J Intell Inf Syst (2006) 27: 215–242
DOI 10.1007/s10844-006-9951-9

Mining changing regions from access-constrained snapshots: a cluster-embedded decision tree approach

Irene Pekerskaya · Jian Pei · Ke Wang

Abstract Change detection on spatial data is important in many applications, such as environmental monitoring. Given a set of snapshots of spatial objects at various temporal instants, a user may want to derive the changing regions between any two snapshots. Most of the existing methods have to use at least one of the original data sets to detect changing regions. However, in some important applications, due to data access constraints such as privacy concerns and limited data online availability, original data may not be available for change analysis. In this paper, we tackle the problem by proposing a simple yet effective model-based approach. In the model construction phase, data snapshots are summarized using the novel *cluster-embedded decision trees* as concise models. Once the models are built, the original data snapshots will not be accessed anymore. In the change detection phase, to mine changing regions between any two instants, we compare the two corresponding cluster-embedded decision trees. Our systematic experimental results on both real and synthetic data sets show that our approach can detect changes accurately and effectively.

Keywords Data mining · Change mining · Clustering · Decision trees · Spatial data mining · Change detection · Access-constrained data sets

Irene Pekerskaya's and Jian Pei's research is supported partly by National Sciences and Engineering Research Council of Canada and National Science Foundation of the US, and a President's Research Grant and an Endowed Research Fellowship Award at Simon Fraser University. Ke Wang's research is supported partly by Natural Sciences and Engineering Research Council of Canada. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

I. Pekerskaya (✉) · J. Pei · K. Wang
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6
e-mail: ipekersk@cs.sfu.ca

J. Pei
e-mail: jpei@cs.sfu.ca

K. Wang
e-mail: wangk@cs.sfu.ca

1 Introduction

Detecting and analyzing changes on multidimensional spatial data is critical for many applications. Particularly, mining changing regions, which are areas significant in population, and whose dominant data classes are changed between two data sets, is often interesting and informative.

For example, changes in the distribution of plant species can be monitored by taking snapshots over time. Mining changing regions from those temporal and spatial data sets is essential for understanding environmental changes. To analyze the changes, a user may want to compare two snapshots and capture the changing regions.

Change detection has been studied extensively for a long time in both statistics and machine learning. While a survey of the related work will be presented in Section 2, most of the previous studies focused on *measuring the global difference* between two data sets rather than identifying the regions of changes. A few methods that address the problem of mining changing regions implicitly or explicitly assume that *original data sets can be accessed* to identify and evaluate changing regions.

In many applications, original data sets may not be available at the time of change analysis. For example, due to the privacy and information protection concerns, owners of some census or customer data sets may be unwilling to release the data sets for direct analysis. Moreover, in applications of mining large and fast evolving data, when a user wants to compare the scenarios at two instants in history, it might be impossible (or, at least, very costly) to recall the two historical data sets. In those applications, the existing change detection methods requiring accesses to original data sets may not be applied. Those applications naturally raise a challenge: *With access constraints, can we still mine changing regions effectively?*

In this paper, we study the problem of mining changing regions from access-constrained data sets. Essentially, given a set of spatial data snapshots taken at various temporal instants, we want to build a database of summaries of the snapshots so that when a query on the changing regions between any two snapshots is raised, we can mine the changes and return the answer effectively and efficiently. We make the following contributions:

- We develop a two-phase framework for changing region detection from access-constrained data sets.
- We propose a novel model called *cluster embedded decision trees* (CEDT for short) to summarize every snapshot. The model carries concise information for comparison of the original data sets.
- We develop a method to compare two cluster-embedded decision trees. The method can identify changing regions accurately. A heuristic method effective in practice is proposed to estimate the population of regions.
- We empirically evaluate our cluster-embedded decision tree method using both synthetic and real data sets. Our systematic performance study shows that the method is effective in both accuracy and recall, and is scalable for large data sets.

The remainder of the paper is structured as follows. In Section 2, we present an overview of related work. In Section 3, a novel two-phase change detection method is developed. The experimental results on both synthetic and real data sets are shown in Section 4. Section 5 concludes the paper.

2 Related work

Change detection has been studied extensively for a long time in both statistics and machine learning. Many previous studies try to compute the difference between two data sets. For example, the Hausdorff distance (Rote, 1991), which is often used for image matching, is defined as the maximum distance between any point in one set and its nearest point in the other set. A problem with the Hausdorff distance is that it is very sensitive to extreme points. In other words, it does not take into account the overall structure of the entire data sets. In addition to the Hausdorff distance, many approaches tried to define similarity between two point sets for different applications, such as the surjection measure, the fair surjection measure, the minimum link distance, etc (Eiter & Mannila, 1997).

In the field of information theory, relative entropy, or the Kullback Leibler (K–L) divergence (Cover & Thomas, 1991), has been suggested as an appropriate measure for comparing discrete data distributions. The K–L distance between two distributions with probability functions p_k and q_k is defined as $D(p|q) = \sum_k p_k \log \frac{p_k}{q_k}$.

The major difficulty with this measure is how to estimate the distribution for high-dimensional data. Without prior knowledge about distribution, we usually estimate by counting the frequency of each data point. This means that we may need a huge amount of data in order to get some statistically meaningful estimation.

If the object identities are available, i.e., we can trace the occurrences of the same object in two data sets, we can compute the distance between its occurrences. The similarity of two data sets can be defined as the sum of the pairwise label distances. Moreover, for objects appearing only in one data set, we can use interpolation to predict unknown values from values observed at known locations. One critical issue is to assign proper weights to objects so that the sum of distances can reflect the distance between two data sets appropriately. The Kriging method (Oliver & Webster, 1990) developed in the field of geostatistics uses semivariogram to assign weights. Semivariogram characterizes the spatial continuity roughness of the data set. Kriging is superior to other interpolation methods because it provides an optimal interpolation estimate for a given coordinate location.

However, all of the above approaches try to identify *global changes*. That is, they try to measure the differences between data sets. In this paper, we are concerned with identifying regions of changes instead of measuring the global difference.

In the context of association rule mining (Agrawal, Imielinski, & Swami, 1993; Agrawal & Psaila, 1995) addresses the problem of monitoring the support and confidence of association rules. Given an association rule, the techniques track the support and confidence variations of the rule over time. The discovered rules from different time periods are collected into a rule base. Changes in support and confidence over time, called history, are defined using specific shape operators. The user can then query the rule base by specifying some history specifications.

In Liu, Hsu, and Ma (2001), fundamental rule changes are obtained by pruning "redundant rules". That is, they report only changes that cannot be explained by the presence of other changes. The algorithm only considers changes in support or confidence of the rules that are not direct consequences of changes in the conditions of the rules. Therefore, many interesting changes may be missing.

None of those studies deal with the classification problem where changes should be extracted with respect to the changes in class label.

To the best of our knowledge, there are three existing studies that are most related to ours. In (Ganti, Gehrke, & Ramakrishnan, 1999), Ganti et al. developed a framework for measuring the deviation between two data sets in terms of the classifiers they induce. The change is measured by the amount of work required to transform them into some common specialization. More precisely, the deviation between two data sets D_1 and D_2 is computed as follows. The decision tree is viewed as a set of regions associated with the leaf nodes. To compare two models, sets of regions for two decision trees are made identical. They are refined to the finer partition obtained by overlaying the two partitions of the attribute space. Next, the deviation is computed between D_1 and D_2 with respect to each region in that partition. To do this, each region is associated with a *measure* reflecting the fraction of tuples in the data set that maps into it. Then, the deviation between D_1 and D_2 is computed by summing up the deviations of all regions in the refined set of regions. Different measures of the deviation are considered in the paper, such as the misclassification rate and the chi-squared metric. The computation requires accessing original data sets.

While the method determines whether the changes between two data sets exist and how significant the changes are, it does not provide an efficient way of identifying changing regions. In other words, additional techniques are needed to obtain the description of changes, which is the topic of this paper.

In (Liu, Hsu, Han, & Xia, 2000), Liu et al. detected changes by requiring the old decision tree to be similar to the new one. To compare two data sets D_1 and D_2 with a decision tree on D_1 available, a new decision tree on D_2 is constructed such that it uses the same attributes and splitting points as the decision tree on D_1. It composes a severe restriction that does not allow us to compare arbitrary two models. For example, if important changes occur at the top levels of the decision tree, the method cannot be used. It is not applicable either in the situations where the original data sets are not available for direct analysis.

In (Wang, Zhou, Fu, & Yu, 2003), Wang et al. transformed a decision tree to a set of rules. Then, the change mining problem is reduced to characterize how well the set of rules obtained from an old data set fits a new data set. They proposed a four-step approach. First, a decision tree is built on the new data set. Second, for each sample in the new data set, the corresponding rules in both the old decision tree and the new decision tree are identified. Third, for each old rule, the corresponding new rules are found which classify the same set of objects, and the quantitative change is estimated. Last, the changes are presented as a comparison (a pair) of rules on the old and the new data sets. Again, such a method has to access the original data sets, and thus cannot handle the data sets with access constraints.

