MINING CHANGING REGIONS FROM ACCESS-CONSTRAINED DATA SETS: A CLUSTER-EMBEDDED DECISION TREE APPROACH

By

Irina Pekerskaya
B.Sc., Nizhny Novgorod State University, Russia, 2003

THESIS
SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

In the School
of
Computing Science

© Irina Pekerskaya 2006
SIMON FRASER UNIVERSITY
Spring 2006

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without permission of the author.
Approval

Name: Irina Pekerskaya
Degree: Master of Science
Title of Thesis: Mining Changing Regions from Access-Constrained Data Sets: A Cluster-Embedded Decision Tree Approach

Examining Committee:
Chair: Dr. Fred Popowich

Dr. Jian Pei
Senior Supervisor

Dr. Ke Wang
Supervisor

Dr. Martin Ester
Examiner

Date Defended/Approved: January 16, 2006
DECLARATION OF PARTIAL COPYRIGHT LICENCE

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection, and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada
Abstract

Change detection is important in many applications. 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, the original data may not be available for change detection.

In this work, we tackle the problem by proposing a simple yet effective model-based approach. In the model construction phase, original data sets are summarized using the novel cluster-embedded decision trees as concise models. Once the models are built, the original data will not be accessed anymore. In the change detection phase, to compare any two data sets, 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.
To my family
Acknowledgements

I would like to express my deep gratitude to my senior supervisor Dr. Jian Pei. I thank him for sharing with me his expertise in data mining through numerous fruitful discussions and for teaching me valuable skills in conducting the research. His commitment to students and his patience make him an excellent mentor and adviser. I am grateful for his enthusiasm and encouragement that helped me overcome the difficulties on the way.

I would like to thank my supervisor Dr. Ke Wang for providing insightful comments and helpful suggestions that helped me to improve the quality of the thesis. My gratitude also goes to Dr. Martin Ester for serving on my examining committee. I thank him for his precious time reviewing my work. I am very thankful to Gabor Melli for inspiring me to work in the field of Data Mining, and for giving me advice that helped me throughout my thesis work. I also want to thank Javier Thaine for proofreading my thesis.

I thank all my friends for their enthusiasm and continuous support, thoughtful discussions and insightful ideas. A particular acknowledgement goes to Steven Bergner, Abhishek Gupta, Jangwon Kim, Michael Letourneau, Maria Li, Yaroslav Litus, Garima Mahadik, Flavia Moser, Rajashree Paul and Thomas Peter. I am very grateful to my dear friend Wendy Wang for all her support and understanding, for sharing joyful and sad moments and for the night walks.

Last but not least, I am very grateful to my family, especially my parents and my brother for their endless love, support, encouragement and wise advice. They helped me stay in good spirits and focus on my work.
# Table of Contents

Approval ............................................................................................................................ ii
Abstract ............................................................................................................................. iii
Dedication .......................................................................................................................... iv
Acknowledgements ......................................................................................................... v
Table of Contents .............................................................................................................. vi
List of Figures ....................................................................................................................... viii
List of Tables ......................................................................................................................... ix

## Chapter 1 Introduction ........................................................................................................
1.1 Motivation ....................................................................................................................... 1
1.2 Contributions ................................................................................................................... 2
1.3 Organization of the Thesis ............................................................................................... 2

## Chapter 2 Background and Related Work ........................................................................
2.1 Background ....................................................................................................................... 3
  2.1.1 Classification ............................................................................................................ 3
  2.1.2 Decision Tree Classifier ......................................................................................... 4
  2.1.3 Clustering ............................................................................................................... 6
  2.1.4 K-means Clustering Algorithm .............................................................................. 7
2.2 Related Work ................................................................................................................... 8

