Hierarchical Discretization of Continuous Attributes Using Dynamic Programming

Amit Goyal
MSc Computer Science
University of British Columbia
Vancouver, BC, Canada
goyal@cs.ubc.ca

ABSTRACT
The area of Knowledge discovery and Data mining is growing rapidly. A large number of methods are employed to mine knowledge. Several of the methods rely of discrete data. However, most datasets used in real application have attributes with continuous values. To make the data mining techniques useful for such datasets, discretization is performed as a pre-processing step. Discretization is a process of converting the continuous attributes of a data set into discrete ones so that data mining algorithms can be applied efficiently and effectively. This study is done as a course requirement of CS532 in University of British Columbia. In this project, we did a literature survey on various methods traditionally used for discretization. Then we examine a new algorithm based on dynamic programming strategy and compare them with other methods.

Keywords
Dynamic Programming, Discretization of Continuous Attributes, Pattern Mining, Knowledge Discovery, Concept Hierarchy, Concept Trees.

1. INTRODUCTION
Tremendous amounts of data is being sampled and collected each day. Much of these data are of no direct use. Intelligent automated techniques are needed to extract knowledge from all this data. Several methods exist for knowledge discovery, and the research in this area is extensive from different angles of attack. Several of the traditional fields of Artificial Intelligence (AI) research can be applied to this area, combined with new ideas from pattern recognition and statistics. Combining all these areas of research is a great challenge to the Knowledge discovery community. This challenge, along with the enormous capabilities of such systems, makes this a very exciting area of research.

The process of Knowledge Discovery in databases can be summarized in 4 steps as shown in Fig 1.1. In the first step, data is collected and pre-processed to make it suitable for pattern discovery. The next step discovers interesting pattern in the data and analyze them. According to the nature of patterns discovered, the recommendations are made and then the suitable actions are taken to maximize the gain. Then the data is collected again and this process continues in a feedback circle. In this project, we focus on a pre-processing step in knowledge discovery known as discretization. Discretization is a process that transforms continuous attributes into a finite number of intervals, where each interval is associated with a numerical discrete value. 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.

Fig. 1.1. Knowledge Discovery In Databases.

There are many advantages of using discrete values over continuous ones. The most important is that data can be reduced and simplified through discretization. Discretization of data has the effect of increasing the speed and accuracy of machine learning and data mining. In machine learning, results obtained through decision trees or induction rules using discretized data are usually more compact, shorter, and more accurate than results derived using continuous values [6]. In data mining, concept hierarchies have been used in at least three sub-areas, including the attribute-oriented induction method [7,8,11], association rules mining [9], and sequential patterns mining [10]. The attribute-oriented induction 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. Association rule mining is the process of discovering regularities between products in large scale transaction data recorded.
by point-of-sale (POS) systems in supermarkets. For example, the rule \{onions, vegetables\} => \{beef\} found in the sales data of a supermarket would indicate that if a customer buys onions and vegetables together, he or she is likely to also buy beef. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. Sequential pattern mining finds the relationships between occurrences of sequential events, to find if there exist any specific orders of the occurrences. We can find the sequential patterns of specific individual items; also we can find the sequential patterns cross different items. Sequential pattern mining is widely used in analyzing of DNA sequence. An example of sequential patterns is that every time Microsoft stock drops 5%, IBM stock will also drops at least 4% within three days.

Traditional methods of building a concept hierarchy from a continuous attribute are usually based on the level-wise approach. Unfortunately, this approach suffers from two 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. In view of these weaknesses, this project looks into a new algorithm based on dynamic-programming as described in [5]. The paper defines a volume function to measure the quality of a concept tree, which can be expressed in terms of volume of the child nodes recursively. Thus, using a Bellman type equation, a global optimum policy can be achieved.

The rest of the report is organized as follows: In Section 2, we looked into various strategies used for discretization. Section 3 formalizes the problem definition. The detailed dynamic programming algorithm is described in Section 4. In section 5, we compare the results with other approaches. Finally, conclusions are drawn in section 6.

2. RELATED WORK

Several methods have been proposed to discretize data as a pre-processing step for the data mining process. In this survey, we investigate on the most popular methods such as equal-width discretization, equal-frequency discretization [11], chi-square discretization [1,2], entropy-based discretization [8,9], and clustering based discretization[16].