This paper is also remotely related to symbolic data analysis. In (Billard & Diday, 2003), Billiard and Diday describe "symbolic" data that is more complex than standard data. Instead of the single value, it can be represented by lists, intervals, distributions and the like. They define 'Symbolic Data Analysis" as the extension of standard data analysis in order to work with symbolic data. In (Diday & Esposito, 2003), Diday provides an overview on recent development in this area, as well as presents some tools and methods to work with symbolic data.

3 Cluster-embedded decision trees

Most of the previous change detection methods described in Section 2 require accesses to the original data sets. In this section, we propose a framework that is

able to mine changes from data with access constraints. The framework, as shown in Fig. 1, is in two phases: model construction and change detection from models.

- In the phase of model construction, every data set that may be used later for change detection is summarized using a cluster-embedded decision tree developed in this section. This model construction phase can be conducted at the data owner's site. Only the models are released for data analysis.
- In the phase of change detection, we detect changing regions between any two data sets specified by the user using the models constructed in the first phase. At this stage, we do not need to access the original data.

The details of the method are presented in this section. In Section 3.1, we define the problem precisely. Section 3.2 addresses what models should be used in the model construction phase. Section 3.3 describes two approaches used in the change detection phase.

3.1 Problem description

Generally, we consider a spatial data set as a set of objects in a multidimensional, numeric space D. Moreover, each object belongs to a pre-defined class. For an object o, $o.A_i$ denotes the value of o on attribute A_i and $o.Class$ denotes the class to which o belongs. Here, we assume that populations in D_1 and D_2 may be different and object identities are not available.

A *hyperrectangle region*, or *region* for short, is a generalization of a rectangle to n dimensional space, defined by the upper and the lower bounds for each dimension. Formally, a hyperrectangle region $R = ((l_1, u_1), ..., (l_k, u_k))$, where $-\infty \le l_i \le u_i \le \infty$ for $1 \le i \le k$ and k is the dimensionality. $R.A_i = (l_i, u_i)$ is the scope of region R on attribute A_i.

The *population* of R with respect to data set D is the number of objects in D that fall into R, denoted by $pop_D(R)$. Moreover, the population of R with respect to class C is the number of objects from class C in D that fall into R, denoted by $pop_D^C(R)$. Clearly, we have $pop_D(R) = \sum_{C_i} pop_D^{C_i}(R)$. A region is called *dominated* by a class C if the population of C in R is larger than the population of any other classes.

To capture the distribution of classes over space, we can take snapshots at various time instants. Each snapshot is a spatial data set. Let D_1 and D_2 be two spatial data sets at two instants. Intuitively, a changing region from D_1 to D_2 is dominated by objects from different classes in the two data sets. To avoid triviality, such changing regions should have enough population.

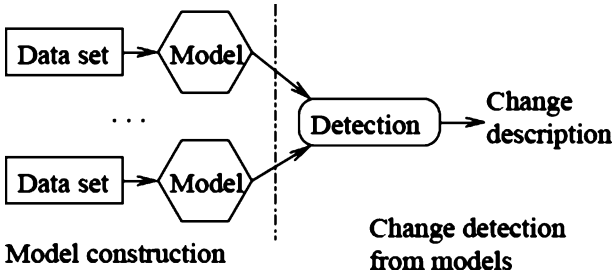


Fig. 1 The framework of model-based change detection construction phase

Definition 3.1 (Changing region) Given two data sets D_1 and D_2 and a minimum population threshold *min_population*, a region R is called a changing region from D_1 to D_2 if (1) R is dominated by class C in D_2 but not in D_1; (2) $pop_{D_1}(R) \geq min\_population$; and (3) $pop_{D_2}(R) \geq min\_population$.

The problem of *mining changes* is to identify changing regions as accurately as possible.

3.2 Model construction phase

3.2.1 Choosing a model for changing region detection

Classification and clustering are two categories of popularly used models in machine learning. It is natural to ask whether we can directly borrow them for changing region detection. One intuitive idea is to build a decision tree for each dataset and compare the two decision trees.

Example 3.1 (Change detection by decision tree comparison) Consider two data sets D_1 and D_2, each has two attributes and two classes. We build decision trees on D_1 and D_2 as shown in Fig. 2(a) and (b), respectively. After a tree is built, the data is partitioned into disjoint regions in the attribute space. The corresponding partitions are shown in Fig. 2(c) and (d), respectively.

Generally, a decision tree partitions the space of a data set into hyper-rectangles. An intuitive way of identifying areas of changes is to simply overlay the hyper-rectangles from the two decision trees, as exemplified in Fig. 2. A hyper-rectangle is reported as a changing region if it carries different class labels in the decision trees under comparison. In this example, the two rectangles on the left are reported as changing regions.

However, if we take a closer look at the data sets, we can find that the changes derived by the direct comparison may not be accurate, or even may not be right. Consider the north-western rectangle. There is no change between the two data sets in this area. It just happens the label of the rectangle is affected by different neighbour regions because it has only very few objects. In other words, this rectangle should not be reported.

On the other hand, the south-western rectangle is also reported as a changing region from class $Class_1$ to class $Class_3$ in Fig. 2(e). However, within the rectangle, only a new group of objects in $Class_3$ appear. There are in fact no objects in $Class_1$ in the region in D_1. Therefore, reporting this as a changing region is inaccurate. Instead, a smaller region ([0.2, 0.4], [0.1, 0.5] should be reported, as shown in Fig. 2(f).

Why directly comparing decision trees may not be accurate or even may not be correct? First, in decision tree construction, the space is divided into hyper-rectangles recursively until the resulted hyper-rectangles are pure or the population is less than a user-specified threshold. Therefore, the information about the data distribution can be largely lost and cannot be recovered for comparison. Second, the order of dimensions used in decision tree constructions is easily different from one data set to another. With minor changes in the data set, or even with minor changes in parameters, two differently structured decision trees may be produced. Some sparse and noisy regions in the data sets with no changes at all may be bound to some other regions and thus be labelled differently in different trees. That may

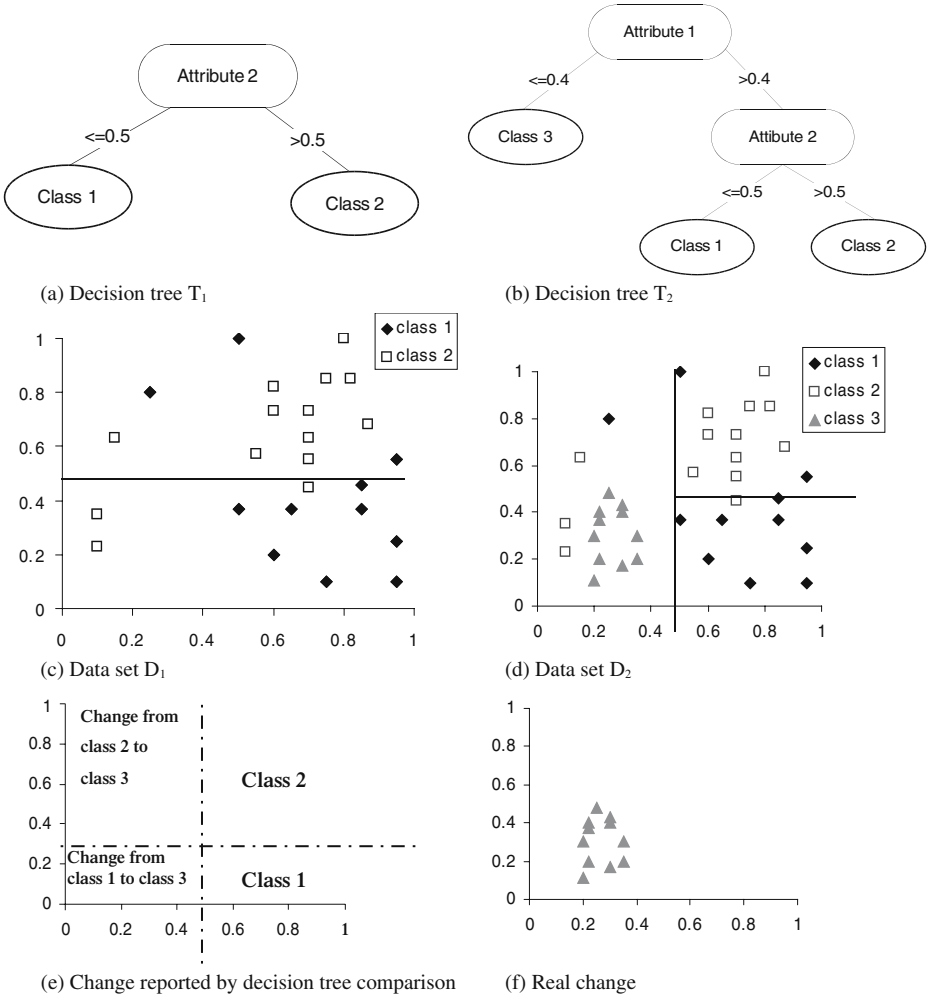


Fig. 2 Two data sets, the decision trees and the changes

introduce false changing regions in the tree comparison. In other words, decision tree models do not contain enough information to distinguish such cases.