## Chapter 3 Cluster-Embedded Decision Trees .................................................................
3.1 Problem Description ....................................................................................................... 13
3.2 Model construction phase ............................................................................................... 14
  3.2.1 Choosing a Model for Changing Region Detection .................................................. 14
  3.2.2 Cluster-Embedded Decision Trees (CEDT) ............................................................. 17
    3.2.2.1 Decision Tree Construction ............................................................................. 18
    3.2.2.2 Clustering ....................................................................................................... 18
    3.2.2.3 Cluster Representation ................................................................................... 20
3.3 Mining Changing Regions .............................................................................................. 22
  3.3.1 Leaf-based Changing Regions .................................................................................. 22
    3.3.1.1 Overlay of Two Trees .................................................................................... 22
    3.3.1.2 Real and False Changing Regions ................................................................. 25
    3.3.1.3 Using CEDT to Estimate Region Population .................................................. 29
    3.3.1.4 Presenting Changing Regions ....................................................................... 32
  3.3.2 Cluster-Based Changing Regions .............................................................................. 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2.1 Summarization of Significant Changing Regions</td>
<td>35</td>
</tr>
<tr>
<td>3.4 Change Mining Language (CML)</td>
<td>36</td>
</tr>
<tr>
<td>Chapter 4 Experimental Evaluation</td>
<td>38</td>
</tr>
<tr>
<td>4.1 Datasets and Experimental Settings</td>
<td>38</td>
</tr>
<tr>
<td>4.1.1 Synthetic Data</td>
<td>38</td>
</tr>
<tr>
<td>4.1.1.1 Data Generator</td>
<td>39</td>
</tr>
<tr>
<td>4.1.1.2 Types of Changes</td>
<td>39</td>
</tr>
<tr>
<td>4.1.2 IPUMS Census Data</td>
<td>40</td>
</tr>
<tr>
<td>4.2 Effectiveness of CEDTs</td>
<td>40</td>
</tr>
<tr>
<td>4.2.1 Synthetic Data</td>
<td>40</td>
</tr>
<tr>
<td>4.2.2 Mortgage Dataset</td>
<td>53</td>
</tr>
<tr>
<td>4.2.2.1 Measuring the Quality of the Algorithm on Real Data Sets</td>
<td>54</td>
</tr>
<tr>
<td>4.3 Efficiency of CEDT</td>
<td>55</td>
</tr>
<tr>
<td>Chapter 5 Conclusions and Future Research</td>
<td>59</td>
</tr>
<tr>
<td>5.1 Summary of the Thesis</td>
<td>59</td>
</tr>
<tr>
<td>5.2 Future Research Directions</td>
<td>60</td>
</tr>
<tr>
<td>Bibliography</td>
<td>62</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A decision tree for the concept PlayTennis</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>The framework of model – based change detection</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Two data sets, the decision trees and the changes</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>The CEDT construction algorithm</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>A decision tree for the Iris dataset</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>The procedure for computing overlay</td>
<td>24</td>
</tr>
<tr>
<td>3.6</td>
<td>The decision tree for the dataset D₁</td>
<td>25</td>
</tr>
<tr>
<td>3.7</td>
<td>Partition produced by the decision tree on the dataset D₁</td>
<td>26</td>
</tr>
<tr>
<td>3.8</td>
<td>Embedded change</td>
<td>26</td>
</tr>
<tr>
<td>3.9</td>
<td>The decision tree for the dataset D₂</td>
<td>27</td>
</tr>
<tr>
<td>3.10</td>
<td>Partition produced by the decision tree for the dataset D₂</td>
<td>27</td>
</tr>
<tr>
<td>3.11</td>
<td>Overlay of partitions</td>
<td>28</td>
</tr>
<tr>
<td>3.12</td>
<td>Clusters summarizing the objects in the region</td>
<td>30</td>
</tr>
<tr>
<td>3.13</td>
<td>Using CEDT to estimate the population of the region</td>
<td>31</td>
</tr>
<tr>
<td>3.14</td>
<td>The leaf-based change detection algorithm</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Accuracy with respect to the number of classes</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Recall with respect to the number of classes</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Accuracy with respect to the number of dimensions</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Recall with respect to the number of dimensions</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Accuracy with respect to the dataset size</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>Recall with respect to the dataset size</td>
<td>44</td>
</tr>
<tr>
<td>4.7</td>
<td>Accuracy with respect to minimum population threshold</td>
<td>45</td>
</tr>
<tr>
<td>4.8</td>
<td>Recall with respect to minimum population threshold</td>
<td>46</td>
</tr>
<tr>
<td>4.9</td>
<td>Accuracy and recall of the leaf-based approach with respect to cluster minimum population threshold</td>
<td>46</td>
</tr>
<tr>
<td>4.10</td>
<td>Accuracy and recall for the cluster-based approach with respect to cluster minimum population threshold</td>
<td>47</td>
</tr>
<tr>
<td>4.11</td>
<td>Number of clusters with respect to the number of classes</td>
<td>48</td>
</tr>
<tr>
<td>4.12</td>
<td>Number of clusters with respect to the number of dimensions</td>
<td>48</td>
</tr>
<tr>
<td>4.13</td>
<td>Number of clusters with respect to the data set size</td>
<td>49</td>
</tr>
<tr>
<td>4.14</td>
<td>Number of clusters with respect to the minimum population threshold</td>
<td>49</td>
</tr>
<tr>
<td>4.15</td>
<td>Number of changing regions returned by the algorithm with respect to the number of classes</td>
<td>51</td>
</tr>
</tbody>
</table>
Figure 4.16  Number of found changing regions with respect to the number of dimensions ......................................................... 51
Figure 4.17  Number of found changing regions with respect to the data set size .......... 51
Figure 4.18  Number of changing regions found by the cluster-based approach with respect to the minimum population threshold .............................................. 52
Figure 4.19  Number of changing regions returned by the leaf-based approach with respect to the minimum population threshold .............................................. 52
Figure 4.20  Clustering and change mining time with respect to the number of classes... 56
Figure 4.21  Clustering and change mining time with respect to the number of dimensions ................................................................................................................. 56
Figure 4.22  Time for change detection with respect to the data set size ...................... 57
Figure 4.23  Time required for clustering with respect to the data set size ................. 57

List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Parameters used by the data generator</td>
<td>39</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Changes found from black versus white</td>
<td>53</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Changes found from white versus American Indian</td>
<td>53</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Changes found from white versus American Indian</td>
<td>53</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

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

For example, mining changing regions with respect to plant species distribution is essential for understanding environmental changes. Moreover, in customer relation management, it is important to monitor and analyze changes of behaviour patterns in various customer groups.

1.1 Motivation

Change detection has been studied extensively for a long time in both statistics and machine learning. A survey of the related work will be presented in Chapter 2. Most of the previous studies are focused on measuring the changes between 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 detection. For example, due to the privacy and information protection concerns, owners of some census or customer data 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 previous change detection methods requiring access to original data sets may not be applied. Those applications naturally raise a challenge: With access constraints, can we still mine changing regions effectively?
1.2 Contributions

In this thesis, we study the problem of mining changing regions from the access-constrained data sets and 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). The model carries concise yet sufficient information to be used 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.

1.3 Organization of the Thesis

The remainder of the thesis is structured as follows:

In Chapter 2, we provide background information for the rest of the thesis and present an overview of related work.

In Chapter 3, a novel two-phase change detection method is developed.

The experimental results on both synthetic and real data sets are shown in Chapter 4.

Chapter 5 concludes the thesis and discusses directions for future research.
Chapter 2
Background and Related Work

In this chapter, we first provide an overview of background information, then we review previous studies related to our work.

2.1 Background

This chapter provides background information for the remainder of the thesis. Section 2.1.1 reviews main concepts and terminologies of classification. The next section gives a brief overview of decision tree learning methods. Section 2.1.3 provides an introduction to clustering algorithms. Description of the K-means algorithm is given in Section 2.1.4.

2.1.1 Classification

Classifying examples into a discrete set of possible categories is referred to as classification. Classification is a two-step process. In the first step, a model, or hypothesis, is learned to describe a predetermined set of data classes. In the context of classification, the data used to build a model forms the training data set. The model is constructed by analyzing data described by attributes. Each sample is assumed to belong to a predefined set of data classes, as determined by its class label.

