted. This local minimization is achieved through the following algorithms for constructing a HE-Tree.

4.2 Constructing the HE-Tree

Building an HE-Tree consists of two phases: *the growing phase* and *the absorbing phase*. The following algorithms are carefully designed to minimize the expected entropy of the subclusters and to adapt to the change of entropy in the data stream with minimal computational cost.

Growing phase. In the growing phase, the tree grows until the number of leaf nodes reaches $\lceil n_c/f \rceil$. It turns to the next phase (absorbing phase) when all entries in the leaf nodes are filled. When a new coming record is inserted into the existing tree, the first subroutine is to locate the target leaf node for insertion. Let e denote the inserted record and e_i denote one of the entries in the current node. The search for the target node begins at the root node. Since each entry in the internal node contains the summarization information of its sub-tree, we can find the most similar subtree to e by finding the minimum value among $M(e, e_i), i = 1, \ldots, f$, i.e.

$$e_t = \text{argmin}_{e_i}\{M(e, e_i), i = 1, \ldots, f\}$$

Iteratively, the same criterion is applied to the selected child node until a leaf node is reached.

If there is an identical entry at the leaf node, i.e., $M(e, e_i) = 0$, the record is merged to the identical entry with the summary table updated. Otherwise, if the target leaf node still has empty entries, the record is assigned to one empty entry. The corresponding summary tables of the nodes in the path from root to the target leaf node are updated to maintain the property Eq. 3. If all conditions are dissatisfied, it is going to split the leaf node. We give the sketch of the subroutines in Algorithms 3 and 4.

Algorithm 3 HE-Tree.locate(node, e)

```
$node \leftarrow$ target node, $e \leftarrow$ target entry
if node is leaf then
  return node
end if
for Each entry $e_i$ in node do
  $M^i \leftarrow M(e, e_i)$
end for
$e_t \leftarrow argmin_{e_i}\{M^i\}$
return locate($e_t.subtree$, e)
```

When the target leaf node is full but the tree size is not grown to the presetting size, a *split* operation is applied. In the split algorithm, the entries are partitioned into two groups. First, a pair of pivot entries (e_r, e_s) is found in the target

Algorithm 4 HE-Tree.insert(node, e)

```
$e \leftarrow$ inserted entry, $node \leftarrow$ target node
for Each entry $e_i$ in node do
  if $M(e, e_i) == 0$ then
    merge(e, $e_i$), return
  end if
end for
if node.have_empty_entry() then
  node.enter(e)
  if ($node.num\_entry() == f - 1$ and (not tree_full() or is_internal(node)) then
    split(node)
  end if
else
  leaf-merging(node, e) //fine merging in absorbing phase
end if
```

node that has the maximum M—they are regarded as the most dissimilar pair among all pairs.

$$(e_r, e_s) = \text{argmax}_{e_r, e_s}\{M(e_r, e_s), \quad i = 1, \ldots, f\}$$

These two pivot entries then become the two seed clusters. The remaining entries are sequentially assigned to the two clusters so that the overall expected entropy of the partition keeps minimized. Then, a new node is generated to accommodate one of the two sets of entries, and one entry is added into the parent node pointing to the new node. The insertion/splitting continues until the number of leaf entries reaches n_c. Algorithm 5 gives the detailed description of the split operation.

Algorithm 5 HE-Tree.split(node)

```
node $\leftarrow$ target node
$(e_a, e_b) \leftarrow argmax_{(e_i, e_j)}\{M(e_i, e_j)\}$
$partition_a \leftarrow e_a$, $partition_b \leftarrow e_b$
for Each entry $e_i$ in node do
  if $M(partition_a, e_i) < M(partition_b, e_i)$ then
    $partition_a \leftarrow partition_a \cup e_i$
  else
    $partition_b \leftarrow partition_b \cup e_i$
  end if
end for
if is_leaf(node) and not done then
  re-insert(root, entries in $partition_a$)
else
  newnode $\leftarrow partition_a$, remove(node, $partition_a$)
  $e_{new} \leftarrow$ summary(newnode), insert(node.parent, $e_{new}$)
end if
```

Absorbing phase. In the second phase, the same locating algorithm is applied to locate the target leaf node for the new record. However, we have no insertion allowed since the entries are all occupied. Instead, in the leaf node we need to merge the most similar two items among the $f+1$ items—the f entries in the leaf node plus the new record. This allows the tree to rapidly adapt to the change of clustering structure at the leaf entry level.

In each leaf node, we maintain an M-table and an M-heap for the f entries. When a new record comes, only f IE calculations are needed to update the M-table and the heap before the algorithm determines the most similar two to merge. Since f is small, this can be done very quickly.

The locating algorithm with the IE criterion will assign a new record to the approximately best leaf node and the merge algorithm will adapt to the local structural change happening within the node. Although the whole process is based on node-based local decision, which by no means is global optimal, experiments show that the leaf summary entries generated with the above procedure can be used to precisely detect the best K number of clusters—as we have discussed in previous paper [13], high-quality *approximate* BKPlot will still give the best Ks.

4.3 Analysis of complexity

The time complexity of the HE-Tree construction differentiates in the two phases. The first phase will be quickly finished. However, we still should guarantee that the cost will be acceptable to a streaming setting. Let h be the height of the tree (the root is at level 1). In the growing phase, about f^h records are inserted into the tree and each record needs at most $O(hf)$ comparisons to locate the target node. The remaining costs will be negligible. In the absorbing phase, besides the cost of locating, each record needs the merge operation at the leaf, which costs $O(f)$ incremental-entropy calculations. Each incremental-entropy calculation involves only weighted entropy, which costs $O(m)$ space, where m is the total column cardinality. Therefore, the cost for one record is $O((h+m)f)$ in the absorbing phase. Since f is usually a small value, e.g. 10–20 and $h = 2$ or 3 in practice, the total cost is only dominated by the total cardinality m of the dataset.

There are $O(f^h)$ nodes in the tree. Each leaf node needs approximately $O(fm + f^2)$ space, where the summary table for each entry needs $O(m)$ and the M-table needs $O(f^2)$ space. Each internal node needs only $O(fm)$ space for holding the summary tables and the pointers to the children nodes. Approximately, an HE-Tree needs $O((m+f)f^{h+1})$ space. With fixed small f and h, again only the factor m of the dataset determines the size of the tree.

Except when the datasets have very large total cardinality m, e.g., over 10k, an HE-Tree can be built up in any commercial computers or even a PC. The computational cost will be very acceptable as well. In summary, the HE-Tree can efficiently summarize cluster entropy of a data stream with small amount of time and space costs.

4.4 Discussion on setting of parameters

The setting of the two parameters f and n_c can affect the efficiency and quality of summarization. For simplicity, we always construct full trees and allow $n_c = f^h$ to vary from hundreds to thousands. For example, for $f = 15$, we can either use a two-layer tree, where the number of leaf entries $n_c = 225$, or a three-layer tree where $n_c = 3375$. A small f always results in faster summarization, but can undermine the quality of summarization when the clustering structure is evolving. The reason is that a small f may cause more imprecise merges to happen in the absorbing phase. The less entries the node has, the lower level of precision is guaranteed in each level of the merge operation. This is fine if the clustering structure keeps unchanged or is slowly changing, but such a tree will not easily adapt to dramatic changes. This problem can be alleviated with a "time-decaying" structure, which we will discuss later.

The setting of f also has concerns in the computational cost. A larger f with the same height of tree will increase the cost due to the complexity $O((h+m)f)$ in the absorbing phase. Increased cost will reduce the streaming rate that a stream processor can handle. In practice, we need to balance the performance and robustness according to the application requirements. Experimental results show that if we set the tree to be 2–3 layers, with $f = 10$–20, we can achieve a good balance between performance and robustness.