There exist some other classification models, such as neural networks and support vector machines. In terms of changing regions detection by model comparison, they suffer from similar problems as decision trees or may be even worse. For example, it is hard to compare two neural networks and a naïve comparison is hard to understand due to the weak understandability of this model.

Clustering models are another category of popular models in machine learning. However, global clustering models cannot serve the tasks of changing region detection due to the following reasons. First, global clustering typically needs some background information about the distribution of the data, such as the number of clusters. This parameter can strongly affect the efficiency of the clustering. Second, clustering is often costly and does not scale well with respect to dimensionality.

Another intuitive approach to change mining is to construct a grid. Assume for each dataset the space is divided into cells of the same sizes. Inside each cell we get

the population for each class. Now we can compare the class distributions of two corresponding grids from two different datasets. There are several problems with this approach as well. First, in high dimensional space this method has a huge space complexity. Second, this method strongly depends on the granularity of the grid. If the grid is too fine, many small changes of little use can be found. If the grid is too coarse, it doesn't provide much insight to the data, because each cell contains a mixture of objects from different classes.

One may wonder whether we can build a spatial index such as an R-tree to index the objects and find the changes using the index. Two obstacles exist for this kind of approaches. First, it is hard to control the purity of objects in bounding boxes in the spatial index. Thus, using the bounding boxes to find changes is difficult. Second, with access constraints, we are not allowed to index the spatial objects directly.

Therefore, we need to develop a new type of model for our change detection task.

3.2.2 Cluster-embedded decision trees (CEDT)

Although decision trees cannot be used directly for model-based changing region detection, heuristically it can help us to identify relatively pure regions dominated by some classes. The major deficiency of decision trees for our task is that they cannot capture the spatial distribution of the objects in leaf nodes. While the decision trees cannot maintain information about local distributions well, clustering can provide good summarization of local distribution. On the other hand, while clustering often requires global background knowledge to perform well, decision trees carry such information naturally (each leaf node is a relatively pure area). This leads us to integrate the two types of models and design the cluster-embedded decision trees (CEDT for short).

The general idea is as follows. *For a data set, we construct a decision tree. To capture the spatial distribution of the objects in leaf nodes, we construct clusters and keep the information on those clusters at leaf nodes.* The details of the CEDT construction are presented in the following sections.

3.2.2.1 Decision tree construction Given a data set, to construct a decision tree that will be later extended to embed clusters, we can adopt any decision tree construction algorithm. The only requirement is that any node containing less than *min_population* (i.e., the minimum population threshold) objects should not be split any more, because the corresponding region cannot contain any changing regions. In a data set D, a leaf node V in a decision tree may be in one of the following two cases.

- Case 1: Every class has a population lower than the user-specified minimum population threshold. The hyper-rectangle corresponding to this node is considered sparse and statistically insignificant. Such a node V cannot contain any changing regions from the other data set to D. Thus, we only keep the population of objects with respect to various classes at the node, as a usual decision tree does.
- Case 2: At least one class has a population higher than or equal to *min_population*. Then, the corresponding node can contain changing regions. Thus, we will capture the local distribution of objects in various classes using clustering.

3.2.2.2 Clustering Recall that we are not allowed to store the details about all objects due to access constraints. However, the output of any decision tree algorithm (for example C4.5) does not provide enough information to determine the changes. The information at a leaf node is often insufficient to indicate the changes in an informative way. To summarize the objects, a natural method is to construct clusters and keep only the summary of the clusters.

Generally, we can use any clustering methods to construct clusters at leaf nodes for a cluster-embedded decision trees. This gives CEDT the flexibility to meet the requirements in various applications.

As a simple yet effective solution we discuss the case where K-means clustering is used to form clusters. The problem now is to determine the number of clusters. Typically, without background knowledge and many trials, a number leading to the optimal clustering is hard to find.

However, clustering within the leaf node may be somewhat easier. Recall that at a leaf node most objects belong to a dominant class. Therefore, comparing to global clustering where the data objects are generally mixed, the clustering quality is less sensitive to the number of clusters.

In the cluster-embedded decision tree design, we adopt a heuristic method to set the number of clusters for a leaf node. The intuition is that the number of clusters is proportional to the population of objects in the leaf node.

Technically, let *min_pop* be a user-specified minimum population threshold. It should be selected so that any clusters with the population less than *min_pop* are considered insignificant and can be ignored from data analysis.

Consider a leaf node with n data objects, where the objects are in classes C_1, ..., C_k, and the number of objects in the classes are n_1, ..., n_k, respectively. For class C_i, we form up to $\left\lfloor \frac{n_i}{min\_pop} \right\rfloor$ clusters to capture the distribution of objects in the class. Clearly, for a node with n objects, at most $\left\lfloor \frac{n}{min\_pop} \right\rfloor$ clusters are formed. An object must be in one and only one leaf node of a decision tree. Thus, we have the following claim.

Lemma 3.1 (Number of Clusters in CEDT) For a data set of m objects and a minimum population threshold *min_pop*, there are at most $\left\lfloor \frac{m}{min\_pop} \right\rfloor$ clusters at leaf nodes of the tree.

3.2.2.3 Cluster representation Now, the problem is how the clusters should be represented. Typically, K-means returns clusters as hyper-spheres. However, it is not convenient for our purpose. To determine changing regions, it is often required to compare two regions and compute the overlapping populations. Computing the intersection of two hyper-spheres in a high-dimensional space is far from trivial. Thus, it is highly desirable that the regions are in regular shapes, such as hyper-rectangles whose edges are parallel to the dimensions. Now we have to find a way to use hyper-rectangles determined by some simple parameters to record the scope of a cluster.

Here, we propose a cluster representation in hyper-rectangles. Technically, for a cluster of objects, we keep the mean of the cluster and the standard deviation d_i on each dimension. We approximate the cluster by a hyper-rectangle with the mean and the edge $2 \cdot t \cdot d_i$ on each dimension, where t is a small number greater than 1. The quality of the approximation is guaranteed by the following result.

Theorem 3.1 (Cluster representation) *In a k-dimensional space (R_1, ..., R_k), suppose that dimensions are statistically independent. Let S be a set of n objects, whose mean is (c_1, ..., c_k) and the standard deviation on each dimension is d_i. Then, for any t >1, in expectation, at least* $n \cdot \left(1-\frac{1}{t^2}\right)^k$ *data objects appear in the hyper-rectangle* ($[c_1 - t \cdot d_1, c_1 + t \cdot d_1], ..., [c_k - t \cdot d_k, c_k + t \cdot d_k]$).

Proof According to Chebyshev's inequality (Papoulis, 1984), the probability that a random variable differs from its expectation by $t \cdot d$ or more cannot exceed $\frac{1}{t^2}$, where d is the standard deviation and t is a number greater than 1. Therefore, the probability that an object is in the hyper-rectangle in the theorem is $\left(1-\frac{1}{t^2}\right)^k$. The theorem follows. ■

Theorem 3.1 provides a lower bound for the quality of summarizing a cluster using hyper-rectangles. Interestingly, as shown in the experimental results, the summarization quality is fairly good in practice. That is, with a small t value, we can obtain good accuracy and recall.

To compute hyper-rectangles for each cluster in the node we should store the mean (i.e., the center) and the variance of objects. Also, we should keep the number of objects in each cluster.

The overall algorithm used for cluster-embedded decision tree construction is described in Fig. 3.

3.3 Mining changing regions

Cluster-embedded decision trees provide enough information for mining changes. We propose two different approaches to use CEDTs: *leaf-based* and *cluster-based.* The leaf-based approach returns a small amount of broad regions of changes. It provides high accuracy, but overlooks smaller regions. The clusters-based approach, on the other hand, is able to find many subtle changes between the data sets, while the result is not as accurate as the leaf-based method. Depending on the preferences of the user, one of the methods can be chosen for change detection.

```
Input: A dataset D1, min_p threshold
Output: CEDT T1
Method:
   1:    build a decision tree, no nodes with population less than min_p
         should be split further, where min_p is the minimum population
         threshold;
   2:    for each leaf node in the tree that has at least min_p data
         objects, do
   3:      for each class in the leaf node that has at least min_p
             objects, do
   4:        form ⌈ni⌉ / min_p clusters, where ni is the number of objects of
             the class at the leaf node;
   5:    compute and store the number of objects, the mean and the
             variance of each cluster.
```

Fig. 3 The CEDT construction algorithm

3.3.1 Leaf-based changing regions

The general idea of this approach is the following. First, we compare two decision trees to find the differences between them. After that, we use the detailed information from CEDTs to eliminate those false changing regions. Also, we rank the regions according to the significance of the changes.