2.1 Equal width discretization

This is the simplest approach to discretization. It divides the range into the user specified $N$ intervals of equal size. If $A$ and $B$ are the lowest and highest values of the attribute, the width of intervals will be: $W = (B-A)/N$. The equal width method assumes that the underlying data fits somewhat well into a uniform distribution. It is very sensible to out-layers, and can perform extremely bad under some conditions.

2.2 Equal frequency discretization

The equal frequency discretization algorithm first sorts the values of the attribute in an ascending (or descending) order, and then divide them into the user specified $N$ intervals, in such a way that each interval contains the same number of samples. It works very good in most of the cases.

2.3 Chi square based discretization

Chi-square ($\chi^2$) is a statistical measure that conducts a significance test on the relationship between the values of a feature and the class. Kerber [2] argues that in an accurate discretization, the relative class frequencies should be fairly consistent within an interval (otherwise the interval should be split to express this difference) but two adjacent intervals should not have similar relative class frequency (in that case the adjacent intervals should be merged into one). The $\chi^2$ is defined as follows:

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

This statistic determines the similarity of adjacent intervals based on some significance level. It tests the hypothesis that two adjacent intervals of an attribute are independent of the class. If they are independent, they should be merged; otherwise they should remain separate. As an example of the use of the Chi-square test, a fair coin is one where heads and tails are equally likely to turn up after it is flipped. Suppose one is given a coin and asked to test if it is fair. After 100 trials, heads turn up 53 times and tails result 47 times. The following is a Chi-square analysis, where the null hypothesis is that the coin is fair:

<table>
<thead>
<tr>
<th></th>
<th>Heads</th>
<th>Tails</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
<td>53</td>
<td>47</td>
<td>100</td>
</tr>
<tr>
<td>Expected</td>
<td>50</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>$(O-E)^2$</td>
<td>9</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>$\chi^2$ = $(O-E)^2/E$</td>
<td>0.18</td>
<td>0.18</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Table 1: Chi-Square Test for fairness of a coin

In this case, the test has one degree of freedom and the chi-square value is 0.36. In order to see whether this result is statistically significant, the P-value (the probability of this result not being due to chance) must be calculated or looked up in a chart. The P-value is found to be $\text{Prob}(\chi^2 \geq 0.36) = 0.5485$. There is thus a probability of about 55% of seeing data that deviates at least this much from the expected results if indeed the coin is fair. This probability is not considered statistically significant evidence of an unfair coin.
The top-down method based on chi-square is ChiSplit. It searches for the best split of an interval, by maximizing the chi-square criterion applied to the two sub-intervals adjacent to the splitting point: the interval is split if both sub-intervals substantially differ statistically. The ChiSplit stopping rule is based on a user-defined chi-square threshold to reject the split if the two sub-intervals are too similar.

The bottom-up method based on chi-square is ChiMerge [2]. It searches for the best merge of adjacent intervals by minimizing the chi-square criterion applied locally to two adjacent intervals: they are merged if they are statistically similar. The stopping rule is based on a user-defined Chi-square threshold to reject the merge if the two adjacent intervals are insufficiently similar. No definite rule is given to choose this threshold.

Boulle [13] proposes the discretization method Khiops, based on the chi-square statistic. In contrast with related methods ChiMerge and ChiSplit, this method optimizes the Chi-square criterion in a global manner on the whole discretization domain and does not require any stopping criterion. The Khiops method starts the discretization from the elementary single value intervals. It evaluates all criteria. The Khiops method starts the discretization domain and does not require any stopping criterion. The Chi-square criterion in a global manner on the whole discretization domain and does not require any stopping criterion. The Khiops method starts the discretization from the elementary single value intervals. It evaluates all criteria.

2.4 Entropy based discretization

Discretization based on entropy criteria have been discussed in several papers, and will be investigated in this section. Entropy discretization is a supervised method, proposed by Fayyad and Irani [8], also discussed in [9]. If there are r distinct data points in a class S (or node of a tree) with frequencies \(f_1, \ldots, f_r\), then the entropy of the node is defined as follows:

\[
\text{Ent}(S) = \sum_{i=1}^{r} \left( \frac{f_i}{|S|} \log_2 \frac{|S|}{f_i} \right)
\]

This class information entropy is a measure of purity and it measures the amount of information which would be needed to specify to which class an instance belongs.