5 A monitoring framework based on the HE-Tree

In last section, we have designed the HE-Tree algorithm for summarizing the data stream. In this section, we discuss how to use the HE-Tree to generate effective BKPlots. Overall, we set two levels of granularity for change detection. First, we cut the data stream into windows. A window covers a relatively long period. Each window is divided into several time units. A time unit is defined as the minimum interval that the monitoring framework can dump summary information from HE-Tree and generate a detection report (i.e., a BKPlot) with an *extended ACE algorithm*. Within a time unit, many records from the data stream are processed and absorbed by the HE-Tree. Between windows, the HE-Tree from the previous window is appropriately discounted, which is called *time-decaying HE-Tree*. The time-decaying HE-Tree gives additional flexibility in tuning HE-Tree's sensitivity to recent changes in data streams. Therefore, the entire framework for detecting the change of clustering structure is based on the basic HE-Tree summarization algorithms, a time-decaying HE-Tree algorithm, and the extended ACE algorithm that generates BKPlots from HE-Tree. Below, we start with the description of time-decaying HE-Tree, followed by the extended ACE algorithm and how to generate and analyze a detection report.

5.1 Problem of time decaying summarization

The HE-Tree organizes the historical entropy characteristic of a categorical data stream into a bunch of subclusters. However, in some cases, the historical structure should be outdated or discarded, and we may focus on the recent changes more if the stream changes frequently. Although the basic HE-Tree is able to adapt to the changes, it will be less sensitive to the recent change of structure. Figure 24 (in Sect. 6) illustrates such a scenario, where new patterns (C3 and C4) are overlapped by old patterns. The recently emerged clusters could be more important than the old ones. In this case, it is necessary to discount the effect of historical clustering structures and give more weight to new clustering structures.

Two approaches have been developed in other stream mining techniques to address this problem. One approach is a fading clustering structure [2], where each data point $\vec{x}_i$ is associated with a time stamp and the weight of the point is decayed with the time difference between its time stamp and the current time. The other is a sliding window, which focuses on the clustering structure of the data points collected in the current time window. We will briefly discuss the related techniques, and propose a window-based decaying structure for the HE-Tree based change detection.

The fading clustering structure [2] was developed for clustering numerical data with K-means-like algorithms, where the first- and second-order moments, i.e., $\sum \vec{x}_i$, and $\sum \vec{x}_i^2$, are enough to describe features of a cluster. A fading clustering structure associates *each point* $\vec{x}_i$ with a time-decaying weight $f(t)$, and the decayed moments become $\sum f(t)\vec{x}_i$, and $\sum f(t)\vec{x}_i^2$, correspondingly. The definition of $f(t)$ enables to incrementally update the weighted moments when the time t increases. A typical f function is $f(t) = 2^{-\lambda \cdot t}$ [2]. There are two advantages of the fading clustering structure. First, it precisely describes the time difference between points and the effect of old points is correspondingly discounted. Second, it is simple to update such a fading clustering structure, if the clustering structure can be sufficiently described by some simple statistics like moments. Clearly, to apply a fading clustering structure, we should have simple and sufficient statistics for describing a clustering structure. Since the weight is associated with each point, another major concern is the cost of updating the structure.

Sliding-window based stream processing [4] handles data points by time windows. The online processing part is based on the data collected in the current time window and only the summary information of some recent historical time windows might be saved. In window-based processing, the information from recent old windows are discarded or discounted in the current window. Window-based stream processing is widely used for lower level information aggregation [4] and stream-join [19]. Compared to the fading structure, within a window, the time related intensive update for each point does not exist, which saves considerable cost. Obviously, the lower cost is traded off by the granularity of time information.

5.2 The time-decaying HE-Tree

It is possible to combine the two approaches with our HE-Tree based algorithm. Because the function of the summary table is to aggregate occurrences of each categorical value in each column, it is possible to discount historical occurrences in the summary table to implement a fading clustering structure. However, the cost of the fading clustering structure could be very high, since each insertion to the HE-Tree will trigger an additional time-stamp update to *all* entries of the summary tables in the path from the leaf node to the root. Considering this cost, we propose the time-decaying HE-Tree structure based on the combination of a fading structure and window-based processing.

Concretely, we assume that the basic decaying unit is the window, which can span over t time units. Within a window no decaying operation happens. We allow the HE-Tree of the last window W_{i-1} to be inherited by the current window W_i, with the following update to each node in the tree. Let the decaying rate $\lambda, 0 < \lambda < 1$, represent the proportion of the information that is preserved from the last window. The window-to-window information passing can be simply implemented in the following steps:

1. passing the HE-Tree from the last window to the current window;
2. updating all summary tables in the tree by multiplying λ to each entry of the table;
3. updating the count of aggregated record n in each entry of node to $n\lambda$;
4. for the M-table in each leaf node, each entry is updated by multiplying by λ, so is each entry in the M-heap.

Intuitively, the above steps are equivalent to uniformly sampling $n\lambda$ records from previous window W_{i-1} that has n records. In general, a previous window W_{i-w} that has n_w records will have $n_w\lambda^w$ preserved at the current window. These steps can be done by scanning each node in the HE-Tree once. In order to accommodate the above design, we also change the counters in the summary table to floating point, allowing non-integer entries generated by discounting the historical structure. Here is a fast conclusion we can draw.

Proposition 3 *Updating each entry in the summary table and the aggregated record n by the rate λ preserves the cluster entropy in the last window.*

Proof Sketch According to the entropy definition, the cluster entropy is determined by $\sum_{i=1}^{d}\sum_{v\in A_i} p(x_i = v)\log p(x_i = v)$, where v is a categorical value in column i and $p(x_i = v)$ is the probability that v occurs in column i. Assume there are n_v records having value v at the column i. Due to the fact that the probability $p(x_i = v) = \frac{n_v\lambda}{n\lambda} = \frac{n_v}{n}$, if we use non-integer counters, the cluster entropy is not changed. □

Theoretically, the non-integer counters will always preserve the clustering structure with any decaying rate $\lambda > 0$. However, the decaying process will surely downgrade the weight of the old clusters, which relatively increases the weight of the new coming records. As a result, the HE-Tree will be more adaptive to changes. How does it happen? We will take a deep look at the micro-level that eventually drives the change of clustering structure.

Since the absorbing operation depends on the IE between subclusters and the new coming record, we analyze the change of the M-table entries. Remember that at the leaf node where a new record reaches, it is either merged to one entry (subcluster), or two entries merge and the new one occupies the empty entry—all depend on the IE. When two entries merge, it means the structure is adapted to the new coming record that becomes the seed for the new subcluster.

To better understand why the decaying algorithm works, consider what happens when the first record comes in the new window. Let the new record denoted as a single cluster C_1, which has zero cluster entropy. Without decaying, the IE in terms of any subcluster C_p is

$$M(C_p, C_1) = (n_p + 1)\hat{H}(C_p \cup C_1) - n_p\hat{H}(C_p)$$

When $n_p \gg 1$, $\hat{H}(C_p \cup C_1) \approx \hat{H}(C_p)$, we get $M(C_p, C_1) \approx \hat{H}(C_p)$, i.e., the existing cluster entropy determines the IE.

With decaying, each M-table entry is multiplied by λ, i.e., $M(C_p^{(\lambda)}, C_q^{(\lambda)}) = \lambda M(C_p, C_q)$. Again, assume each window accumulates significant amount of data records $- n \gg 1$ and $n\lambda \gg 1$ are true. A new coming record in the new window will still have

$$\begin{aligned} M\big(C_p^{(\lambda)}, C_1\big) &= (n_p\lambda + 1)\hat{H}(C_p \cup C_1) - n_p\lambda\hat{H}(C_p) \\ &\approx \hat{H}(C_p \cup C_1) \approx M(C_p, C_1) \end{aligned}$$

$M(C_p^{(\lambda)}, C_q^{(\lambda)})$ has been decayed by the rate λ, while $M(C_p^{(\lambda)}, C_1)$ of absorbing the new record is not changed. Therefore, old entries are more likely to be merged than those in the non-decaying structure.