3.3.1.1 Overlay of two trees To understand this approach, let us first have a closer look at the decision tree construction. After a tree is built, the data is partitioned into disjoint regions in the attribute space. Each path from the root node to a leaf represents a *hyperrectangle region*. In a decision tree, for each dimension, the splitting points can be used to set upper and lower bounds. Notice that the procedure of constructing a tree with numeric attributes can use the same attribute for splitting several times.

Definition 3.2 (Path) A *path* from the root to a leaf node is a pair ⟨region, Class label⟩, where region is a vector of intervals for attributes. Notice that for every path we use the attributes in the same order.

For convenience, we represent a path as:

$$\langle (L_1, U_1), (L_2, U_2), \ldots, (L_n, U_n) \rangle \Rightarrow \text{Class label},$$

where L_i are the lower bounds and U_i are the upper bounds of the intervals for the corresponding attributes. If the attribute is not used for partitioning the data, we set its interval to $(-\infty, +\infty)$.

Given two decision trees, we can easily compare them by overlaying the partitions.

Definition 3.3 (Intersection) To form an *intersection* of two paths $P_1 = \{[L_1, U_1], \ldots [L_k, U_k]\}$ and $P_2 = \{[L'_1, U'_1], \ldots [L'_k, U'_k]\}$ from decision trees T_1 and T_2, respectively, we compute an intersection for each attribute: $P_1 \cap P_2 = \{[\max\{L_1, L'_1\}, \min\{U_1, U'_1\}], \ldots, [\max\{L_k, L'_k\}, \min\{U_k, U'_k\}]\}$. An intersection is empty if there exist an attribute A_i such that $\max\{L_i, L'_i\} > \min\{U_i, U'_i\}$.

Definition 3.4 (Exclusive paths) We call two paths P_1 and P_2 *exclusive* if their intersection $P_1 \cap P_2$ is empty.

Definition 3.5 (Delta region) Given data sets D_1 and D_2, a delta region is a region that has different class labels in the decision trees in D_1 and D_2.

Definition 3.6 (Overlay) An *overlay* of two decision trees D_1 and D_2 is a set of delta regions. It can be described by a set of rules:

$R \Rightarrow$ *OldClass, NewClass*, where *OldClass* and *NewClass* are class labels in D_1 and D_2, respectively, and R is a delta region. To form such an overlay the procedure in Fig. 4 is used. Using the procedure in Fig. 4, we can identify all possible changing regions by finding all the intersections between paths in the old and the new decision trees.

```
For each path Pi in first dataset
  For each path Pj in second dataset
    If Class (Pi) ≠ Class (Pj)
      If Not Exclusive(Pi, Pj)
        Overlay ← add rule(Intersection(Pi,Pj)⇒ Class(Pi), Class(Pj))
```

Fig. 4 The procedure for computing overlay

Theorem 3.2 *If decision tree T_1 has n leaf nodes, and T_2 has m leaf nodes, then the time complexity to compute the overlay of two trees is O(n*;m).*

3.3.1.2 Real and false changing regions Path-by-path comparison of two trees may enumerate many differences between the models. However, many of them can be trivial or caused by noise.

Suppose R is a candidate changing region. Two cases may happen:

- The corresponding samples in the data sets have changed their class labels
- R is a false changing region. The data distribution in R does not really change. It is assigned different class labels due to the fact that decision trees are constructed in different ways

The first case is exactly what we are looking for. However, we should eliminate cases falling into the second category. Several reasons can be found to explain the second case.

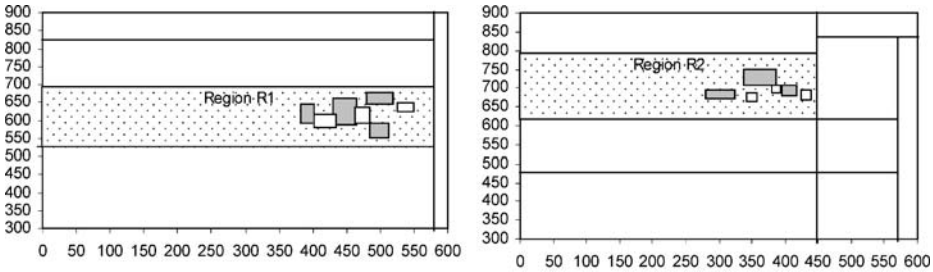
First, there might be no or very few examples in a data set covering the particular region. In this case, the region can be classified by a decision tree to either class. Therefore, if they are classified differently by decision trees from D_1 and D_2, it appears as a change but actually it is not.

Another possible reason is that some regions may be noisy, i.e., they may contain samples from different classes without any class being strongly prevalent. With changes in the neighbouring areas of the region, those noisy regions will be classified differently by two decision trees. It is reported as a change region if a path-by-path comparison is used, but actually it is not.

In summary, a naïve comparison of regions may raise false signals. Particularly, two cases may exist:

- "Empty" regions — there are very few samples in such a region. Empty regions can be identified by estimating its population and comparing it with a pre-specified threshold.
- "Noisy" regions — the regions that are "impure", i.e., there are many samples with the class label different from the prevalent one. However, impurity by itself is not what we are trying to avoid. Rather, it is just an indicator that the change between populations of dominant classes in data sets D_1 and D_2 is not strong. In the next section, we develop a measure of change significance.

3.3.1.3 Using CEDT to estimate region population Recall that given two decision trees T_1 and T_2, delta regions are obtained by intersecting the leaf nodes from T_1 and T_2. Therefore, each delta region R belongs to a node in T_1 and a node in T_2. The population of a region R in a data set can be estimated using the clusters at the leaf nodes to which R belongs. Let $R = ([x_1, y_1], ..., [x_k, y_k])$ be a region. Three cases may happen:



(a) Region R_1 classified as class 3 in D_1 (b) Region R_2 classified as class 2 in D_2

Fig. 5 Clusters summarizing objects in regions

- The leaf node has low population thus no clusters are stored. In this case, the population of R is also low and insignificant for our analysis.
- Although the leaf node has a population passing the minimum population threshold, R has no overlaps with any clusters at the leaf node. In this case, the population of R is still low and insignificant.
- R overlaps with some clusters at the leaf node. In this case, we can estimate the population of R as well as the class distribution using the clusters.

Example 3.2 (Estimating Region Population) Let us revisit the data sets described in Example 3.2. The region (524,689], (0,584] [Region R_1 in Fig. 5(a)] is classified as class 3 by decision tree on D_1, while the region (608, 781], (0, 453] [Region R_2 in Fig. 5(b)] is classified as class 2 by decision tree on D_2. Therefore, their intersection is a potential changing region. The clusters summarizing the objects in these regions are shown in Fig. 6. To estimate the population of $R = R_1 \cap R_2$, we have to check whether R has any overlaps with clusters from R_1 and R_2.

Generally, let $L = ([x_1', y_1'],..., [x_k', y_k'])$ be a cluster in D_1 with a number of objects N_L. R and L overlap if and only if $x_i' < y_i$ and $x_i < y_i'$ for all $1 \le i \le k$.

We need to estimate the population of R in D_1 of class C and the population of R in D_1 of classes other than C. Thus, for each overlapping cluster L in T_1, we compute the overlap volume, which is given by $\text{overlap}(L, R) = \prod_{1 \le i \le k} (\min\{y_i, y_i'\} - \max\{x_i, x_i'\})$.

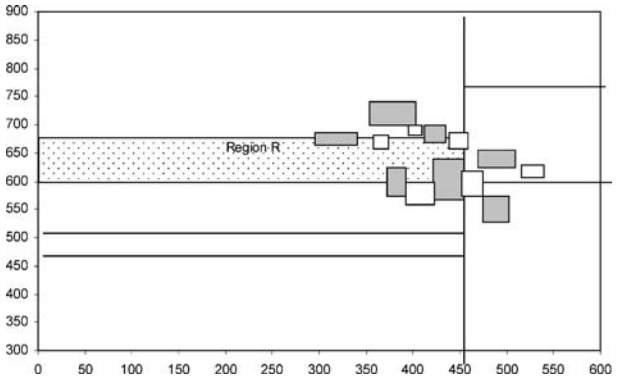


Fig. 6 Using CEDT to estimate the population of a region

To estimate the population, we assume that the objects in a cluster are evenly distributed. In a small region, uniform distribution is a good estimation, which is verified by the experimental results on both synthetic data sets and real data sets.

Therefore, the contribution in population from a cluster L *to* R in D_1 can be estimated as $\frac{\text{overlap}(L,R)\cdot N_L}{\prod_{1\le i\le k}(y_i'-x_i')}$.