In the second step, the model generated in the preceding step is tested on a test data set. The test set is different from the training set, and every element is also preclassified in advance. As a quality measure the predictive accuracy of the classifier is estimated. It is determined by comparing true class labels in the test set with those assigned by the model. For each sample, the predicted class label is compared with the known class label.

If the data set is not large enough to be divided into training and test sets of significant sizes, a cross-validation can be used. In a k-fold cross validation, the available data is partitioned into k separate sets of approximately equal size. The cross-validation procedure involves k iterations in which the learning method uses (k-1) subsets as training data, and is tested on the set that is left. Each subset is used for testing only once.
The cross-validation accuracy of the given algorithm is an average of the accuracy measurement from the individual folds.

If the accuracy of the model is satisfactory, the model can be used to classify samples with unknown class labels.

Different techniques from statistics, machine learning and data mining are used for classification. They include Bayesian methods [DH73, Hec96, Jen96], decision trees[Qui93], neural networks[RHW86] and support vector machines[Vap95]. A good survey of these and other methods is given in [HK00]. In this thesis we use a decision tree classifier for the task of change mining.

2.1.2 Decision Tree Classifier

A decision tree is a tree structure where each node specifies a test of some attribute, each branch descending from the node represents an outcome of the test, and leaf nodes represent classes or class distributions.

![Figure 2.1 A decision tree for the concept PlayTennis](image)

Figure 2.1 illustrates a typical decision tree. It shows whether a particular day is suitable for playing tennis.

In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node that provides a class prediction for that sample.
Decision tree learning involves constructing a tree by recursively partitioning the training examples. To determine which attribute should be tested at the root of the tree, each attribute is evaluated using a statistical test to determine how well it alone classifies the training instances. The best attribute is selected and used as the test at the root node of the tree. A branch is created for each known value of the test attribute, and the samples are partitioned accordingly. The entire process is then repeated recursively to form a decision tree for the samples at each partition. This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

Many algorithms have been developed for learning decision trees. In this thesis, we choose C4.5 as a representative algorithm [Qui93].

One of the key aspects of decision tree algorithms is selecting the attribute that will best separate the samples into individual classes. C4.5 uses an evaluation measure called information gain. The attribute with the highest information gain is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the number of tests needed to classify an object.

In order to define information gain precisely, a measure that characterizes the impurity of an arbitrary collection of samples, called entropy, should be defined. If there are \( c \) different classes, the entropy for the set \( S \) is defined as

\[
\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \log_2 p_i,
\]

where \( p_i \) is the proportion of \( S \) belonging to class \( C_i \). The smaller the entropy value, the greater the purity of the subset partitions.

The information gain is the reduction in entropy caused by partitioning the examples according to this attribute. Let attribute \( A \) have \( k \) distinct values, \( \{A_1, A_2, \ldots, A_k\} \). Attribute \( A \) can be used to partition \( S \) into \( k \) subsets, \( S_1, S_2, \ldots, S_k \), where \( S_j \) contains those samples in \( S \) that have value \( A_j \) on \( A \). Then information gain is defined as:
\[ \text{Inf } \_ \text{Gain}(S, A) = \text{Entropy}(S) - \sum_k \frac{|S_k|}{|S|} \text{Entropy}(S_k) \]

The algorithm computes the information gain for each attribute. The attribute with the highest information gain is chosen as the test attribute for the given set \( S \).

Another key aspect of a decision tree algorithm is to determine when to stop growing a tree. C4.5 uses several stopping criteria to decide when to leave a node as a leaf. It will not split at the node in the following cases:

- If all the samples in a node belong to the same class;
- If there are no remaining attributes on which the samples may be further partitioned, or
- If there are no samples in a node. In this case the node is labeled with the most frequent class of the parent.

After a decision tree is built, many of the branches can reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. C4.5 uses an approach called postpruning that removes branches from a tree after it is built. C4.5’s pruning method considers replacing each internal node by either a leaf or one of the branches of the node. In order to decide if a change should be made, C4.5 computes a confidence interval to estimate the predicted error rate. A change is made to a subtree if the resulting error rate for the modified subtree is within a specified confidence interval of the error rate of the unmodified subtree.

2.1.3 Clustering

Clustering is the process of grouping the data into clusters so that objects within a cluster have high similarity, but are very dissimilar to objects in other clusters. Similarity is commonly defined in terms of how “close” the objects are in space, based on a distance function.
Clustering can be applied to data with attributes of different types, such as numeric, binary, nominal, and others. In this thesis, we only consider data sets in a numeric domain, therefore we provide more details here on clustering variables of numeric type.

If the dimensions have very different scales, it can strongly affect the clustering analysis. For example, changing measurement units from meters to inches can lead to a very different cluster structure. Expressing a variable in smaller units will lead to a larger range for that variable. To deal with this problem, the data should be normalized. For the variable with \( n \) values \( x_1, \ldots, x_n \) and a mean \( m \), the mean absolute deviation is computed first:

\[
S = \frac{1}{n} \left( |x_1 - m| + |x_2 - m| + \ldots + |x_n - m| \right)
\]

Then, the standardized measurement, or z-score, is calculated:

\[
z_i = \frac{x_i - m}{S}
\]

The choice whether to use standardization or not is normally left to the user.

The dissimilarity between the objects is typically computed based on the distance between each pair of objects. The most popular is Euclidean distance, which is defined as

\[
d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ip} - x_{jp})^2}
\]

where \( i \) and \( j \) are two \( p \)-dimensional data objects.

There is a large number of clustering algorithms [KR90, JMF99]. In this thesis, we choose the K-means clustering algorithm [Mac67] that is described in the next section.

### 2.1.4 K-means Clustering Algorithm

The K-means algorithm is one of the most commonly used clustering algorithms. It takes an input parameter, \( k \), and partitions a set of objects into \( k \) clusters so that the similarity
inside the cluster is high but similarity among different clusters is low. Cluster similarity is measured by the average distance between the objects and the cluster mean.