By decaying old records, the accumulated new records in one entry will more easily dominate that entry as well. When an old subcluster absorbs significant amount of similar new records, with decaying structure it would appear more like slow drifting of clusters (the right of Fig. 9), rather than simple expanding of the old cluster (the left of Fig. 9).

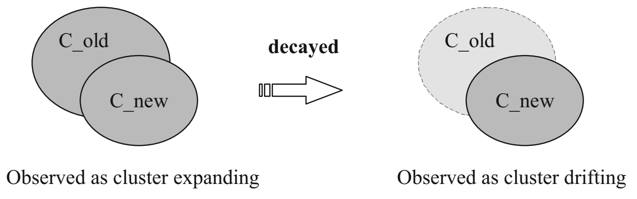


Fig. 9 The effect of decaying

Decaying rate and other factors It would be complicated to determine the right decaying rate for a specific data stream, since it is related to many factors, such as the size of large historical clusters, the data pumping rate in the stream, the length of interested observation period, and specific requirements from the application. Except the application-specific factors that we do not know, we can use the following method to estimate the setting of decaying rate based on the information we have known.

First of all, we assume the data pumping rate is approximately r records per time unit, which can be easily estimated. So a window having t time units will process rt records approximately. Assume we are more interested on clusters with more than n_0 records. Apparently, the data rate r put a limit on the size of the smallest cluster we want to observe, i.e., $r \le n_0$. Also, we are more interested in a period of w windows. Then, we expect a cluster at W_i should be diminished to a size less than n_0 at the window W_{i+w}.

With the above setting we are able to estimate the range of appropriate λ. Based on the leaf entries in the HE-Tree and the extended ACE algorithm, we can estimate the size of the largest cluster in previous windows, assuming it is n. In order to guarantee that a cluster with n items will be diminished to at most n_0, we need to establish the relation

$$n \times \lambda^w \le n_0$$

which leads to $\lambda \le \sqrt[w]{\frac{n_0}{n}}$.

On the other hand, we do not want λ be too small. Otherwise, the records from the last window will have only negligible effect on the current window. In experiments, we will further study how the decaying rate λ is related to the sensitivity of change detection.

Cost Finally, it is easy to calculate the additional cost of decaying the previous HE-Tree, which is about $O(f^h(m + f^2))$, since there are $O(f^h)$ nodes, while each summary table has m entries and each M-table has f entries.

5.3 The extended ACE algorithm

At the end of each window, sub-clusters, usually hundreds to thousands, in leaf nodes are output for detecting the change in the clustering structure. The ACE algorithm is revised to take the sub-clusters as input and generate BKPlots. Suppose there are n_c sub-clusters generated by the summarization.

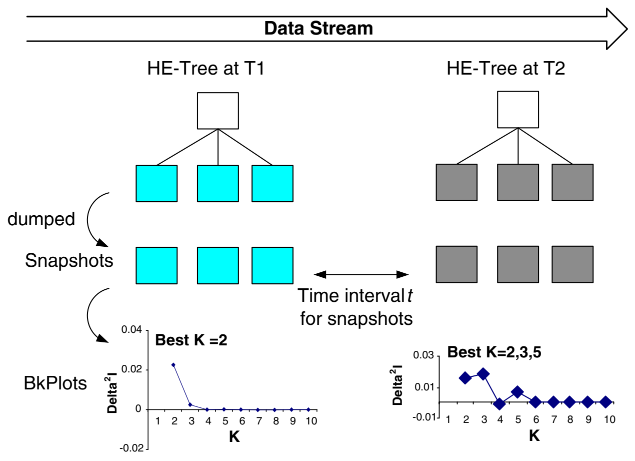


Fig. 10 Detecting the change of clustering structure in a categorical data stream in one window

The property of ACE algorithm allows us to do this. Instead of starting with each point as a cluster, it takes n_c sub-clusters as input and consecutively merges the pair of clusters that minimizes the IE among the remaining clusters. With the same set of structures, i.e., the summary table, the M-table and the heap, which are used by ACE algorithm, the extended ACE algorithm has complexity $O(n_c^2 \log n_c)$. Since n_c is only several hundreds to thousands in practice, the extended ACE algorithm can be done very quickly. The only question left is whether the extended ACE algorithm together with the HE-Tree can generate high-quality BKPlots, which will be studied in experiments.

5.4 The monitoring procedure

With the time-decaying HE-Tree and the extended ACE algorithm, we can precisely monitor the change of the clustering structure in the categorical data stream. The framework is illustrated in Fig. 10. The working mechanism can be described as follows.

1. The HE-Tree from the last window is decayed, which costs $O(f^h(m + f^2))$.
2. Records from the data stream are inserted into the HE-Tree by their arrival order. Each insertion costs $O((h + m)f)$;
3. At certain time interval Δt, the summary tables in the leaf nodes are dumped out (to certain memory area or to disk) as a snapshot of the clustering structure. It costs $O(mn_c)$ bytes to store each snapshots;
4. The extended ACE algorithm is performed on the snapshot as needed, the result of which generates a BKPlot. The cost is $O(mn_c^2 \log n_c)$.

From step 2, we know that f is also an important cost factor in processing a record. There is a tradeoff between the precision and the capacity of the monitoring system, tuned by the parameter f. The costs of steps 3 and 4 affect how often we can generate a BKPlot. The time interval Δt in step 3 can be directly determined by the cost of generating the BKPlot. If we want to maximize the number of BKPlots in unit time, Δt should be proportional to the cost $O(m(f^h + n_c^2 \log n_c))$. Therefore, smaller n_c and f allow more snapshots to be processed in unit time, and thus more details about the changes to be observed. In experiments, n_c=400–1,000 and f = 15 shows it is sufficient to generate precise BKPlots for the experimental data, which means the necessary Δt is usually very small. On the other hand, how often we need to monitor the change in the data stream also depends on the application requirement. For example, in monitoring a communication network, we may need to detect the changes between seconds, but in road traffic monitoring, we may only need to check the change in every ten minutes.

The neighboring BKPlots can be analyzed to see the difference between the clustering structures. BKPlots can be represented as a function $B(K)$, where K is the number of clusters and the peaks of $B(K)$ indicate the candidate best Ks. Without loss of generality, we suppose the corresponding Ks of the highest κ peaks on BKPlots are $\Gamma = \{k_1, k_2, \dots k_\kappa\}$. Let Γ^{old} and Γ^{new} represent two set of Ks on the consecutive BKPlots, respectively. There are two kinds of important difference we need to notice.

1. If Γ^{old} and Γ^{new} are not identical, the clustering structure is dramatically changed, which raises an "alarm" that we need to analyze the snapshot of Γ^{new} in detail.
2. If Γ^{old} and Γ^{new} are identical, but at certain k_i that $|B(k_i^{new}) - B(k_i^{old})| > \theta$, where θ is a threshold we need to notice, we can infer that some minor changes happen in clustering structure: if $B(k_i^{new}) > B(k_i^{old})$, i.e., $I(K)$ curve changes more dramatically at k_i, the clustering structure of k_i clusters becomes more clear than before; reversely, the boundaries between the k_i clusters may become vague and some clusters tend to converge.

Certainly, more diagnosis techniques, and other stream mining methods can be combined with our framework to generate valuable information.