If the node is partitioned into 2 intervals \(S_1, S_2\), then the entropy after partitioning is

\[
\text{Ent}(\text{Split}(S)) = \left| \frac{S_1}{|S|} \right| \text{Ent}(S_1) + \left| \frac{S_2}{|S|} \right| \text{Ent}(S_2)
\]

The boundary that minimizes the entropy function over all possible boundaries is selected as a binary discretization. The process is recursively applied to partitions obtained until some stopping criterion, for example, Minimum Description Length (MDL) Principle or an optimal number of intervals are achieved.

Any set of data can be represented by a string of symbols from a finite (say, binary) alphabet. "The fundamental idea behind the MDL Principle is that any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally." [14]. Thus, the MDL principle states that the best hypothesis is the one with minimal description length. As partitioning always decreases the value of the entropy function, considering the description lengths of the hypotheses allows balancing the entropy gain and eventually accepting the null hypothesis. Performing recursive bipartitions with this criterion leads to a discretization of the continuous explanatory attribute at hand.

2.5 Clustering based discretization

Clustering is the process of grouping the data such that objects within a cluster have higher similarity in comparison to objects in other clusters [4]. Cluster analysis has been widely used in various disciplines such as pattern recognition, computer vision and data mining [15]. There are numerous types of clustering algorithms partitioning, hierarchical, nearest-neighbours, fuzzy set theory, density-based and grid-based approaches etc.

In this report, we restricted our survey on hierarchical clustering only. In hierarchical clustering, the algorithm begins with all objects in a single large cluster and proceeds to sequentially divide them into smaller clusters. Hierarchical Clustering is subdivided into agglomerative methods, which proceed by series of fusions of the n objects into groups, and divisive methods, which separate n objects successively into finer groupings. Agglomerative techniques are more commonly used [17]. In a hierarchical agglomerative, each cluster consists of one object. The clusters are combined step by step. In each step those two clusters with the smallest dissimilarity or the highest similarity are merged. Iteration continues until all objects are in one single cluster.

3. PROBLEM FORMULATION

In Section 3.1, we first introduce the input data and input parameters of the problem. Next, we formally define the problem in Sections 3.2, and 3.3 elaborate on some distance and volume definitions. Finally, this information is used to develop a tree construction algorithm discussed in Section 4.
3.1 Input Data

Usually, the raw input data is stored in log files having various attributes, e.g., a person’s metadata can be represented as <person_id, age, sex, income, phone bills, occupation>. Here, age, income and phone bills are numeric continuous attributes. Since we discretize one such attribute at a time, we select one attribute and simultaneously find the frequency of each data point of that attribute. i.e., we calculate the value of data point and its frequency in the form <value, frequency>. Thus, after this pre-processing, the input data becomes \( \text{val}(1), \text{val}(2), \ldots, \text{val}(n) \) and \( \text{num}(1), \text{num}(2), \ldots, \text{num}(n) \), where \( \text{val}(i) \) is the value of data and \( \text{num}(i) \) is the number of occurrences of value \( \text{val}(i) \) for \( 1 \leq i \leq n \) and \( \text{val}(i) = \text{val}(i+1) \) for \( 1 \leq i < n-1 \). For example, suppose that we have 20 pieces of data, where five of them are 1’s, three are 2’s, five are 4’s, five are 6’s, and two are 9’s. Then, we have \( \text{num}(1)=5, \text{val}(1)=1, \text{num}(2)=3, \text{val}(2)=2, \text{num}(3)=5, \text{val}(3)=4, \text{num}(4)=5, \text{val}(4)=6, \text{num}(5)=2, \text{val}(5)=9 \). For ease of reference, we will refer to data containing \( \text{val}(i) \) and \( \text{num}(i) \) as data \( i \).