The algorithm works as follows. First, it randomly selects k objects, each of which initially represents a cluster mean. Each of the remaining objects is assigned to a cluster to which it is the most similar, based on the distance to the cluster mean. It then computes the new mean of the cluster. The process is repeated until the criterion function converges.

Usually, the squared-error criterion is used, defined as:

\[ E = \sum_{i=1}^{k} \sum_{p \in C_i} (p - m_i)^2, \]

where E is the sum of squared-error for all objects in the database, p is the point in the space representing a given object, and \( m_i \) is the mean of cluster \( C_i \). This criterion tries to make the resulting clusters as compact and separate as possible.

### 2.2 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 [Rot91], 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. Aside from Hausdorff distance, many approaches tried to define similarity between two point sets for different applications. These include the surjection measure, the fair surjection measure, the minimum link distance, etc. [EM97]

In the field of information theory, relative entropy, or the Kullback Leibler (K–L) divergence [CT91], has been suggested as the 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 $D(p \| q) = \sum_i p_i \cdot \log \frac{p_i}{q_i}$.

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, then 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 [OW90] developed in the field of geostatistics uses a semivariogram to assign weights. The 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 thesis, we are concerned with identifying regions of changes instead of measuring the global difference.

In the context of association rule mining [AIS93], [AP95] 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 [LHM01], 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 can be missing.

None of these works 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 [GGR99], 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. These computations require accessing original data sets.

While the method determines whether the changes between two data sets exist and measures how significant they 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 thesis.

In [LHHX00], 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 two arbitrary models. For example, if important changes occur at the top levels of the decision tree, the method cannot be used. It is also not applicable in the situations where the original data sets are not available for direct analysis.

In [WZFY03], 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.
Chapter 3
Cluster-Embedded Decision Trees

Most of the previous change detection methods described in Chapter 2 require accesses to the original data sets. In this chapter, we propose a framework that is able to mine changes from data with access constraints. The framework, as shown in Figure 3.1, 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 chapter. 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, the methods detect changing regions between any two data sets specified by the user. At this stage, data sets might not be available and thus cannot be accessed.

The details of the method are presented in this chapter. 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. In the Section 3.4 we introduce CML, a script language used to specify change mining tasks.

Figure 3.1 The framework of model-based change detection
3.1 Problem Description

Consider two data sets D1 and D2, where all data objects in the data sets are in a numeric multidimensional space D (i.e., each dimension is in a numeric domain).

Moreover, each object belongs to a 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.

**Definition 3.1 (Region)** 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 \( R = (l_1, u_1), \ldots (l_k, u_k) \), where \( -\infty \leq l_i \leq u_i \leq \infty \) for \( 1 \leq i \leq k \) and k is the dimensionality. \( R.A_i = (l_i, u_i) \) is the scope of region R on attribute A_i.

**Definition 3.2 (Region Population)** Let R be a region. The population of R with respect to data set D is the number of objects in D that fall into R, denoted by \( \text{pop}_D(R) \). 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 \( \text{pop}^C_D(R) \). Clearly, we have \( \text{pop}_D(R) = \sum_{C} \text{pop}^C_D(R) \).

**Definition 3.3 (Dominating class)** 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.

Intuitively, a changing region from a data set to another is dominated by objects from different classes in the two data sets. To avoid triviality, such changing regions should have enough population.

**Definition 3.4 (Changing region)** Given two data sets D_1 and D_2 and a minimum population threshold \( \text{min\_population} \), 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) \( \text{pop}_{D_1}(R) \geq \text{min\_population} \)
3) \( \text{pop}_{D_2}(R) \geq \text{min\_population} \)

13
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 the 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 Figures 2(a) and 2(b), respectively. After a tree is built, the data is partitioned into disjoint regions in the attribute space. The corresponding partitions are shown in Figures 2(c) and 2(d), respectively.
Figure 3.2  Two data sets, the decision trees and the changes
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 by Figure 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 northwestern rectangle. There is no change between the two data sets in this area. It just happens that 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 southwestern rectangle is also reported as a changing region from class $\text{Class}_1$ to class $\text{Class}_3$ in Figure 2(e). However, within the rectangle, only a new group of objects in $\text{Class}_3$ appear. There are in fact no objects in $\text{Class}_1$ 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 Figure 2(f).

Why might directly comparing decision trees not be accurate or even correct? First, in decision tree construction, the space is divided into hyper-rectangles recursively until the resulting 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 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 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 size. 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.

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 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 falls into 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.
Even though each node will be mostly populated by the objects of one class, there can also be objects from other classes. Clusters are used to group objects from the same class. Note that clustering in this setting is not used to separate classes from each other, but rather to provide additional information on the spatial distribution of objects within each class.

Generally, we can use any clustering method to construct clusters at leaf nodes for a cluster-embedded decision tree. 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 \( cl_{\text{min\_pop}} \) be a user-specified cluster minimum population threshold. It should be selected so that any clusters with the population less than \( cl_{\text{min\_pop}} \) are considered insignificant and can be ignored from data analysis.

Consider a leaf node with \( n \) data objects, where the objects are in classes \( C_1, \ldots, C_k \) and the number of objects in the classes are \( n_1, \ldots, n_k \), respectively. For class \( C_i \), we form up to \( \left\lfloor \frac{n_i}{cl_{\text{min\_pop}}} \right\rfloor \) clusters to capture the distribution of objects in the class. Clearly, for a
node with \( n \) objects, at most \( \left\lfloor \frac{n}{cl\_min\_pop} \right\rfloor \) 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 cluster minimum population threshold \( cl\_min\_pop \), there are at most \( \left\lfloor \frac{m}{cl\_min\_pop} \right\rfloor \) 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, they would not be 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 be 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 \((D_1, \ldots, D_k)\), let \( S \) be a set of \( n \) objects, whose mean is \((c_1, \ldots, c_k)\) 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 \( ([c_1 - t \cdot d_1, c_1 + t \cdot d_1], \ldots, [c_k - t \cdot d_k, c_k + t \cdot d_k]) \)
Proof: According to Chebyshev’s inequality [Pap84], 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 Figure 3.4.

\begin{verbatim}
Input: A dataset \( D \), min_p threshold 
Output: CEDT \( T \)
Method:
build a decision tree, no nodes with population less than \( \text{min}_p \) should be split further, where \( \text{min}_p \) is the minimum population threshold;
for each leaf node in the tree that has at least \( \text{min}_p \) data objects, do
  for each class in the leaf node that has at least \( \text{min}_p \) objects, do
    form \( \frac{n_i}{\text{min}_p} \) clusters, where \( n_i \) is the number of objects of the class at the leaf
\end{verbatim}

Figure 3.3 The CEDT construction algorithm
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 number 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.