6 Experiments

The goal of the experiments is three-fold. (1) We investigate the parameter setting for the HE-Tree and give an estimate of appropriate settings; (2) the non-decaying HE-Tree together with the extended ACE algorithm will provide high-quality BKPlots; and (3) we study the properties and benefits of the

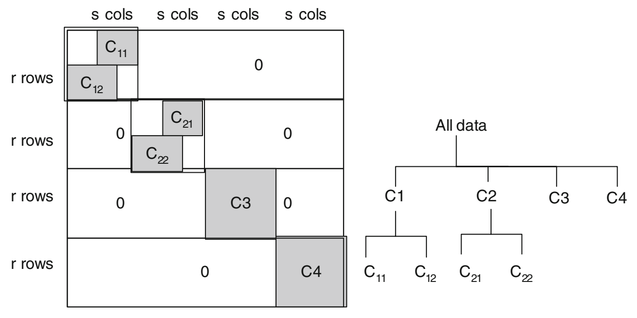


Fig. 11 Clustering structure of DS1

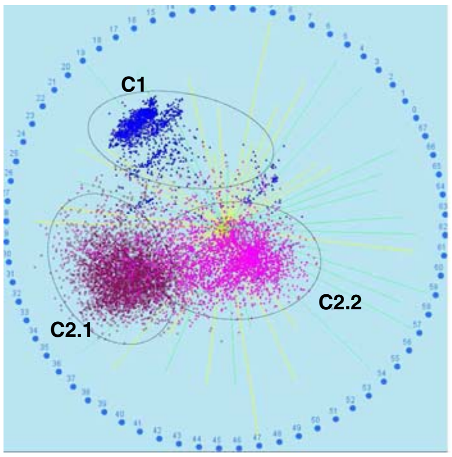


Fig. 12 Clustering structure of Census data

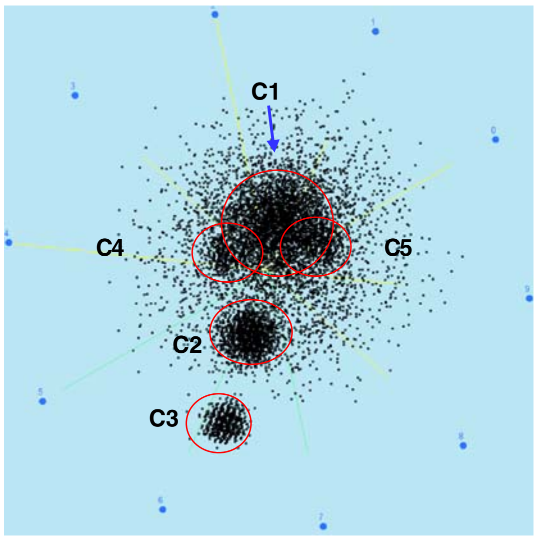


Fig. 13 Clustering structure of simulated data DMIX

decaying HE-Tree structure, in terms of the factors, such as the decaying rate λ and the size of old and new clusters.

Datasets We construct the first synthetic dataset DS1 with the following way, so that the clustering structure can be intuitively identified and manually labeled before running experiments. Such datasets can be used to evaluate the precision of the clustering result. The synthetic dataset has a two-layer clustering structure (Fig. 11) with 30 attributes and N rows. It has four equal-sized clusters in the top layer. Each cluster has random categorical values selected from {‘0’,‘1’,‘2’,‘3’,‘4’, ‘5’} in some distinct set of attributes (the dark area in Fig. 11), while the remaining attributes are set to ‘0’. Two of the four clusters also have a clustering structure in the second layer. This synthetic data has a clearly defined clustering structure, and each record in the dataset distinctly belongs to one cluster. This dataset is primarily used to explore the effect of different parameter settings of the HE-Tree to (1) the precision of the clustering result and (2) the efficiency of the summarization. This dataset is also used to illustrate the time-decaying effect.

We also use a real dataset: “US Census 1990 Data ” in the experiment. This dataset is a discretized version of the raw census data, originally used by [35]. It can be found in the UCI KDD Archive.[1] Many of the less useful attributes in the original data set have been dropped, the few continuous variables have been discretized and the few discrete variables that have a large number of possible values have been collapsed to have fewer possible values. The total number of preserved attributes is 68, including the column of sequence number, which is discarded in clustering. This dataset contains about 2 million records. Since this dataset is the discretized version of the original numerical data and distances are approximately preserved with the discretized values, we are able to visualize it with the VISTA tool [15]. The visualization of its sample dataset validates three major clusters, two of which are close to each other and the third one is smaller than the other two (Fig. 12). Experimental results of categorical clustering should be consistent with the visualization.

The third dataset DMIX follows the similar idea of the Census data, used for studying the effect of clusters of unbalanced sizes, noise data, and cluster overlapping caused by historical large clusters. We simulate a Gaussian mixture model with a structure visualized in Fig. 13, and then discretize it. The five clusters in the order of $\{C_1, C_2, C_3, C_4, C_5\}$ are pumped into the data stream. About 10% random noise records are also injected, uniformly distributed over the whole data stream. The cluster C_1 is the largest cluster with 50% records, followed by the 4 equally sized clusters. Clusters C_4 and C_5 are overlapped by the large cluster C_1, which is used to study how the time-decaying structure addresses the sequential cluster overlapping problem.

Metrics: error rate Cluster labels in the synthetic dataset DS1 allow us to precisely evaluate the quality of the clustering result with the *Error Rate* measure. Suppose the best K clusters are identified. Error Rate is defined based on the

[1] http://kdd.ics.uci.edu/.

confusion matrix, where each element c_{ij} $1 \leq i, j \leq K$ represents the number of points from the labeled cluster j assigned to cluster i by the algorithm. Let {(1), (2), …, (K)} be any permutation of sequence {1, 2, …, K}. There is a permutation that best matches the clustering result and the labeled result that maximizes the number of consistent points m_c.

$$m_c = \max \left\{ \sum_{i=1}^{K} c_{i(i)}, \text{ for any } \{(1), (2), \ldots, (K)\} \right\}$$

We define the Error Rate as $1 - \frac{m_c}{N}$, where N is the total number of points.

6.1 Parameter Setting for the HE-Tree

This set of experiments will roughly investigate the effect of parameters, primarily, f and n_c, for the HE-Tree, with the non-decaying trees. To simplify the investigation and maximize the quality of the summarization, we always use full trees in the experiments. Intuitively, for a fixed f, the higher tree (the larger h), the finer granularity of summarization will be delivered. In most cases, we care about only clustering structures having less than 20 clusters. Therefore, a short tree, which generates less than one thousand subclusters, is enough for achieving high-quality BKPlots with the extended ACE clustering algorithm. The experiment will focus on full short trees (e.g., $h = 2$) with varying fan-out f from 10 to 30. If the height of tree h is fixed, n_c is determined by f. Therefore, we will study only the effect of f. A set of datasets (20 datasets) in the same structure shown in Fig. 11 are generated, and the result is based on the average and variance of 20 runs.

Figure 14 shows that the cost of HE-Tree summarization is linear to f, which is consistent with our analysis. Figure 15 shows the effect of different settings of f on the quality of the final clustering result for "Unordered DS1". Unordered DS1 randomly stores the records from different clusters, i.e., there is little change of clustering structure in processing the data stream. The result shows some variances between the error rates for different f, but overall the error rates are similar and low.

"Ordered DS1" shows a more interesting scenario, where the clustering structure dramatically changes in one window. In such situations, f may significantly affect the quality of monitoring. Figure 16 shows the result of sequentially processing the clusters C_{11}–C_4. A tree with larger f seems more adaptive to the change of clustering structure. The phenomenon be understood as follows. The initial records from the same cluster already occupy the slots in the growing stage. When a new cluster emerges, since there is no empty entry in the tree belonging to the new cluster, new slots are created by merging other similar entries. Small f may result in merging

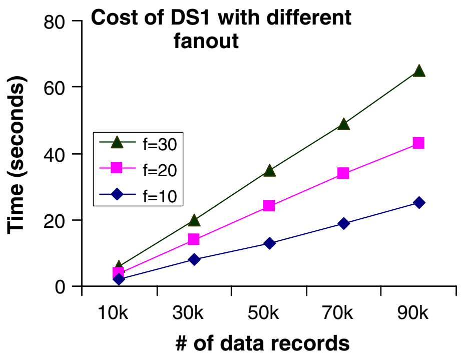


Fig. 14 Cost of HE-Tree summarization with different fanout f

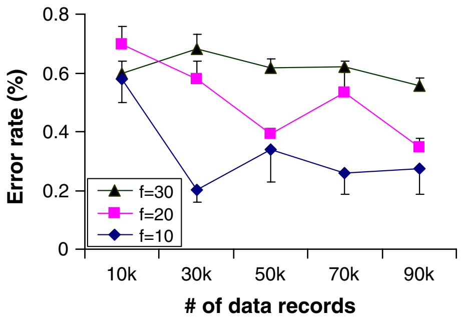


Fig. 15 Error rate of clustering result with HE-Tree summarization on randomly ordered records