By examining all clusters that have overlap with R, we can estimate the population of R in data set D_1 with respect to class C, i.e., $pop^C_{D_1}(R)$, and that with respect to classes other than C.

3.3.1.4 Presenting changing regions After we estimate the population of each class in the region, we can determine the dominant classes. By comparing those populations, we can determine whether R is dominated by the same class in D_1 as in D_2. Since we are interested in the regions that change the class labels, if the class is the same, we remove R from the list of candidate changing regions.

Also, the knowledge of population allows us to eliminate sparse changing regions. If the estimated population for all classes is smaller than minimum population threshold *min_population* in both datasets, we consider the region insignificant. In addition to eliminating sparse regions, knowing the population allows us to present the regions with higher population as more important to the user.

Now the question is how to measure the noise. As mentioned before, noise by itself is an indicator that the change is not strong. We want to measure how significant the change is between two data sets with respect to the certain region. More precisely, we use the "change distance" to measure how strongly a particular region changes. Given $P^{C_i}_{D_1}(R)$ — the probability that a point from region R belongs to class C_i in data set D_j, the distance is computed as follows:

$$\text{ChangeDist}\ (R) = \sqrt{\sum_i \left(P^{C_i}_{D_1}(R) - P^{C_i}_{D_2}(R)\right)^2} = \sqrt{\sum_i \left(\frac{pop^{C_i}_{D_1}(R)}{pop_{D_1}(R)} - \frac{pop^{C_i}_{D_2}(R)}{pop_{D2}(R)}\right)^2},$$

where $pop^{C_i}_{D_1}(R)$ and $pop^{C_i}_{D_2}(R)$ are populations of class C_i in the region R in data sets D_1 and D_2, respectively, and $pop_{D_1}(R)$, $pop_{D_2}(R)$ — populations of the region in the data sets D_1 and D_2, respectively.

While eliminating regions with low population is a straightforward solution, dealing with noise is not that easy. A naïve approach is to ask the user to set a threshold to eliminate regions where the change is not very strong. However, this parameter is not easy to set since it is hard to distinguish which level of change is "significant enough".

Therefore, instead of asking the user to define the threshold, we propose a method of ranking change regions. There may be many approaches to rank changing regions. In general, ranking functions can strongly depend on the particular application. Here, we propose one method based on the regions population and significance of changes. Intuitively, the more interesting regions have bigger population and bigger change distance. More precisely, the function to rank the regions is computed as follows:

$$\text{Rank}(R) = \left(\frac{pop_{D_1}(R)}{pop_{D_1}} + \frac{pop_{D_2}(R)}{pop_{D_2}} \right) \cdot \text{ChangeDist}(R).$$

The overall algorithm for leaf-based change detection is presented in Fig. 7.

3.3.2 Cluster-based changing regions

The regions returned by the leaf-based approach might be too rough. If the user is interested in finer details of the dataset comparison, the cluster-based approach can be used to refine the results of leaf-based approach.

In this method, we compare pairs of clusters from the old and the new data sets. We consider all clusters including those belonging to the regions classified to the same class by both models.

Suppose we are given CEDTs T_1 and T_2 built on data sets D_1 and D_2, respectively. To find changing regions, we compare clusters in CEDTs one by one.

Let $R = ([x_1, y_1], ..., [x_k, y_k])$ be a cluster in T_2. Since D_1 is not available, we use T_1, the CEDT on D_1, to estimate both the population and the dominant class of R in data set D_1. Therefore, we need to search all clusters in T_1 that overlap with R. A naïve method is to compare R with every significant cluster in T_1. Obviously, it is inefficient.

As the clusters are stored in the CEDT T_1, we can use the decision tree as an index for the clusters. That is, we allocate the leaf nodes of T_1 that have overlaps with R. Only clusters in those leaf nodes should be checked.

Technically, we start from the root of T_1, and consider all children nodes of the root. Suppose the splitting condition is on dimension D_i, and the range for the child node V is $[z, w]$, then we need to search V if and only if the range overlaps with $[x_i, y_i]$, the range of R on dimension D_i. That is, $z < x_i$ and $w \geq y_i$. All the children of

```
Leaf - based change detection:
Input: CEDTs T1 and T2
Output: a set of rules Rules, ranked from the most meaningful change
Method:

1: compute the overlay of two decision trees
2: for each delta region R
3:    Determine C1 and C2 - the dominant classes for D1 and D2;
4:    if C1 ≠ C2 then
5:     Old_Population ← estimated population in the old  data set;
6:     New_Population ← estimated population in the new
       data set;
7:      if Old_Population and New_Population are larger then min_p
        then
8:      Add rule to Rules: <Region, C1, C2>
9: for each element in the Rules
10:   Compute rank function
11:Order Rules according to the rank
12 return Rules
```

Fig. 7 The leaf-based change detection algorithm

the root node that overlap with R on dimension D_i will be checked recursively, until the leaf nodes that have overlap with R are obtained.

We retrieve all the clusters at the leaf nodes that overlap with R, and check whether the clusters overlap with R. The overlapping population can be computed in the same way as described in Section 3.3.1.3.

3.3.2.1 Summarization of significant changing regions As mentioned, cluster-based approach can identify many changing regions. Now one problem remains unsolved — how can we present the mining results effectively?

A naïve approach to present a list of all changing regions may not be good in many cases. A long list of such changes is often hard to understand. Moreover, a user may also want to know how the changing regions are distributed in the data space.

Here, we propose a simple yet effective solution: we build another decision tree on the changing regions. That is, instead of single objects we are using changing regions as training data. Each changing region R carries a label "$C \Rightarrow C'$", where C and C' are the dominant classes of region R in data sets D_1 and D_2, respectively. In this case, if we have m classes in data sets D_1 and D_2, there will be m^2 classes in the new training set. Small modifications are required in the decision tree algorithm to take regions instead of single values.

The resulted decision tree is presented as a result of cluster-based change mining. It provides an effective way to allocate small changing regions into meaningful groups.

4 Experimental evaluation

In this section, we present the experimental results obtained on both synthetic and real datasets.

All the experiments are performed on a 1600-MHz Pentium PC machine with 256 megabytes main memory, running Microsoft Windows 2000. All the programs were written in Java, and the open source software Weka (Witten & Frank, 2005) was used to facilitate the implementation of the algorithm. In particular, the decision tree C4.5 (Quinlan, 1993) and the K-means clustering algorithm were used as they were implemented in Weka.

We describe the procedure of constructing synthetic datasets and introduce a mortgage dataset from IPUMS repository in Section 4.1. Section 4.2 provides an analysis of the algorithm effectiveness with respect to various parameters. The efficiency of the method is verified in Section 4.3.

4.1 Datasets and experimental settings

4.1.1 Synthetic data

To verify if the proposed algorithm is able to find the changes accurately, we compose a data generator. The experiments consist of 4 steps:

- Generate an "old" dataset;
- Produce a new dataset, embedding changes of different types;
- Run the algorithm to build models and compare them;

- Evaluate whether the embedded changes and the obtained regions agree.

4.1.1.1 Data generator To run a generator, a user specifies the following parameters: data set size, number of dimensions and number of classes. Given the number of classes, a number of clusters are generated for each of them. Each cluster is described by its mean and standard deviation. Cluster means are uniformly distributed in the attribute space. The parameters used by data generator are summarized in Table 1.

4.1.1.2 Types of changes To test the algorithm, four types of changes are embedded:

1. *Changing class* — clusters are randomly selected and their class labels are changed.
2. *Removing a cluster* — all samples from a particular cluster are removed from a dataset.
3. *Adding a cluster* — for an arbitrary class a new cluster with random values of mean and standard deviation is added.
4. *Splitting a cluster* — a cluster is split into two clusters, one with the same class label and the other one with a different one.

4.1.2 IPUMS census data

The IPUMS dataset (Ruggles et al., 2004) contains USA census data from 1990. A random sample containing the information on 20,000 personal records is chosen in the source. The data set contains 10 numeric attributes, where "Mortgage" is chosen as the class attribute. Mortgage has 4 values, "N/A", "No, owned free and clear", "Yes, mortgage or deed of trust" and "Contract for sale". After removing all the samples with "N/A" values, 17,460 tuples remain. To set the data for the change mining task, we compare different ethnic groups, including "white", "black" and "American Indian".

4.2 Effectiveness of CEDTs

4.2.1 Synthetic data

The quality of the algorithms is measured in terms of *accuracy* and *recall*. *Recall* describes what proportion of embedded changes is captured by the algorithm.