**Fig.3.1. An example of concept hierarchy.**

We now define \( R \) as the depth of output tree, \( ub \) as the upper boundary on the number of subintervals spawned from an interval and \( lb \) as the lower boundary. These three parameters specify that the constructed tree must be of depth \( R \) and the number of children in each node is between \( lb \) and \( ub \). Furthermore, we require all leaf nodes to be in the bottom level. For example, Fig. 3.1 shows a concept tree with \( R = 2, ub = 3 \), and \( lb = 2 \).

3.2 Problem Definition

A good concept hierarchy should contain nodes that correspond to concepts embodied in the data. Thus, a good concept tree should satisfy at least two goals: (1) data in the same node should be as similar as possible and (2) data between different nodes should be as different as possible. In this project, we quantify the similarity of data points within a node as intra-distance of the node and differences of data points between the nodes as inter-distance between the nodes. Hence, an optimal concept tree has inter-distances maximized and intra-distance minimized. Thus, our problem is how to minimize the intra-distance inside the same nodes while maximizing the inter-distance between different nodes. An immediate difficulty is that we may have different ways of realizing the second goal. For example, there are three well-known methods for computing the inter-distance among clusters: complete-link, single-link, and average-link methods [4]. Therefore, we can define the inter-distance between different nodes in three ways: the summation of the largest differences for all pairs of sibling nodes (complete-link), the summation of the smallest differences for all pairs of sibling nodes (single-link), and the average of differences for all pairs of sibling nodes (average-link). Since the distance between two siblings can be naturally measured by the gap of values from the left sibling to the right sibling, this paper defines the inter-distance of two sibling nodes as the result of subtracting the largest value of the left sibling from the smallest value of the right sibling, which is the single-link method.

Using the tree in Fig. 3.1 as an example, there are ten nodes in the tree. Among these ten nodes, there are five pairs of adjacent sibling nodes: nodes 1 and 2, nodes 3 and 4, nodes 5 and 6, nodes 7 and 8, and nodes 8 and 9. In building the tree, we need to minimize the intra-distance inside the same nodes while maximizing the inter-distance between adjacent sibling nodes. Therefore, this paper defines the volume of a tree as the result of the total intra-distance minus the total inter-distance. Since these two types of distances may not be equal-valued, however, we use another parameter \( \beta \) to reflect the relative importance between the inter-distance and intra-distance. We formally define the problem as follows.

**Problem Definition.** Given parameters \( R, ub, lb \) and \( \beta \) input data \( \text{val}(1), \text{val}(2), \ldots, \text{val}(n) \) and \( \text{num}(1), \text{num}(2), \ldots, \text{num}(n) \), our goal is to build a minimum volume tree subject to the constraints that all leaf nodes must be in level \( R \) and that the branch degree must be between \( ub \) and \( lb \).

3.3 Distances and Volume

In the previous section, we talked about the ideas to quantify the similarity and differences of the data points in and between the nodes. This section will formally give definitions of distances and the volume of a tree. Let \( \text{intradist}(i, j) \) be the intra-distance of a node containing data from data \( i \) to data \( j \), \( \text{mean}(i, j) \) be the mean of the data in this node, and \( \text{totalnum}(i, j) \) be the total number of data occurrences in this node. We define the intra-distance of this node as follows:

\[
\text{intradist}(i, j) = \sum_{x=i}^{j} \text{val}(x) - \text{mean}(i, j) \right) \times \text{num}(x)
\]

where \( \text{val}(x) \) is the numeric value of data \( x \). Furthermore, the following defines the inter-distance \( \text{interdist}(i, j, u) \) between two adjacent sibling nodes, where the first node contains data from data \( i \) to data \( u \) and the second contains data from data \( u + 1 \) to data \( j \). Since the value gap between
the two nodes is \( \text{val}(u + 1) - \text{val}(u) \), we define its inter-distance as the multiplication of the value gap between the two nodes and the total number of data occurrences in these two nodes

\[
\text{interdist}(i, j, u) = \beta \times (\text{val}(u + 1) - \text{val}(u)) \times \text{totalnum}(i, j)
\]

\[
= \beta \times (\text{val}(u + 1) - \text{val}(u)) \times \left( \text{totalnum}(i, u) + \text{totalnum}(u + 1, j) \right)
\]