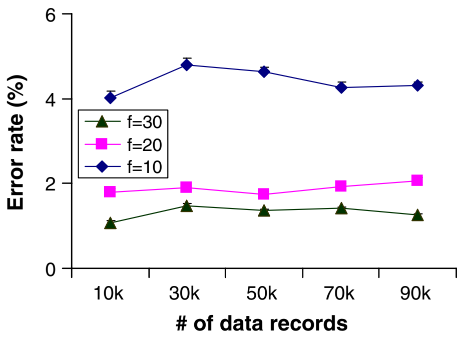


Fig. 16 Error rate of clustering result with HE-Tree summarization on ordered records

entries that belong to different cluster eventually. However, it shows that increasing f from 10 to 20 can considerably reduce the error, but $f = 30$ will not significantly improve the result any more. Balanced with the time cost and the robustness, we will use $f = 15$ in later experiments. Similarly, the effect of ordered records may also happen across windows. With the decaying HE-Tree, the confliction is kind of relieved, compared to the scenario within a window.

6.2 Robustness of BKPlots by the HE-Tree/extended ACE

In this set of experiments, we compare the accuracy of BKPlots generated by the ACE algorithm on small sample

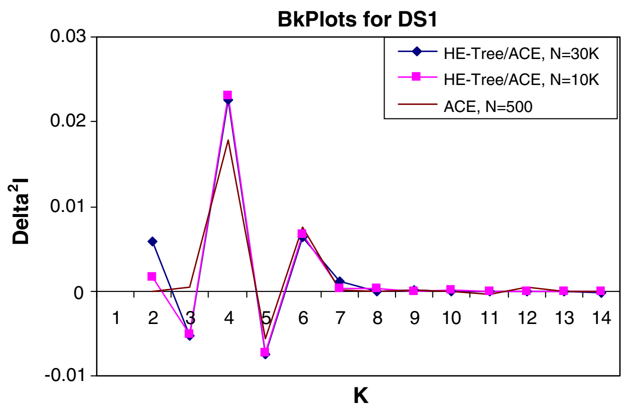


Fig. 17 BKPlots for DS1

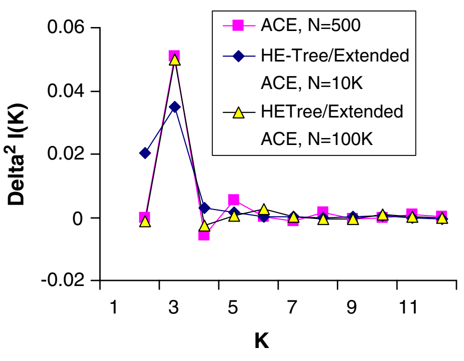


Fig. 18 BKPlots for Census

sets and by the HE-Tree/extended ACE on large stream data. We run the experiment on both the synthetic data and the real US Census data. The small sample size is set to 500 for ACE, and large streams consist of sample sets with $10K$ and $100K$ records, respectively. Sample sets are uniformly drawn from the original dataset, therefore, they are supposed to have the same clustering structure for sufficiently large sample set. Again, the study is done with non-decaying trees.

Figure 17 for DS1 shows that all of the three BKPlots can identify the primary best Ks: 4 and 6, while a little noise appears at $K=2$ when the sample size is large. All BKPlots for Census data (Fig. 18) strongly indicate $K = 3$ is the best, while $K = 2$ is probably another candidate(which groups the cluster C2.1 and C2.2 together). The result confirms that HE-Tree summarization can preserve the primary clustering structure.

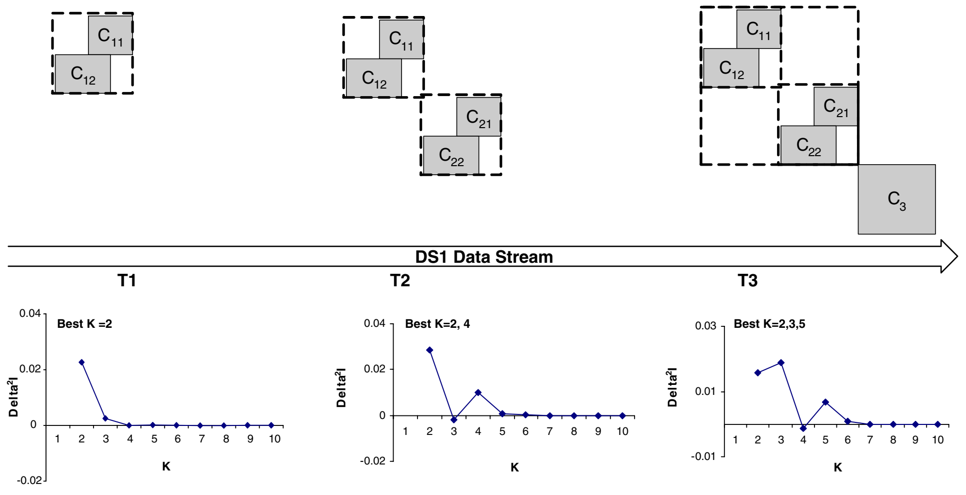
6.3 Detecting changes

We demonstrate the progressive monitoring results of the two data streams: DS1-stream and Census-stream, with non-overlapping clustering structures and non-decaying HE-Tree ($\lambda = 1$). DS1-stream simulates the 4/6-cluster structure shown in Fig. 11. The clusters enter the stream in the sequence of C_{11}, C_{12},C_{21},C_{22}, C_3, and C_4, without overlapping. Each of the small clusters has $5K$ records and each of the large clusters have $10K$ records. Snapshots are saved at three windows corresponding to $N=10K$, $20K$, and $30K$, respectively.

The progressive result for DS1-stream in Fig. 19 clearly identifies the change of clustering structure. At T1:$N=10K$, C_{11} and C_{12} have been present at the stream, thus two clusters are identified. At T2:$N=20K$, C_{21} and C_{22} emerge and the two-layer structure is identified (the best $K = 2, 4$). At T3:$N = 30K$, C_3 appears, and the BKPlot detects that the primary two-layer structure is changed to $K = 3, 5$, while the BKPlot also suggests an additional layer at $K = 2$, which consists two cluster (C_{11}, C_{12}, C_{21}, C_{22}) and (C_3).

We partition the census dataset into four parts and feed the parts sequentially into the HE-Tree, so that the special clustering structures appear in different stage as Fig. 20 shows. At the first snapshot, there are clearly two clusters; in the second one, the third cluster shows vaguely; finally, a two-layer clustering structure ($K=2$ and 3) appears in the third