Table 1 Parameters used by the data generator

Parameter	Distribution
Number of clusters per class	Gaussian with the mean of 15 and the standard deviation of 5
Number of examples in cluster	Gaussian with mean = $\alpha \cdot \frac{D}{C \cdot L}$, where D is a dataset size, C is an number of classes and L is the number of clusters per class
Cluster mean	Uniform in space $[0, 1{,}000]^d$, where d is the number of dimensions
Cluster standard deviation (radius)	Gaussian with the mean of 50 and the standard deviation of 15

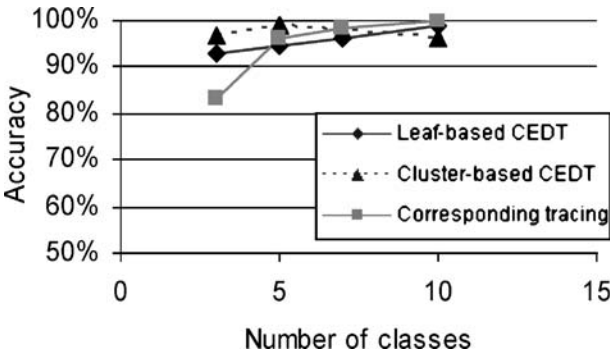


Fig. 8 Accuracy with respect to the number of classes

Accuracy represents the percentage of the embedded changing regions with respect to the number of regions detected by the algorithm.

In this section, we evaluate the effectiveness of leaf-based and cluster-based approaches. Also, we present a performance comparison with Correspondence tracing, the changing mining algorithm (Wang et al., 2003) described in Section 2. Compared with CEDT method, it requires at least one dataset to detect changing regions. However, as shown further, the quality of Correspondence Tracing is comparable with the quality of CEDT.

The Correspondence Tracing algorithm outputs many changing regions ranked by the proposed quantitative change. To compare the performance with our method, we use a threshold on this measure to distinguish "significant enough" changing regions from the others. Such a threshold provides a trade-off between accuracy and recall: if chosen too high, the accuracy is high while the recall is low, if the threshold is too low, the accuracy is not high. For the experiments, the threshold is set so that the accuracy of Correspondence Tracing is comparable with the accuracy of CEDT.

If not mentioned otherwise, each dataset in the experiments below contains 5,000 samples, 5 dimensions and 5 classes.

The quality of the algorithms with respect to the number of classes is shown in Figs. 8 and 9.

Figure 9 shows that the recall is decreasing as the number of classes increases. It can be explained by the fact that the decision trees produced for the datasets with more classes have more leaf nodes. Since the number of samples in the dataset stays

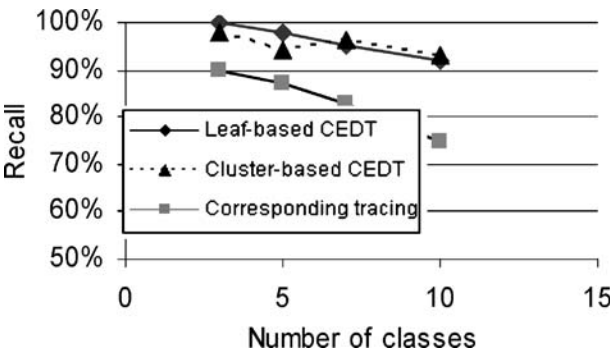


Fig. 9 Recall with respect to the number of classes

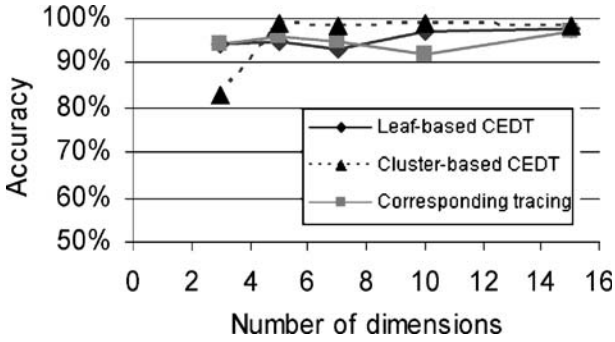


Fig. 10 Accuracy with respect to dimensionality

constant, there are fewer samples in each node. Because of that, there are more nodes where population is smaller than the minimum population threshold, therefore at those nodes the clustering is not performed. Thus, the information about some delta regions is lost, which reduces the recall of the algorithm.

Interestingly, the accuracy of CEDT algorithm increases when data sets contain more classes. As explained above, many nodes contain fewer samples than population threshold and thus are not summarized. On the other hand, remaining nodes have a high probability to contain real changes.

We also test the accuracy and the recall with respect to dimensionality. The quality of the algorithm is stable with respect to the number of dimensions. Since the clusters are used to summarize data, the algorithm is able to capture the spatial distribution of the data set even when the number of dimensions increases.

Figures 12 and 13 illustrate the quality of the algorithm with respect to the dataset size. We can see that both the leaf-based and the cluster-based approaches achieve good scalability. However, the accuracy of the cluster-based approach slightly decreases with increase in the data set size. With larger data set, it is more likely that the obtained changes reflect randomness in distribution rather than the changes embedded in data generator.

Figure 13 shows that recall is relatively low on a small dataset. The reason for that is that there are not many samples in each leaf node of the tree, therefore it is very likely that the number of samples would be less than minimum population threshold, and the information in the node would not be stored. This means that on the second phase of the algorithm such a region cannot be identified.

Together, Figs. 8, 9, 10, 11, 12 and 13 show that CEDT and Correspondence Tracing have comparable effectiveness. However, in comparison with CEDT,

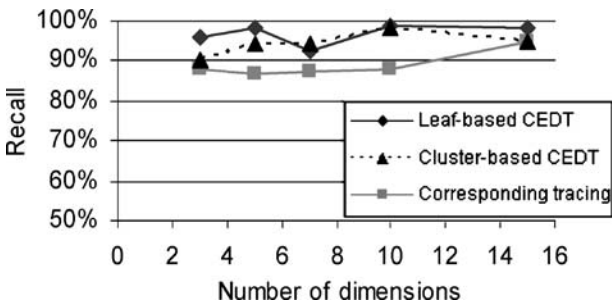


Fig. 11 Recall with respect to dimensionality

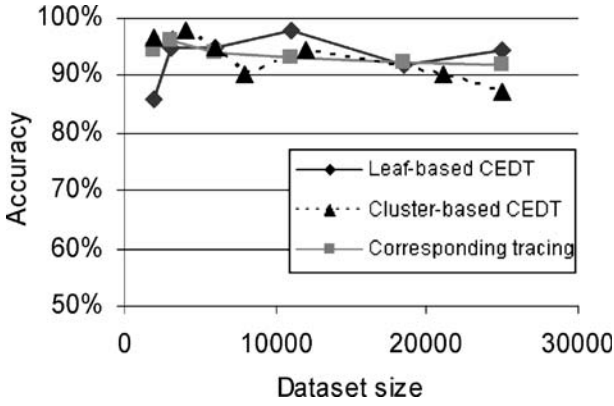


Fig. 12 Accuracy with respect to the dataset size

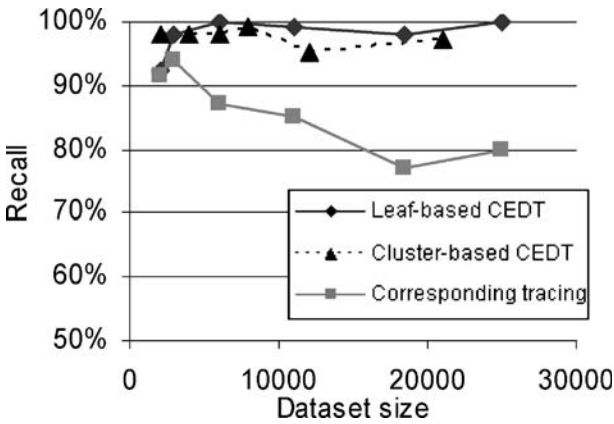


Fig. 13 Recall with respect to the dataset size

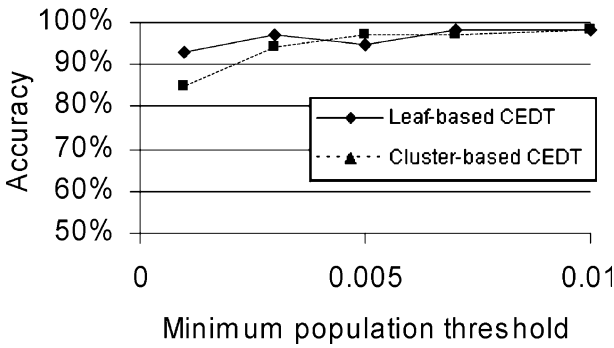


Fig. 14 Accuracy with respect to minimum population threshold

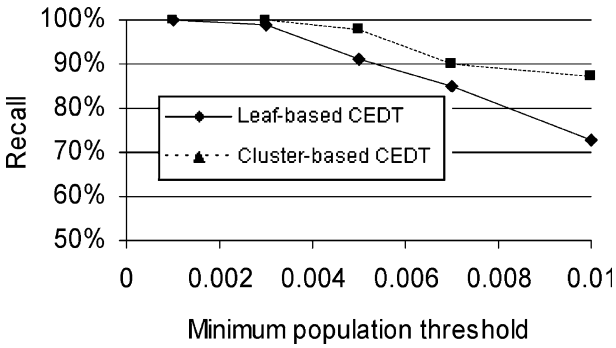


Fig. 15 Recall with respect to minimum population threshold

Correspondence Tracing requires at least one dataset to identify changes in data, therefore cannot be used on the dataset with access constraints.