Here, \( \text{interdist}^L(i, u) = \beta \times (\text{val}(u + 1) - \text{val}(u)) \times \text{totalnum}(i, u) \) and \( \text{interdist}^R(i, u) = \beta \times (\text{val}(u + 1) - \text{val}(u)) \times \text{totalnum}(u + 1, j) \). This is similar to single-link inter-distance as defined in [4]. The relationship indicates that the inter-distance between two nodes, one from data \( i \) to data \( u \) and the other from data \( u + 1 \) to data \( j \), can be split into two parts, where one can be totally attributed to the left node and the other from data \( u + 1 \) to data \( j \). Similarly, we can say that \( \text{interdist}(i, u) \) is the inter-distance when the node containing data \( i \) to data \( u \) is put on the right.

Next, we define the volume of a tree as the total intra-distance in the tree minus the total inter-distance in the tree. It is similar to volume definition defined in [5]. As an example, the volume of the tree in Fig. 3.1 is the sum of the ten intra-distances minus the sum of the five inter-distances. Let volume \( (i, j, T) \) denote the volume of tree \( T \) containing data from data \( i \) to data \( j \). The volume of the tree in Fig. 1 can then be written as follows:

\[
\text{volume}(1, 20, T) = \text{intradist}(1, 5) + \text{intradist}(6, 7) + \text{intradist}(8, 10) + \text{intradist}(11, 14) + \text{intradist}(15, 16) + \text{intradist}(17, 20) + \text{intradist}(11, 17) + \text{intradist}(8, 14) + \text{intradist}(15, 20) + \text{intradist}(12, 17) + \text{intradist}(8, 14, 10) - \text{intradist}(15, 20, 16) - \text{intradist}(14, 17, 7) - \text{intradist}(8, 20, 14).
\]

Similarly, from the subtrees immediately below the root, we have:

\[
\text{volume}(1, 7, T) = \text{intradist}(1, 5) + \text{intradist}(6, 7) - \text{intradist}(1, 7, 5)
\]

\[
\text{volume}(8, 14, T) = \text{intradist}(8, 10) + \text{intradist}(11, 14) - \text{intradist}(8, 14, 10).
\]

\[
\text{volume}(15, 20, T) = \text{intradist}(15, 16) + \text{intradist}(17, 20) - \text{intradist}(15, 20, 17).
\]

By substituting these three equations into the original volume equation, we find that the original \( \text{volume}(1, 20, T) \) can be rewritten as

\[
\text{volume}(1, 20, T) = \text{intradist}(1, 20) + \text{volume}(1, 7, T) + \text{volume}(8, 14, T) + \text{volume}(15, 20, T) - \text{intradist}(14, 17, 7) - \text{intradist}(8, 20, 14).
\]

The results of this example can be formally stated in the following theorem.

**Theorem 1**

The volume of a tree is the intra-distance of the root node + the volumes of all its sub-trees - the inter-distances among its children.

**Proof:**

The theorem can be proven by induction. First, we have to show the basis hold, i.e., the relationship holds for a tree without children. Next, we assume that the theorem holds for a tree with a depth less than \( h \). Finally, we must show that the volume of a tree \( T \) with depth \( h \) is equal to: the intra-distance of the root node + the volumes of all its sub-trees with depth \( h-1 \) the inter-distances among its children. Since this part of the proof is similar to the example given above, we omit the details.

### 4. Dynamic Programming Solution

In the last section, we defined the volume of a tree as a function of volumes of its sub-trees, intra-distance of the root node and inter-distance b/w its child nodes. This theorem forms the basis of dynamic programming solution for the global optimum concept tree. In section 4.1, we first introduce some notations and recursive formulas regarding tree volume. Based on these formulas, section 4.2 develops two algorithms, one that computes the tree volume and one that builds the concept tree. Finally, Section 4.3 analyzes its time complexity.