Fig. 19 Monitoring DS1-stream



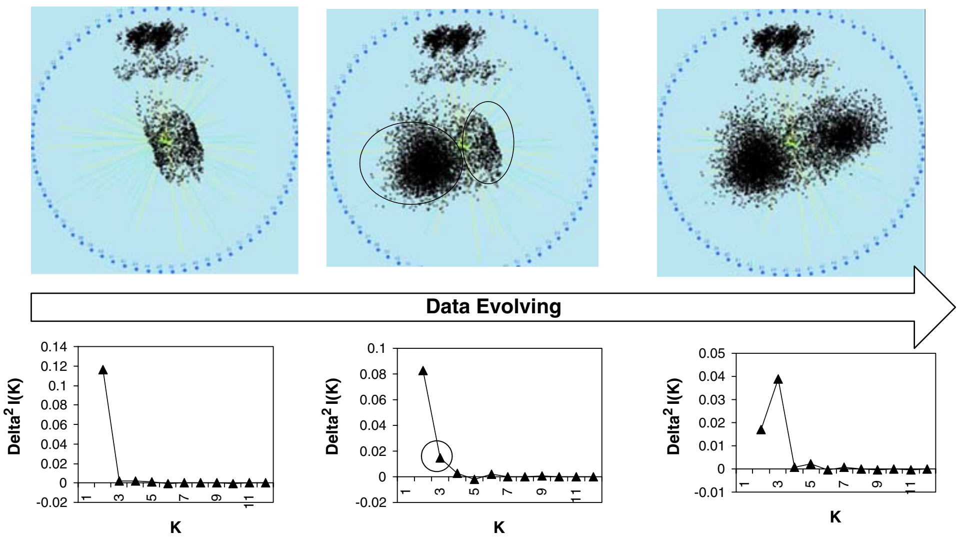


Fig. 20 Monitoring Census-stream

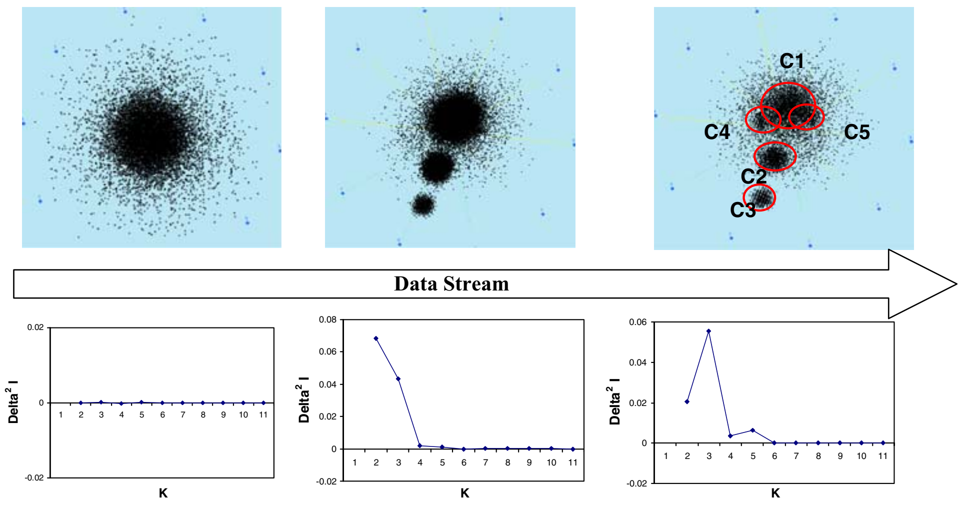


Fig. 21 Sequential overlapping clusters in data stream

snapshot. Snapshot 2 demonstrates that the HE-Tree can also capture the fine changes in the clustering structure (Fig. 20).

The DMIX stream has a more complicated structure, which includes a large cluster (C1), noise (10% of the overall data records), and overlapping clusters. The five DMIX clusters are pumped into the data stream sequentially: C1 is in the first window, C2&3 are in the second window, and C4&5 in the third. With non-decaying HE-Tree, we see that the first two windows can clearly identify the change of major clustering structure, although considerable amount of noise exists (Fig. 21). When it comes to the third window, the new two clusters C4 and C5 are overlapped by the old cluster C_1. The corresponding BKPlot shows some changes of the peak values but the best clustering structure is still at $K = 2$ and $K = 3$. This sequential cluster overlapping problem may delay the prompt decision making. To further address this problem, we turn to the discussion on the time-decaying structure.

6.4 Effect of the time-decaying HE-Tree

To further study the properties of time-decaying HE-Tree, we start with a simple case. We simulate a scenario with two consecutive windows based on the $DS1$ data structure as Fig. 22 shows. The simulated data has 40 columns and each column can have 6 distinct categorical values. In the window i, we have two clusters: C1 has non-zero values on the first 20 columns and C1 has non-zero values on the last 20 columns. In window $i + 1$ a stream is fed in with two clusters, C3 has non-zero values on the first 10 columns and C4 on the second 10 columns. We fix the size of clusters in window i: C1 and C2 have 5K records, respectively. The tree parameters are also fixed crossing windows, $f = 15$ with the height $h = 2$.

In the first set of experiments, we change the decaying rate λ from 0.1 to 0.7 while keeping the stream fed into

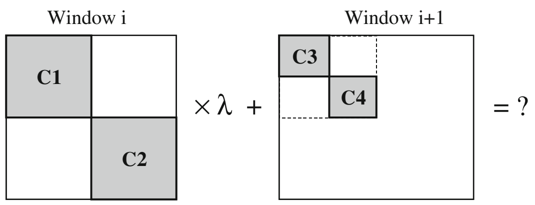


Fig. 22 Experiment with two consecutive windows

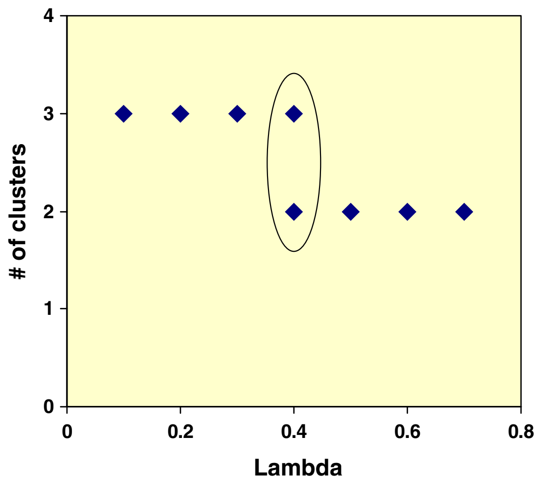


Fig. 23 Each cluster in window $i + 1$ has $2K$ records. The detected best number of clusters changes with different decaying rate λ

window $i + 1$ unchanged, with $2K$ records in clusters C3 and C4, respectively. Figure 23 shows the detected number of clusters in window $i + 1$ with respect to different settings of decaying factor λ. With a larger λ, $\lambda = 0.5, 0.6, 0.7$, the clustering structure of window i plays a dominant role in window $i + 1$ as Fig. 24 shows. Starting at $\lambda = 0.4$ the effect of clustering structure at window i is reduced to a similar level of the new clustering structure at window $i + 1$, which shows a two-layer structure (Fig. 25). Reducing λ further, the clustering structure of window $i + 1$ dominates the result (Fig. 26).

We then increase the size of data in window $i + 1$ to $5K$ per cluster. Figure 27 shows the "switching point" happens later at $\lambda = 0.5$ and 0.6. This is consistent with the earlier discussion based on Fig. 24, higher decaying rate preserves more of the old clustering structure. When the size of new clustering structure is sufficiently large compared to that of the decayed old clustering structure, the effect of the old structure will become secondary.

Lastly, we go back to the problem of DMIX stream to show how time-decaying structure helps. Different decaying rates will certainly affect the result. Table 1 lists the detection results at sample decaying rates at window 3. With $\lambda = 0.7$, C1 still plays the dominating role, so the best Ks are as same as the non-decaying structure. At $\lambda = 0.5$, the C1 effect is significantly reduced, and C4 and C5 become one of the

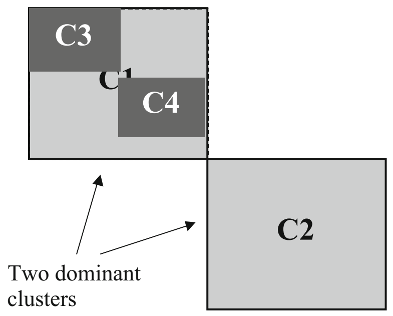


Fig. 24 With higher decaying rate, the old clustering structure dominates. Therefore, there are two clusters

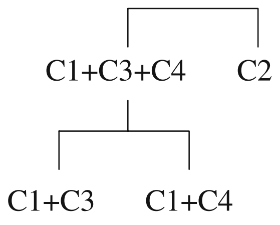


Fig. 25 At the switch point ($\lambda = 0.4$ and each cluster has $2K$ records in the new data) , a two-layer structure emerges