Further we present the analysis on how the quality of the method is affected by the choice of the minimum population threshold. In our experiments, this parameter is proportional to the data set size (Figs. 14 and 15).

From Fig. 15 we can see that the recall is decreasing as the minimum population threshold is increasing. Indeed, in this case more nodes are considered "not significant enough" and therefore are not summarized by clusters. Later, on the change detection phase those regions cannot be identified.

The results above are presented for the mixture of changes described in Section 4.1.1.2. The experiments for each type of changes show similar results for both accuracy and recall. The only difference is that the experiments with adding and removing a cluster display stronger sensitivity to the minimum population threshold. Indeed, for these types of changes the exact definition of "empty" region is very important. Limited by space, we do not present details for each type.

In addition to measuring accuracy and recall to verify the effectiveness of the algorithm, we analyze some other important characteristics of the model, such as the number of clusters stored and the number of changing regions returned by the algorithm (Figs. 16, 17, 18 and 19). For the experiments below, we set the minimum population threshold to 0.005.

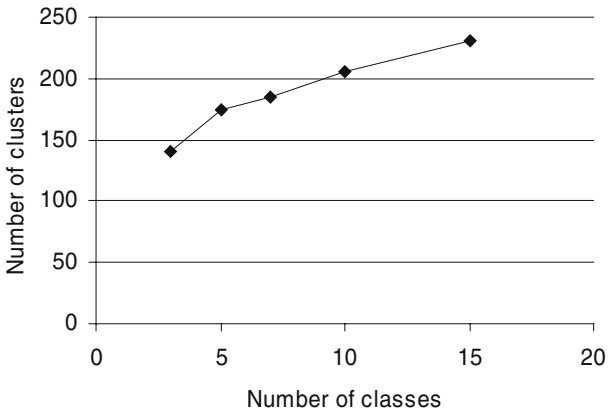


Fig. 16 Number of clusters with respect to number of classes

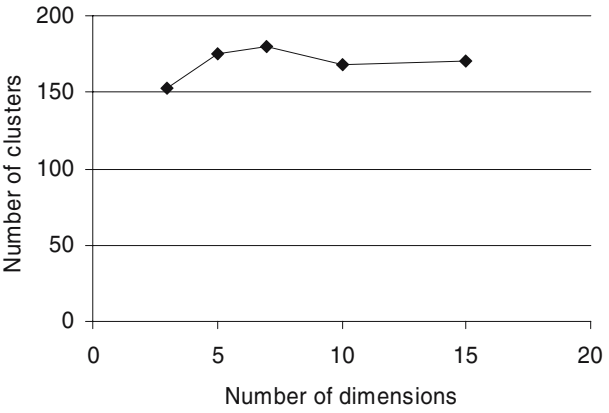


Fig. 17 Number of clusters with respect to dimensionality

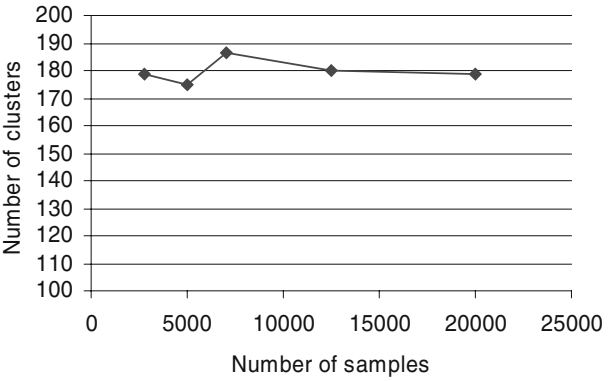


Fig. 18 Number of clusters with respect to the data set size

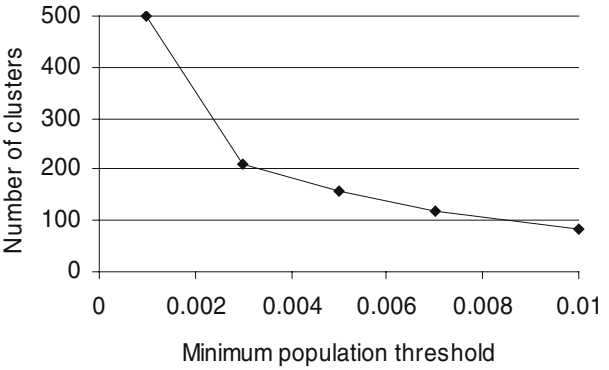


Fig. 19 Number of clusters with respect to the minimum population threshold

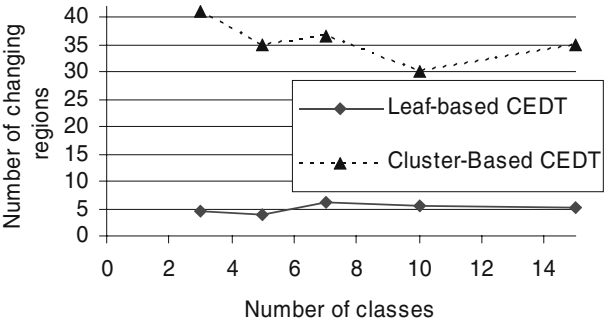


Fig. 20 Number of changing regions returned by the algorithm with respect to number of classes

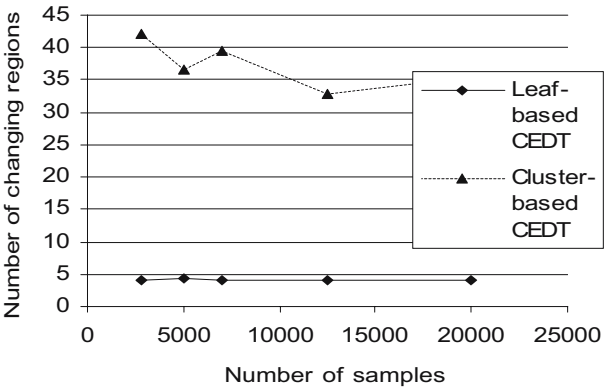


Fig. 21 Number of found changing regions with respect to dimensionality

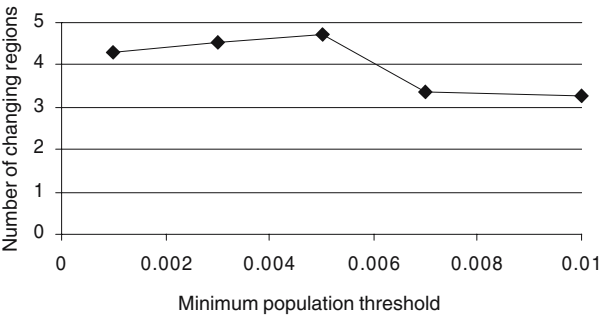


Fig. 22 Number of changing regions returned by the leaf-based approach with respect to minimum population threshold

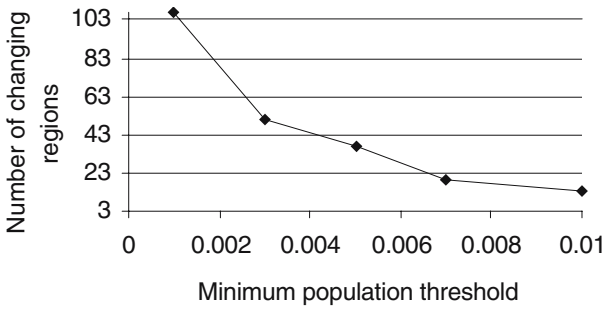


Fig. 23 Number of changing regions found by the cluster-based approach with respect to minimum population threshold

The number of clusters increases as the number of classes goes up. Clearly, datasets with more classes produce decision trees with more nodes, therefore more clusters are stored. The minimum population threshold that we use in the method is proportional to the number of samples. Therefore, the number of clusters stored in the model does not depend on the data set size. As expected, the number of clusters decreases when the minimum population threshold goes up.

Figures 20, 21, 22 and 23 show the number of changing regions detected by the algorithm. As mentioned before, the main difference between the cluster-based and the leaf-based approaches is in the number of regions they return. It is clearly illustrated by the figures below. The cluster-based approach detects 30–40 small changing regions, while the leaf-based approach detects 5–10 broad regions.

As the minimum population threshold increases, less nodes are clustered to summarize the data. Therefore, the number of regions obtained by the cluster-based approach decreases substantially. As can be seen from Fig. 15, the recall decreases as well.