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 3.5 (Path)** A path from the root to a leaf node is a pair $(\text{region}, \text{Class label})$, 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:

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

22
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 \((-\infty, +\infty)\).

![Decision Tree](image)

**Figure 3.4** A decision tree for the Iris dataset

Example 3.2 (Path):

For the decision tree in figure 3.5 we use the following order of attributes: *petalwidth*, *petallength*. That way, the path from the root to Leaf 3 can be described as:

\((0.6, 1.5], (4.9, \infty) \Rightarrow \text{Iris\_virginica}\)

Notice that the order of the dimensions may not be the same as the order in the decision tree.

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

**Definition 3.6 (Intersection)** To form an *intersection* of two paths \( P_1 = \{[L_i, U_i] : i = 1, \ldots, k\} \) and \( P_2 = \{[L_i, U_i] : i = 1, \ldots, k\} \), from decision trees \( T_1 \) and \( T_2 \), respectively, we compute an intersection for each attribute:
\[ P_1 \cap P_2 = \left[ \left[ \max \{ L_1, L_2 \}, \min \{ U_1, U_2 \} \right], ... , \left[ \max \{ L_k, L_2 \}, \min \{ U_k, U_2 \} \right]\right] \]

An intersection is empty if there exists an attribute \( A_i \) such that \( \max \{ L_i, L_i \} > \min \{ U_i, U_i \} \)

**Example 3.3 (Intersection)**

Path1 = \{(0.6, 1.51, (4.9, \infty)]
Path2 = \{(-\infty, 3.71, (3.5, 6.2)]
\( I = \text{Path1} \cap \text{Path2} = \{(0.6, 1.51, (4.9, 6.2)]\)

**Definition 3.7 (Exclusive paths)** We call two paths \( P_1 \) and \( P_2 \) exclusive if their intersection \( P_1 \cap P_2 \) is empty.

**Definition 3.8 (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 3.9 (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:

\( R \Rightarrow \text{OldClass}, \text{NewClass} \), where \( \text{OldClass} \) and \( \text{NewClass} \) are class labels in \( D_1 \) and \( D_2 \), respectively, and \( R \) is a delta region.

To form such an overlay the following procedure is used:

<table>
<thead>
<tr>
<th>For each path ( P_1 ) in first dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each path ( P_j ) in second dataset</td>
</tr>
<tr>
<td>If ( \text{Class} (P_j) \neq \text{Class} (P_1) )</td>
</tr>
</tbody>
</table>

**Figure 3.5. The procedure for computing overlay**
Using the procedure in Figure 3.6, we can identify all possible changing regions by finding all the intersections between paths in the old and the new decision trees.

**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 $O(n \times m)$.

### 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.

**Example 3.4. (Decision tree overlay)** Let us consider another two-dimensional example (Figures. 3.6-3.12). The datasets have two attributes and 3 classes. We build the decision trees for the datasets $D_1$ and $D_2$ (Figure 3.7 and Figure 3.10) and show an introduced change in Figure 3.9. The partitions produced by the trees in the two-dimensional space are shown in Figures 3.8 and 3.11. A comparison of partitions is shown in Figure 3.12.

![Decision Tree Diagram](image)

**Figure 3.6** The decision tree for the dataset $D_1$
Figure 3.7 Partition produced by the decision tree on the dataset $D_1$

Figure 3.8 Embedded change

Change class label from class 3 to class 1
Figure 3.9 The decision tree for the dataset $D_2$

Figure 3.10 Partition produced by the decision tree for the dataset $D_2$
A closer look at the results of computing the decision trees overlay indicates that the changing regions are reported due to two reasons:

The corresponding samples in the data sets have changed their class labels.

The paths in the decision trees are different where there are no actual changes in data.

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 (for example, Region 3 in the Figure 3.12). 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. Region 2 in Figure 3.12 is one example. 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 $D_2$. 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 $R = ([x_1, y_1], \ldots, [x_k, y_k])$ be a region. There are three possibilities:

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 3.5 (Estimating Region Population). Let us revisit the data sets described in the example 3.4. The region $(524, 689], (0, 584]$ (Region $R_1$ in Figure 12(a)) is classified
as class 3 by decision tree on $D_1$, while the region $(608, 781), (0, 453)$ (Region $R_2$ in Figure 12 (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 Figure 12. 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 $R_2$.

Figure 3.12 Clusters summarizing the objects in the region
Figure 3.13 Using CEDT to estimate the population of the region

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 $1 \leq i \leq k$.

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

$$\text{overlap}(L, R) = \prod_{i \in C}(\min\{y_i', y_i\} - \max\{x_i, x_i'\})$$

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

$$\frac{\text{overlap}(L, R) \cdot N_L}{\prod_{i \in C}(y_i' - x_i')}$$

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., $\text{pop}_C^L(R)$, and that with respect to classes other than $C$. 