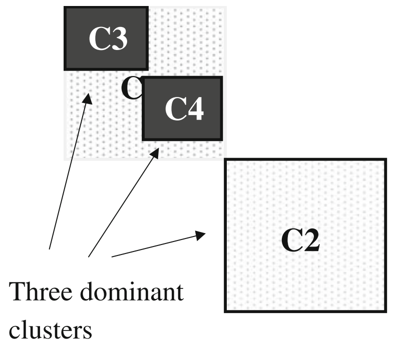


Fig. 26 With small decaying rate, the new clustering structure dominates and absorbs part of $C1$ in the old clustering structure. Therefore, there are three clusters

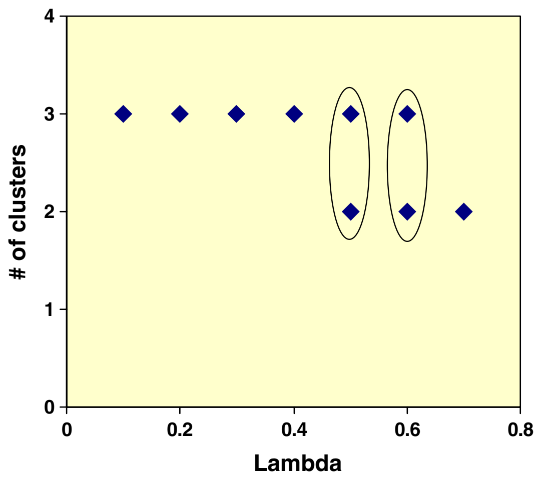


Fig. 27 Each cluster in window $i + 1$ has $5K$ records. The detected best number of clusters changes with different decaying rate λ

Table 1 Detected best number of clusters with different decaying rate λ for DMIX-stream

λ	0.7	0.5	0.2
# of clusters	2,3	2,4,5	3,4

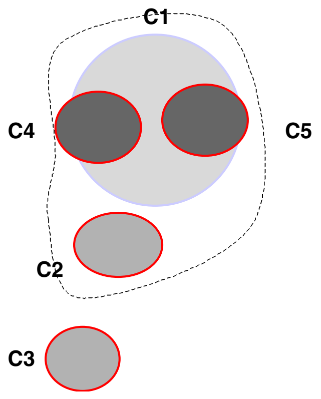


Fig. 28 C1 still affects clustering structure at $\lambda = 0.5$ at window 3 of DMIX-stream

major clustering structures ($K = 4$). Figure 28 illustrates what happens after decaying with rate 0.5. The color represents the population (density) of the records in each cluster. The darker the area is, the more records it has. C1 keeps residual effect on the C1, C2, C4, and C5 regions, which preserves the upper level structure at $K = 2$, as illustrated by the dot lined area in Fig. 28. At the same time, part of the C1 residue that is not overlapped by C4 and C5 might appear as noise to give a noisy prediction $K = 5$. Figure 29 shows the corresponding BKPlot of the clustering structure, where $K = 2$ still appears as a dominating structure.

The expected decayed structure appears at $\lambda = 0.2$, which captures the emerging structure of C4 and C5. We have discussed the similarity between decaying and sampling in Sect. 5.2. To reflect this property, the size and the density of the cluster from previous windows are further reduced (Fig. 30). Correspondingly, the BKPlot (Fig. 31) from the decaying HE-Tree suggests two candidate clustering structures with three and four clusters, respectively. This exactly describes the expected major clustering structure.

The above experiments have shown some intrinsic interactions between the decaying rate, the size of old/new clusters and the detected clustering structure. In practice, it will be delicate to set single decaying rate and window size, which are indirectly related to the cluster distribution in the stream. Multiple decaying HE-Trees with different settings might be applied to digest the same data stream.

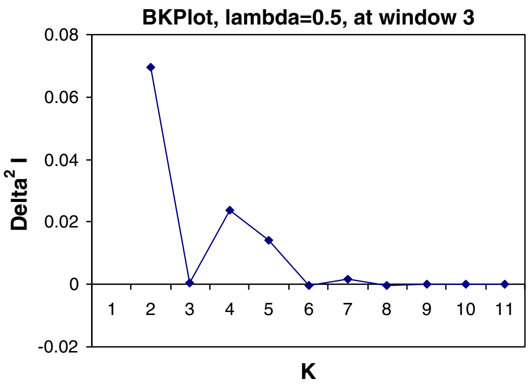


Fig. 29 BKPlot at $\lambda = 0.5$

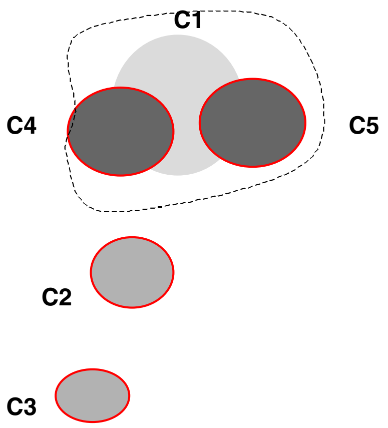


Fig. 30 Expected emerging pattern appears at $\lambda = 0.2$ at window 3 of DMIX-stream

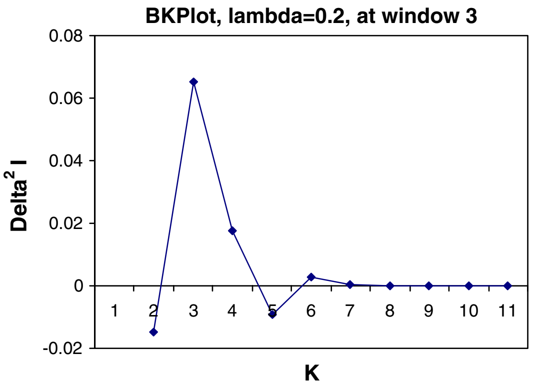


Fig. 31 BKPlot at $\lambda = 0.2$

7 Related work

While many numerical clustering algorithms [31,32] have been published, only a handful of categorical clustering algorithms appear in literature. Some are based on the design of distance between a pair of categorical records, such as, K-Modes [30], ROCK [28], and CACTUS [22]. Gibson et al. introduced STIRR [26], an iterative algorithm based on non-linear dynamical systems. A few algorithms utilize concepts in information theory. Coolcat [5] and Monte-Carlo method [34] use the same entropy criterion as we do. Cross Association [11] uses the principle of MDL to partition boolean matrix along row direction and column direction at the same time. Co-clustering [20], Information Bottleneck [37] and LIMBO [3], are based on KL divergence [18].

Clustering data streams becomes one of the important techniques for analyzing data streams [27]. In [2], a framework CluStream is proposed for clustering evolving *numerical* data streams, which mainly concerns summarizing and storing the sketch of the data stream. It uses the novel fading clustering structure. Other frameworks include the one based on velocity density estimation [1]. The problem of clustering categorical data stream was first addressed by Coolcat [5], but no more related issues such as detecting the change of clustering structure were addressed for categorical data. Recently, we propose the BKPlot method [13] that is based on the change of cluster entropy characteristics that correlates to the change of the number of clusters. Based on the BKPlot method, we proposed to detect the change of clustering structure in data streams [14], which covers the basic concepts of the HE-Tree. In this journal paper, we present more details of HE-Tree, the framework for change detection, and experimental study. Most importantly, we develop the new time-decaying structure and perform the corresponding experimental study to address the effect of stocked historical data records in the HE-Tree.

Supervised learning from data streams, or incremental or online learning algorithms is another category of stream mining algorithms, which assumes the data stream contains both feature vectors and labels for prediction. Typical algorithms include regression on multidimensional data streams [16] and incremental decision tree algorithms [21,25,38]. Similar to the change of clustering structure in data streams, the underlying classification/regression models may also change, which is often called "concept drifting". There are algorithms proposed to address this problem, such as ensemble classifiers for capturing drifting concepts [39].

In addition, nonparametric testing is used to monitor the information change in data streams [6], as well as various statistical testing [23,24]. Other loosely related work includes indexing categorical data with KL divergence as the distance function [36] and efficiently calculating entropy of a data stream [10].

