Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 184 (2008) 636–651
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Computing, Artificial Intelligence and Information Management

A dynamic-programming algorithm for hierarchical discretization of continuous attributes

Ching-Cheng Shen [a], Yen-Liang Chen [b,*]

[a] Department of Information Management, Vanung University, Chung-Li 320, Taiwan, ROC
[b] Department of Information Management, National Central University, Chung-Li 320, Taiwan, ROC

Received 22 September 2005; accepted 6 December 2006
Available online 22 December 2006

Abstract

Discretization techniques can be used to reduce the number of values for a given continuous attribute, and a concept hierarchy can be used to define a discretization of a given continuous attribute. Traditional methods of building a concept hierarchy from a continuous attribute are usually based on the level-wise approach. Unfortunately, this approach suffers from three weaknesses: (1) it only seeks a local optimal solution instead of a global optimal, (2) it is usually subject to the constraint that each interval can only be partitioned into a fixed number of subintervals, and (3) the constructed tree may be unbalanced. In view of these weaknesses, this paper develops a new algorithm based on dynamic-programming strategy for constructing concept hierarchies from continuous attributes. The constructed trees have three merits: (1) they are global optimal trees, (2) each interval is partitioned into the most appropriate number of subintervals, and (3) the trees are balanced. Finally, we carry out an experimental study using real data to show its efficiency and effectiveness.

Keywords: Dynamic programming; Concept hierarchy; Data mining; Continuous data

1. Introduction

The task of attribute-discretization techniques is to discretize the values of continuous attributes into a small number of intervals. Interval labels can then be used to replace actual data values. Many flat discretization techniques have been proposed to discretize continuous attributes (Han and Kamber, 2001). By applying these discretization techniques recursively, we can obtain a hierarchical discretization of the attribute values, known as a concept hierarchy. Concept hierarchies can be used to reduce the data by collecting and replacing low-level concepts with higher-level concepts.

Since concept hierarchies are a useful structure in which to organize the values of continuous attributes, it has been widely used in many applications, such as data warehousing and data mining. In data warehousing,

* Corresponding author. Tel.: +886 3 4267266; fax: +886 3 4254604.
E-mail address: ylchen@mgt.ncu.edu.tw (Y.-L. Chen).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2006.12.013

one aspect of designing a data warehouse scheme is specifying concept hierarchies for attributes, so that users can roll up or drill down along the specified concept hierarchies; see Codd et al. (1993) or Chaudhuri and Dayal (1997). In data mining, concept hierarchies have been used in at least three sub-areas, including the attribute-oriented induction method (Cai et al., 1990; Han et al., 1992, 1993; Han and Fu, 1994; Chen and Shen, 2005), association rules mining (Srikant and Agrawal, 1995; Han and Fu, 1999; Hong et al., 2003), and sequential patterns mining (Srikant and Agrawal, 1996; Chen and Ye, 2004). Using concept hierarchies, the researchers in these sub-areas have developed various algorithms to discover generalized knowledge, multiple-level rules, or multiple-level sequential patterns from databases. For example, the attribute-oriented induction method generalizes a great bulk of relational data into a small set of generalized knowledge by repeatedly rolling up the attribute values along the concept hierarchies. Therefore, a preprocess task that must be done in all these applications, whether data warehousing or data mining, is to define appropriate concept hierarchies for the selected continuous attributes.

Since building concept hierarchies for continuous attributes is essential for performing data mining or data warehousing tasks, various methods have been proposed. The existing methods can be roughly classified into two major approaches. The first approach is building concept hierarchies manually by domain experts. Although this approach may be the most intelligent, it can be a tedious and time-consuming task for the user or domain expert. If we only need to process a few continuous attributes, the method may work properly. But if we have numerous attributes, or if the costs of the experts are high, or if the experts are out of jobs, then this approach would not be successful. Another approach is to build concept hierarchies automatically using the level-wise partitioning approach. First, we partition the whole range into several disjoint intervals. After that, we recursively partition each interval into smaller intervals. This process continues until the depth of the tree has reached a certain limit or the amount of data in a node is less than a given threshold. There are several partitioning methods available for determining how to partition an interval (or a node). The major methods include equal-depth partition (Han and Fu, 1994), equal-width partition, chi-square partition (Kerber, 1992), entropy-based partition (Quinlan, 1993; Fayyad and Irani, 1993), and clustering partition (MacQueen, 1967; Kaufman and Rousseeuw, 1990). The equal-depth partition splits an interval into several subintervals with the same number of data objects, while the equal-width partition splits an interval into several subintervals of the same length. Suppose we are given a fixed value of K. The entropy-based approach would repeatedly partition an interval into two subintervals with the best entropy value until K subintervals are generated, while the chi-square partition measures its efficiency by the chi-square test results. Finally, the clustering method first partitions an interval into several clusters of data objects that are close in proximity. Then, the range of values of data objects in each cluster forms a subinterval. No matter which method is used to partition an interval, the second approach can be viewed as a greedy approach, because it considers the partition level-by-level without considering global optimization.

The second approach, the level-wise partition approach, has three drawbacks. First, it only seeks local optimization instead of global optimization, meaning when partitioning an interval, it never considers how this partition may affect the partitions in the lower levels or even the entire tree. For example, entropy-based methods only consider how the entropy of the current node can be optimized in the current partition. Unfortunately, a good partition for a single node does not necessarily equal having good entropy values in future partitions or the entire tree.

Second, the level-wise approach usually partitions a node, or interval, into a fixed number of subintervals in all levels and in all nodes. They lack the flexibility of having different numbers of partitions in different nodes. Since data distributions of attributes may differ, it is possible that the most appropriate numbers of subintervals in different nodes may differ. Therefore, an improvement would be having different numbers of subintervals in different nodes, according to the data distributions in those nodes.

Third, building a concept hierarchy using the level-wise approach may result in an unbalanced tree, meaning leaf nodes may be in different levels of the tree. A simple example to explain why this occurs would be as follows: suppose we have 100 pieces of data spread over a certain interval and the number of subintervals in each partition is three. Assume that according to our greedy partition method, these 100 pieces of data are partitioned into three sets with 46, 53, and 1 data object(s), respectively. It is clear that the third interval cannot be further partitioned; therefore, it stops here, but the other two can be further partitioned repeatedly. This will result in an unbalanced tree. Unfortunately, most data mining or data warehousing applications