31
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 D1 as in D2. 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 \( \text{min\_population} \) 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 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 a certain region. More precisely, we use the "change distance" to measure how strongly a particular region changes. Given \( P_{D_i}^C(R) \) - the probability that a point from region R belongs to class \( C_i \) in the data set \( D_i \), the distance is computed as follows:

\[
\text{ChangeDist}(R) = \sqrt{\sum_i \left( P_{D_i}^C(R) - P_{D_i}^C(R) \right)^2} = \sqrt{\sum_i \left( \frac{\text{pop}_{D_i}^C(R)}{\text{pop}_{D_i}(R)} - \frac{\text{pop}_{D_i}^C(R)}{\text{pop}_{D_i}(R)} \right)^2},
\]

Where \( \text{pop}_{D_i}^C(R) \) and \( \text{pop}_{D_i}^C(R) \) are populations of class \( C_i \) in the region R in the data sets \( D_1 \) and \( D_2 \), respectively, and \( \text{pop}_{D_i}(R) \), \( \text{pop}_{D_i}(R) \) - populations of the region in the data sets \( D_1 \) and \( D_2 \), respectively.
**Example 3.5. (Computing change distance)** Consider a dataset with 3 classes. The region R has the following populations:

\[ \text{pop}^D_1(R) = 30 \quad \text{pop}^D_2(R) = 90 \]
\[ \text{pop}^D_1(R) = 60 \quad \text{pop}^D_2(R) = 45 \]
\[ \text{pop}^D_1(R) = 10 \quad \text{pop}^D_2(R) = 15 \]

Then, the change distance is calculated as follows

\[
\text{ChangeDist}(R) = \sqrt{\left(\frac{30 - 90}{100 - 150}\right)^2 + \left(\frac{60 - 45}{100 - 150}\right)^2 + \left(\frac{10 - 15}{100 - 150}\right)^2} = 0.42
\]

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 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 region's population and significance of changes. Intuitively, the more interesting regions have a larger population and a larger change distance.

More precisely, the function to rank the regions is computed as follows:

\[
\text{Rank}(R) = \left( \frac{\text{pop}^D_1(R)}{\text{pop}^D_1} + \frac{\text{pop}^D_2(R)}{\text{pop}^D_2} \right) \cdot \text{ChangeDist}(R)
\]

Here we normalise the population of each region with respect to the population in its respective data set. Since data sets \(D_1\) and \(D_2\) can have different numbers of samples, this approach prevents regions belonging to larger data sets from dominating.

The overall algorithm for leaf-based change detection is presented below:
Leaf-based change detection:

**Input:** CEDTs $T_1$ and $T_2$

**Output:** a set of rules $\text{Rules}$, ranked from the most meaningful change

**Method:**

compute the overlay of two decision trees

for each delta region $R$

Determine $C_1$ and $C_2$ - the dominant classes for $D_1$ and $D_2$;

if $C_1 \neq C_2$ then

$\text{Old\_Population} \leftarrow$ estimated population in the old data set;

$\text{New\_Population} \leftarrow$ estimated population in the new data set;

if $\text{Old\_Population}$ and $\text{New\_Population}$ are larger then $\text{min\_p}$

then

Add rule to $\text{Rules}$: $\langle \text{Region}, C_1, C_2 \rangle$

for each element in the $\text{Rules}$

Compute rank function

Order $\text{Rules}$ according to the rank

---

**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 the 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 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, the 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, 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.

Then, those changing regions are used as the training examples to build a decision tree, which is presented as a result of cluster-based change mining.
To take regions as training data instead of individual samples, a decision tree algorithm should be slightly modified. All the steps remain the same as they are described in Chapter 2, except for the step of selecting the attribute with the highest information gain. Recall that information gain for the attribute $A_i$ that partitions the set of examples $S$ into $k$ subsets $\{S_1, \ldots, S_k\}$ is computed as follows:

$$\text{Inf}_G\text{ain}(S, A_i) = \text{Entropy}(S) - \sum_k \frac{|S_k|}{|S|}\text{Entropy}(S_k)$$

To compute it, we need to compute entropy for the set $S$ and subsets $\{S_1, \ldots, S_k\}$:

$$\text{Entropy}(S_j) = \sum_{i=1}^c -p_{ij} \log_2 p_{ij}$$

where $p_{ij}$ is the proportion of $S_j$ belonging to class $C_i$.

Given a set of regions instead of a set of samples, how can we calculate the proportion $p_{ij}$? Recall that all the samples in the regions belong to the same class and the value of the particular attribute is given by an interval instead of a single point.

Consider a region $R$ with $n$ samples and range $[l, u]$ on the dimension $A_i$. If we use $p$ as the splitting point on this dimension that divides $S$ into subsets $S_1$ and $S_2$, the following cases can occur:

- $p > u$. In this case, all $n$ samples belong to subset $S_1$
- $p < l$. All $n$ samples belong to subset $S_2$
- $l \leq p \leq u$. We assume that samples in the region are uniformly distributed. Thus, $\frac{p - l}{u - l} \cdot n$ samples belong to subset $S_1$, and $\frac{u - p}{u - l} \cdot n$ samples belong to subset $S_2$.

### 3.4 Change Mining Language (CML)

In practice, many different formulations of the change mining task exist. Instead of looking for all possible changes from any class to any other, the user may be interested in
a particular combination of classes, e.g. changes from class $C_1$ to class $C_3$. To facilitate the interaction between the user and the change mining system, we propose the scripting language called CML (change mining language).

The BUILD MODEL statement is used to build CEDT on the chosen dataset:

```
BUILD MODEL FOR $D_1$,
```

Where $D_1$ is a file containing a dataset

The user can mine changes from two saved CEDT models $T_1$ and $T_2$ using MINE CHANGES statement. The syntax is presented below:

```
MINE CHANGES
FROM $T_1$ TO $T_2$
USING Approach
[WITH CLASS CHANGE $C_i$ TO $C_j$]
[OLD POPULATION $\geq P_{old}$]
[NEW POPULATION $\geq P_{new}$]
[ORDER BY RANK [WITH Rank_Parameter PREFERRED]] / SUMMARISE BY TREE]
```

Where

- Approach can be either LEAF-BASED or CLUSTER-BASED
- $C_i$ – a specific class or ANY
- $C_j$ – a specific class or ANY OTHER
- Rank_Parameter can be either POPULATION or CHANGE_DISTANCE
Chapter 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 with 256 megabytes of main memory, running Microsoft Windows 2000. All the programs were written in Java, and the open source software Weka [WF05] was used to facilitate the implementation of the algorithm. In particular, the decision tree C4.5 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:

1. Generate an “old” dataset;

2. Produce a new dataset, embedding changes of different types;

3. Run the algorithm to build models and compare them;

4. 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 is 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 the data generator are summarized in Table 4.1.