Note that data stream clustering has very different setting from clustering times series data [33]. If we treat a data stream as a table with fixed number of columns but endless number of rows, clustering times series data tries to find column similarity between same or different periods, while data stream clustering tries to find row similarity.

8 Conclusion

In this paper, we address the problem of detecting the change of clustering structure in categorical data streams with a novel framework. The key of the framework is the combination of BKPlot method and Hierarchical Entropy Tree (HE-Tree) summarization structure and its algorithms. The HE-Tree is designed as a memory-efficient structure—the tree is usually a short tree (height = 2 or 3) with a small number of leaf nodes, which store the information of summarized sub-clusters. In order to observe the change of clustering structure, snapshots of leaf entries are dumped in certain time intervals, which is then processed by the extended ACE clustering algorithm to generate high-quality approximate BKPlots, with which we can easily identify whether and how the clustering structure in the stream is changed. A time-decaying HE-tree is also proposed to appropriately discount the historical clustering structure and to make the framework more sensitive to recently emerging clustering structures. Experiments show that with the HE-Tree and the BKPlot method we can effectively detect the change of critical clustering structure, which is indicated by the best number of clusters, in categorical data streams. There are still outstanding research issues on how to set the appropriate window sizes and decaying rates to adapt to different types of clustering structures.

Acknowledgments This research is partially supported by NSF CNS, NSF CCR, NSF ITR, DoE SciDAC, DARPA, Georgia Tech CERCS Research Grant, IBM Faculty Award, IBM SUR grant, HP Equipment Grant, and LLNL LDRD.

References

1. Aggarwal, C.C.: On change diagnosis in evolving data streams. IEEE Trans. Knowl. Data Eng. **17**, 5 (2005)
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A framework for projected clustering of high dimensional data streams. In: Proceedings of Very Large Databases Conference (VLDB) (2004)
3. Andritsos, P., Tsaparas, P., Miller, R.J., Sevcik, K.C.: Limbo: scalable clustering of categorical data. In: Proceedings of International Conference on Extending Database Technology (EDBT) (2004)
4. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of ACM Conference on Principles of Database Systems (PODS) (2002)
5. Barbara, D., Li, Y., Couto, J.: Coolcat: an entropy-based algorithm for categorical clustering. In: Proceedings of ACM Conference on Information and Knowledge Management (CIKM) (2002)
6. Ben-David, S., Gehrke, J., Kifer, D.: Detecting change in data stream. In: Proceedings of Very Large Databases Conference (VLDB) (2004)
7. Bock, H.: Probabilistic aspects in cluster analysis. In: Conceptual and Numerical Analysis of Data. Springer, Berlin (1989)
8. Brand, M.: An entropic estimator for structure discovery. In: Proceedings Of Neural Information Processing Systems (NIPS), pp. 723–729 (1998)
9. Celeux, G., Govaert, G.: Clustering criteria for discrete data and latent class models. J. Classif. (1991)

10. Chakrabarti, A., Cormode, G., McGregor, A.: A near-optimal algorithm for computing the entropy of a stream. In: Annual ACM-SIAM Symposium on Discrete Algorithms (2007)
11. Chakrabarti, D., Papadimitriou, S., Modha, D.S., Faloutsos, C.: Fully automatic cross-associations. In: Proceedings of ACM SIGKDD Conference (2004)
12. Chen, K., Liu, L.: VISTA: Validating and refining clusters via visualization. Inf. Vis. 3 **4**, 257–270 (2004)
13. Chen, K., Liu, L.: The "best k" for entropy-based categorical clustering. In: Proceedings of International Conference on Scientific and Statistical Database Management (SSDBM), pp.253–262 (2005)
14. Chen, K., Liu, L.: Detecting the change of clustering structure in categorical data streams. In: SIAM Data Mining Conference (2006)
15. Chen, K., Liu, L.: iVIBRATE: Interactive visualization based framework for clustering large datasets. ACM Trans. Inf. Syst. **24**, 2 (2006)
16. Chen, Y., Dong, G., Han, J., Wah, B.W., Wang, J.: Multidimensional regression analysis of time-series data streams. In : Proceedings of Very Large Databases Conference (VLDB) (2002)
17. Cheng, C.H., Fu, A. W.-C., Zhang, Y.: Entropy-based subspace clustering for mining numerical data. In: Proceedings of ACM SIGKDD Conference (1999)
18. Cover, T., Thomas, J.: Elements of Information Theory. Wiley, NY (1991)
19. Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: Proceedings of ACM SIGMOD Conference (2003)
20. Dhillon, I.S., Mellela, S., Modha, D.S.: Information-theoretic co-clustering. In: Proceedings of ACM SIGKDD Conference (2003)
21. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of ACM SIGKDD Conference, pp. 71–80 (2000)
22. Ganti, V., Gehrke, J., Ramakrishnan, R.: CACTUS-clustering categorical data using summaries. In: Proceedings of ACM SIGKDD Conference (1999)
23. Ganti, V., Gehrke, J., Ramakrishnan, R.: Demon: Mining and monitoring evolving data. IEEE Trans. Knowl. Data Eng. **13**, 1 (2001)
24. Ganti, V., Gehrke, J., Ramakrishnan, R., Loh, W.: A framework for measuring differences in data characteristics. J. Comput. Syst. Sci. **64**, 3 (2002)
25. Gehrke, J., Ganti, V., Ramakrishnan, R., Loh, W.-Y.: BOAT—optimistic decision tree construction. In: Proceedings of ACM SIGMOD Conference, pp. 169–180 (1999)
26. Gibson, D., Kleinberg, J., Raghavan, P.: Clustering categorical data: An approach based on dynamical systems. In: Proceedings of Very Large Databases Conference (VLDB), pp. 222–236 (2000)
27. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE Trans. Knowl. Data Eng. 15 (2003)
28. Guha, S., Rastogi, R., Shim, K.: ROCK: A robust clustering algorithm for categorical attributes. Inf. Syst. 25 **5**, 345–366 (2000)
29. Halkidi, M., Batistakis, Y., Vazirgiannis, M.: Cluster validity methods: Part I and II. SIGMOD Record 31 **2**, 40–45 (2002)
30. Huang, Z.: A fast clustering algorithm to cluster very large categorical data sets in data mining. In: Workshop on Research Issues on Data Mining and Knowledge Discovery (1997)
31. Jain, A., Murty, M. Flynn P.: Data clustering: A review. ACM Comput. Surv. **31**, 264–323 (1999)
32. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice hall, New York (1988)
33. Keogh, E., Lin, J., Truppel, W.: Clustering of time series subsequences is meaningless: Implications for previous and future research. In: Proceedings of International Conference on Data Mining (ICDM) (2003)
34. Li, T., Ma, S., Ogihara, M.: Entropy-based criterion in categorical clustering. In: Proceedings of International Conference on Machine Learning (ICML) (2004)
35. Meek, C., Thiesson, B., Heckerman, D.: The learning-curve sampling method applied to model-based clustering. J. Mach. Learn. Res. **2**, 397–418 (2002)
36. Singh, S., Mayfield, C., Prabhakar, S., Shah, R., Hambrusch, S.: Indexing uncertain categorical data. In: Proceedings of IEEE International Conference on Data Engineering (ICDE) (2007)
37. Tishby, N., Pereira, F.C., Bialek, W.: The information bottleneck method. In: Proceedingd of the 37-th Annual Allerton Conference on Communication, Control and Computing (1999)
38. Utgoff, P.E.: Incremental induction of decision trees. Mach. Learn. **4**, 161–186 (1989)
39. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept drifting data streams using ensemble classifiers. In: Proceedings of ACM SIGKDD Conference (2003)
40. Yang, Y., Pedersen, J.O.: A comparative study on feature selection in text categorization. In: Fisher, D.H. (ed.) Proceedings of ICML-97, 14th International Conference on Machine Learning (Nashville, US, 1997), pp. 412–420. Morgan Kaufmann, San Francisco (1997)