4.2.2 Mortgage dataset

Here we report the changes found for the different subpopulations: "black" versus "white" (Table 2), "white" versus "American Indian" (Table 3) and "black" versus "American Indian" (Table 4).

From the Table 2 we can see that for a senior age group "black" people are still paying off the mortgage.

Tables 3 and 4 show that there are several cases when white and black people have a mortgage while native Indian people of the same age and income group own a house without any debts. This can be explained by the fact that native Indian people get some grants and loans from the government. In summary, the experiments show that the proposed method found the changes that are meaningful.

Table 2 Changes found from black versus white

House value	Age	Class label for "white"	Class label for "black"
12,500–37,500	59–63	No mortgage	Mortgage

Table 3 Changes found from white versus American Indian

House value	Age	Education	Income	Class label for "white"	Class label for "American Indian"	Change distance	Population proportion
>225,000	<51	Any	Any	Mortgage	No mortgage	0.73	0.075
5,000–32,500	<48	Any	Any	Mortgage	No mortgage	0.34	0.142
62,500–67,500	<51	<10	Any	Mortgage	No mortgage	0.57	0.048
37,500–42,500	13–48	Any	<9500	Mortgage	No mortgage	0.65	0.008

4.3 Efficiency of CEDT

The efficiency of the algorithm is estimated in terms of execution time. First, we measure the time required to build clusters. Clustering is used in the model construction phase of the algorithm as an enhancement for the decision tree algorithm. Second, we measure the time required for change detection.

Figures below show that most of the time required by the algorithm is during the model construction phase. Recall that the user has to construct the model only once from each dataset. Change detection phase, in contrast, should be repeated every time the user wants to compare a data set with another, and it takes very little time.

From Fig. 24 we can see that the time required for clustering decreases when the number of classes increases. As explained earlier, the amount of samples in each leaf node decreases, therefore clustering takes less time.

Time required for change detection stays constant with increase in the number of dimensions (Fig. 25) and increases linearly with increase in data set size (Fig. 26). The algorithm for change detection is quadratic in the number of clusters, therefore we may anticipate the quadratic increase of time with the data set size. However, due to the fact that the minimum population threshold is proportional to the data set size, the number of clusters stays constant, as can be seen from Fig. 18.

Clustering time increases linearly with the number of dimensions and with increase in the number of samples (Figs. 25 and 27). Recall that K-means clustering algorithm is used, and it is linear in the number of objects.

5 Conclusions

Detecting important changes and developing strategies for adapting to them is important in the changing world. In this paper, we focus on the problem of mining changing regions accurately and efficiently from access-constrained data sets. We develop a two-phase framework: every data set is summarized using a cluster-

Table 4 Changes found from white versus American Indian

House value	Age	Class label for "black"	Class label for "American Indian"
12,500–32,500	51–63	Mortgage	No mortgage

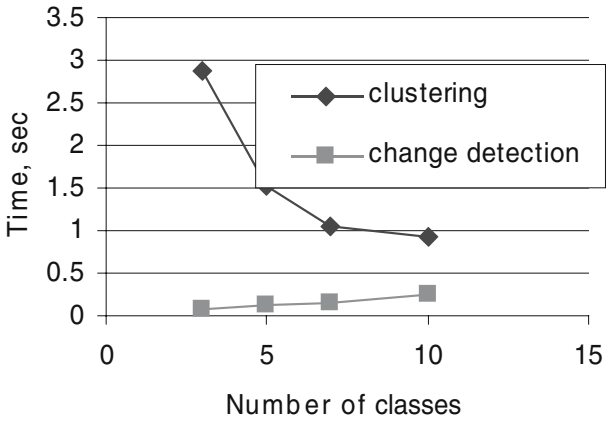


Fig. 24 Clustering and change mining time with respect to number of classes

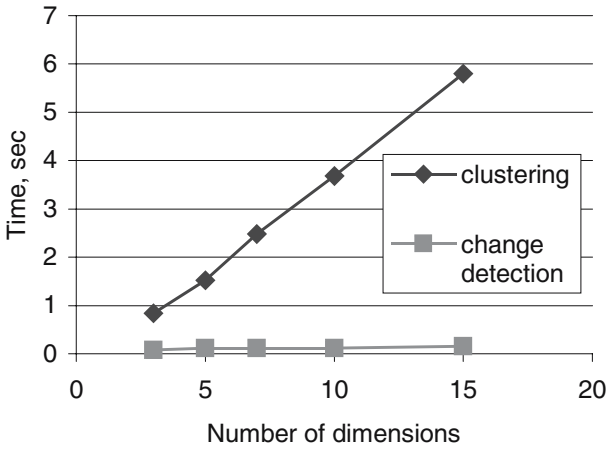


Fig. 25 Clustering and change mining time with respect to dimensionality

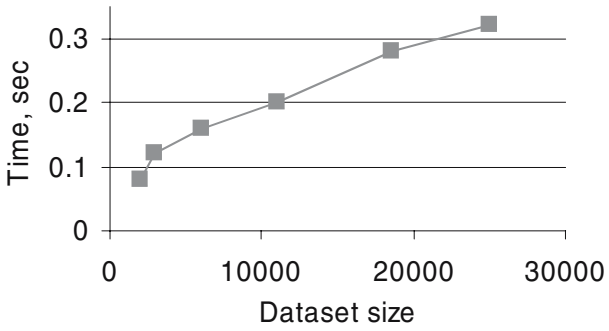


Fig. 26 Time for the change detection with respect to the data set size

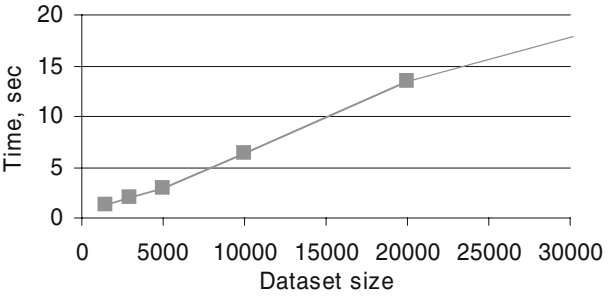


Fig. 27 Time required for clustering with respect to the data set size

embedded decision tree, and trees are used to detect changing regions between any two specified data sets. Our extensive experimental study on both real and synthetic datasets shows that the CEDT method is both accurate and efficient.

Acknowledgments The authors are grateful to the anonymous reviewers for their constructive and insightful comments and suggestions, which help to improve the quality of the paper.

References

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *proceedings of the ACM-SIGMOD international conference on management of data (SIGMOD'93)* (pp. 207–216). Washington, DC, May.

Agrawal, R., & Psaila, G. (1995). Active data mining. In *Proceedings of the 1st international conference on knowledge discovery in databases and data mining*.

Billard, L., & Diday E. (2003). From the statistics of data to the statistic of knowledge: Symbolic data analysis. *JASA. Journal of the American Statistical Association*, *98*, Nb. 462, June.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. New York: Wiley.

Diday, E., & Esposito, F. (2003). An introduction to symbolic data analysis and the sodas software. *IDA. International Journal on Intelligent Data Analysis, 7*(6).

Eiter T., & Mannila, H. (1997). Distance measures for point sets and their computation. *Acta Informatica, 34*(2), 103–133.

Ganti, V., Gehrke, J., & Ramakrishnan, R. (1999). A framework for measuring changes in data characteristics. In *proceedings of the Eighteenth ACM SIGAST–SIGMOD–SIGART Symposium on Principles of Database Systems, May 31–June 2, 1999, Philadelphia, Pennsylvania* (pp. 126–137). ACM.

Liu, B., Hsu, W., Han, H., & Xia, Y. (2000). Mining changes for real-life applications. In DaWaK 2000: Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery (pp. 337–346). London, UK: Springer.

Liu, B., Hsu, W., & Ma, Y. (2001). Discovering the set of fundamental rule changes. In Proceedings of the seventh ACM SIKDD international conference on Knowledge discovery and data mining (KDD'01) (pp. 335–340). San Francisco, California, USA: ACM.

Oliver, M. A., & Webster, R. (1990). Kriging: A method of interpolation for geographical information systems. *International Journal of Geographic Information Systems, 4*(3).

Papoulis, A. (1984). *Probability, random variables, and stochastic processes* (2nd ed.) (pp. 149–151). New York: McGraw-Hill.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rote, G. (1991). Computing the minimum Hausdorff distance between two point sets on a line under translation. *Information Processing Letters, 38*, 123–127.

Ruggles, S., Sobek, M., Alexander, T., Fitch, C. A., Goeken, R., Hall, P. K., et al. (2004). Integrated public use microdata series: Version 3.0 [http://www.ipums.org]. Minneapolis, MN: Minnesota Population Center.

Wang, K., Zhou, S., Fu, C. A., & Yu, X. J. (2003). Mining changes of classification by correspondence tracing. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM'2003)*. San Francisco, CA, USA, May.

Witten, I. H., & Frank, E. (2005) *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.