Table 4.1 Parameters used by the data generator

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of clusters per class</td>
<td>Gaussian with mean = 15, standard deviation = 5</td>
</tr>
<tr>
<td>Number of examples in cluster</td>
<td>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 a number of clusters per class</td>
</tr>
<tr>
<td>Cluster mean</td>
<td>Uniform in space $[0, 1000]^d$, where $d$ is the number of dimensions</td>
</tr>
<tr>
<td>Cluster standard deviation (radius)</td>
<td>Gaussian with mean = 50, standard deviation = 15</td>
</tr>
</tbody>
</table>

4.1.1.2 Types of Changes

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

Changing class – clusters are randomly selected and their class labels are changed

Removing a cluster – all samples from a particular cluster are removed from a dataset

Adding a cluster – for an arbitrary class a new cluster with random values of mean and standard deviation is added.

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 [RSA+04] contains USA census data from 1990. A random sample containing the information on 20000 personal records is chosen in the source. The data set contains 10 numeric attributes, where "Mortgage" is chosen as the class attribute. We select "Mortgage" 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, 17460 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. 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 [WZFY03] described in section 2.2. 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 5000 samples, 5 dimensions and 5 classes. For the experiments below, we set the minimum population threshold to 0.005.

The quality of the algorithms with respect to the number of classes is shown in Figures 4.1 and 4.2.

![Accuracy with respect to the number of classes](image)

**Figure 4.1** Accuracy with respect to the number of classes

![Recall with respect to the number of classes](image)

**Figure 4.2** Recall with respect to the number of classes
Figure 4.2 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 constant, there are fewer samples in each node. Because of that, there are more nodes where the 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 the population threshold and thus are not summarized. On the other hand, remaining nodes have a high probability to contain real changes.

![Figure 4.3 Accuracy with respect to the number of dimensions](image)
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 4.5-4.6 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 an increase in the data set size. With a larger data set, it is more likely that the obtained changes reflect randomness in the distribution rather than the changes embedded in the data generator.
Figure 4.5  Accuracy with respect to the dataset size

Figure 4.6  Recall with respect to the dataset size

Figure 4.6 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 the 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, Figures 4.1-4.6 show that CEDT and Correspondence Tracing have comparable effectiveness. However, in comparison with CEDT, Correspondence Tracing requires at least one dataset to identify changes in data, therefore it 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.

![Figure 4.7 Accuracy with respect to minimum population threshold](image)

Figure 4.7 Accuracy with respect to minimum population threshold
From Figure 4.8 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.
As described in section 3.2.2.2, the number of clusters formed in a node is inversely proportional to the cluster minimum population threshold. Therefore, figures 4.9 and 4.10 show that the effectiveness of the algorithm drops with the decrease in the number of clusters. However, they also show that both proposed approaches are not very sensitive to this threshold. Only when the number of clusters decreases significantly is the decrease in efficiency noticeable.

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.
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.

Figure 4.15 shows 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.

![Figure 4.15 Number of changing regions returned by the algorithm with respect to the number of classes](image-url)
Figure 4.16 Number of found changing regions with respect to the number of dimensions

Figure 4.17 Number of found changing regions with respect to the data set size
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 Figure 4.8, the recall of this method decreases as well.

Figure 4.18 Number of changing regions found by the cluster-based approach with respect to the minimum population threshold

Figure 4.19 Number of changing regions returned by the leaf-based approach with respect to the minimum population threshold
4.2.2 Mortgage Dataset

Here we report the changes found for the different subpopulations: “black” versus “white” (Table 4.2), “white” versus “American Indian” (Table 4.3) and “black” versus “American Indian” (Table 4.4)

Table 4.2 Changes found from black versus white

<table>
<thead>
<tr>
<th>House value</th>
<th>Age</th>
<th>Class label for “white”</th>
<th>Class label for “black”</th>
</tr>
</thead>
<tbody>
<tr>
<td>12500-37500</td>
<td>59-63</td>
<td>No mortgage</td>
<td>Mortgage</td>
</tr>
</tbody>
</table>

From the Table 4.2 we can see that for a senior age group “black” people are still paying off the mortgage.

Table 4.3 Changes found from white versus American Indian

<table>
<thead>
<tr>
<th>House value</th>
<th>Age</th>
<th>Education</th>
<th>Income</th>
<th>Class label for “white”</th>
<th>Class label for “American Indian”</th>
<th>Change distance</th>
<th>Population proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;225000</td>
<td>&lt;51</td>
<td>any</td>
<td>any</td>
<td>mortgage</td>
<td>No mortgage</td>
<td>0.73</td>
<td>0.075</td>
</tr>
<tr>
<td>5000-32500</td>
<td>&lt;48</td>
<td>any</td>
<td>any</td>
<td>mortgage</td>
<td>No mortgage</td>
<td>0.34</td>
<td>0.142</td>
</tr>
<tr>
<td>62500-67500</td>
<td>&lt;51</td>
<td>&lt;10</td>
<td>any</td>
<td>mortgage</td>
<td>No mortgage</td>
<td>0.57</td>
<td>0.048</td>
</tr>
<tr>
<td>37500-42500</td>
<td>13-48</td>
<td>any</td>
<td>&lt;9500</td>
<td>mortgage</td>
<td>No mortgage</td>
<td>0.65</td>
<td>0.008</td>
</tr>
</tbody>
</table>

Table 4.4 Changes found from white versus American Indian
Tables 4.3 and 4.4 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.

### 4.2.2.1 Measuring the Quality of the Algorithm on Real Data Sets

Apart from obtaining changes that are meaningful, can we actually measure the accuracy of the algorithm on real data sets?

For the classification task, we can use the test set with known class labels to test the accuracy of the algorithm. When using a synthetic dataset, we can still measure the accuracy of the change mining task by analyzing generator parameters. However, changes occurred in the real datasets are not known. So the question is how can we measure accuracy of the change mining algorithm without any prior knowledge about the changes?

Even having two datasets in hand, finding changing regions is not an easy task. Several algorithms mentioned in the section 2 [LHHX00, WZFY03] are proposed to solve this problem. They try to identify non-trivial changing areas that are meaningful to the user. We, on the other hand, need some method to simply return all the changing regions as accurately as possible. This method should be precise enough to be used as an accuracy standard.

Here, we use a grid-based method. The general idea is as follows. The space is partitioned into a set of small, disjoint hyper-rectangular cells. Each dimension is
partitioned into $k$ intervals of equal length. A cell is the intersection of one interval from each dimension.

Clearly, the whole domain is divided into $k^n$ hyper-rectangular cells. An object $o = (x_1, \ldots, x_n)$ is said to be contained in a cell $c = \{w_1, \ldots, w_n\}$, if $l_i \leq x_i < u_i$ for all $w_i$.

By examining the samples in both datasets we can determine the “old” and “new” class labels for each cell. Later, cells can be used as training samples for a decision tree construction, as described in section 3.3.2.1. The decision tree built using a grid method is compared with a decision tree presenting changes found by the CEDT algorithm.

For the experimental study, we use 10-fold cross validation to test the performance of the algorithm. Training set is used to build decision trees, and then we compare results of classification among two decision trees on the test set.

The accuracy on the “Mortgage” dataset measured as described above is 90%. Notice that a grid method has a huge space complexity, requiring us to store $k^n$ counters, which is infeasible in high-dimensional space. However, CEDT provides the accuracy level comparable with that of a grid method. It can be explained by the fact that for a grid method the division into the cells is fixed and cannot be controlled. A decision tree, on the other hand, partitions the space into relatively pure areas. It allows us to store only as many leaves as needed for a particular dataset.

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. The 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 Figure 4.20 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.

Figure 4.21 Clustering and change mining time with respect to the number of dimensions
The time required for change detection stays constant with an increase in the number of dimensions and increases linearly with an increase in data set size. 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 Figure 4.13. Clustering time increases linearly with
the number of dimensions and with an increase in the number of samples. Recall that the K-means clustering algorithm is used, and it is linear in the number of objects.
Chapter 5
Conclusions and Future Research

Detecting important changes and developing strategies for adapting to them is important in the changing world. In this thesis, we focus on the problem of mining changes accurately and efficiently.

In this chapter, we first summarize the thesis, and then discuss some interesting future directions.

5.1 Summary of the Thesis

Change detection has been studied extensively in both data mining and machine learning. Most of the work, however, is focused on identifying global changes rather than detecting a set of changing regions. A few studies on mining regions of changes require at least one original data set. However, in some important applications, the original data may not be available. In this thesis, we propose a model-based approach to detect and analyze changes without accessing the original data and make the following contributions:

We develop a two-phase framework for changing regions detection from access-constrained data sets. In the phase of model construction, every data set is summarized using a model. In the phase of change detection, the models are used to detect changing regions between any two specified data sets. The original data at this point is not available.

We show that existing models are not suitable to use in the model-construction phase. Therefore, we develop a novel model called cluster embedded decision tree (CEDT for short). CEDT is an extended decision tree model, where clusters are attached to leaf nodes to capture the spatial distribution of data objects. Comparing to a decision tree, a CEDT is much more accurate in capturing the data distribution and the locality with minor space overhead. That leads to accurate change detection when two models are compared.
We develop a method to compare two cluster-embedded decision trees. Naïve model comparison may enumerate many differences between two models. However, many of them may be trivial or caused by noise. Here, we develop two approaches that can identify changing regions accurately. The leaf-based approach returns several broad changing regions that can be ranked according to the user preferences. The cluster-based approach is capable of finding many subtle changes between datasets. To present the changes in a convenient way, changing regions can be summarized by another decision tree. Depending on the preference of the user, either of the approaches can be chosen.

We propose a heuristic method effective in practice to estimate populations of the regions. Since original datasets are not available, we use information from CEDTs for that purpose.

We conduct an extensive experimental study on both real and synthetic datasets. Our performance study shows that the CEDT method is both accurate and efficient. The experiments on the synthetic data sets show high accuracy and recall. Our method produces results comparable to those obtained by Correspondence Tracing [WZFY03], and furthermore our method can be used on the data sets with access constraints. Experiments on real data sets with census data show that the CEDT method is able to identify meaningful changing regions. Also, we propose a grid-based method to estimate accuracy on real data sets.

5.2 Future Research Directions

There are several aspects of the algorithms that can be improved.

At the moment, only the datasets with numerical values can be analyzed. It would be beneficial to provide a way of dealing with symbolic attributes.

On the model construction phase, we have chosen a K-means clustering algorithm to summarize the objects. Although the experimental study demonstrates the high efficiency of this method, it would be interesting to develop a special clustering method for this task. Recall that we approximate the clusters returned by K-means with hyper-rectangles.
Therefore, a special algorithm outputting clusters in a hyper-rectangular shape may achieve higher accuracy.

In this thesis we consider only one type of change: change in the class label. Recall that we define a changing region as follows:

Given two data sets $D_1$ and $D_2$ and a minimum population threshold $\text{min\_population}$ a region $R$ is called a \textit{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) $\text{pop}_{D_1}(R) \geq \text{min\_population}$
3) $\text{pop}_{D_2}(R) \geq \text{min\_population}$

However, a broader notion of change is also possible. For example, the user may be interested in the regions where the population has significantly changed from $D_1$ to $D_2$, even if they are assigned to the same class. Another example is identifying the regions that are populated only by samples from class $C_1$ in data set $D_1$, while in $D_2$ the same region contains a mixture of samples from $C_1$ and $C_2$. An interesting task would be to explore more complicated types of changes and develop an algorithm to identify all of them.

We have proposed two approaches to change detection that provide different detail level. More research can be done to develop an algorithm combining two approaches.
Bibliography


63