#### 4.1 Notations and recursive formulas

This section develops recursive formulas, which we can use to develop algorithms to construct concept trees from the given data. Let \( T^*(i, j, r) \) be the minimum volume tree that contains data from data \( i \) to data \( j \) and has depth \( r \). Similarly, let \( T(i, j, r, k) \) be the minimum volume tree that contains data from data \( i \) to data \( j \), has depth \( r \), and whose root has \( k \) branches. Now, let \( D^*(i, j, r) \) be the volume of \( T^*(i, j, r) \) and \( D(i, j, r, k) \) be the volume of \( T(i, j, r, k) \). Then, we define \( QD(i, j, r, k) \) as follows:

\[
QD(i, j, r, k) = \min \left\{ \sum_{v=1}^{l} \left( \text{the volume of } v^v \text{ node} \right) - \sum_{l=1}^{t} \left( \text{the interdistance b/w } v^l \text{ node and } \left( v+1 \right)^l \text{ node} \right) \right\}
\]

Recall that in theorem 1, we defined the volume of a tree as a function of volumes of its sub-trees, intra-distance of the root node and inter-distance b/w its child nodes. The term \( QD \) represents the last two terms in this volume. Next we define another term \( QD^M \) as follows:

\[
QD^M(i, j, r, k) = \min \left\{ \sum_{v=1}^{l} \left( \text{the volume of } v^v \text{ node} \right) - \text{interdistance}^{k}(i, u) \right\}
\]

\[
\sum_{l=1}^{t} \left( \text{the interdistance b/w } v^l \text{ node and } \left( v+1 \right)^l \text{ node} \right)
\]
We use Fig. 3.2 to explain the difference between $QD(i,j,r,k)$ and $QD^M(i,j,r,k)$. In this example, $QD(i,j,r,4)$ equals $D'(u,r) + D'(v+1,r) + D'(w+1,r) + D'(w+1,j,r) - \text{interdist}(i,u) - \text{interdist}(v+1,v,w) - \text{interdist}(w+1,w,j)$. $QD^M(i,j,r,k)$ is similar to $QD(i,j,r,4)$, but it has to subtract an additional inter-distance $\text{interdist}^R(i,u)$.

**Corollary 1**: $QD^M$ can be recursively defined as follows:

$$QD^M(i,j,r,k) = \min\{D^*(i,u,r) + QD^M(u+1,j,r,k-1) - \text{interdist}^R(i,u)\}$$

**Proof**: Let's assume there are $k$ sub-trees of the tree $T(i,j,r,k)$, with first two optimal split data points as $u$ and $v$. As per the definition of $QD$:

$$QD(i,j,r,k) = \min\{\sum (\text{the volume of } v^a \text{ node}) - \sum (\text{the interdistance } b/w v^a \text{ node and } (v+1)^b \text{ node})\}$$

$$= \min\{[D^*(i,u,r) + \sum (\text{the volume of } v^a \text{ node}) - \text{interdist}^k(i,u) - \sum \text{the interdistance } b/w \text{ nodes} - \text{interdist}^k(i,u)\}$$

$$= \min\{D^*(i,u,r) + QD^M(u+1,j,r,k-1) - \text{interdist}^R(i,u)\}$$

The recursion is depicted in Fig. 3.3 for $k=4$.

**Corollary 2**: $QD$ can be recursively defined as follows:

$$QD(i,j,r,k) = \min\{D^*(i,u,r) + QD^M(u+1,j,r,k-1) - \text{interdist}^R(i,u)\}$$

**Proof**: Let's assume there are $k$ sub-trees of the tree $T(i,j,r,k)$, with first two optimal split data points as $u$ and $v$. As per the definition of $QD$:

$$QD(i,j,r,k) = \min\{\sum (\text{the volume of } v^a \text{ node}) -$$

$$\sum (\text{the interdistance } b/w v^a \text{ node and } (v+1)^b \text{ node})\}$$

$$= \min\{D^*(i,u,r) + \sum (\text{the volume of } v^a \text{ node}) -$$

$$\text{interdist}^R(i,u) - \sum (\text{the distance } b/w v^a \text{ node and } (v+1)^b \text{ node})\}$$

$$= \min\{D^*(i,u,r) + QD^M(u+1,j,r,k-1) - \text{interdist}^R(i,u)\}$$

The recursion is depicted in Fig. 3.4 for $k=4$.

Using the definition of $QD$, we can define the recursion for the optimal volume $D^*(i,j,r)$ of the tree $T^*(i,j,r)$ in corollary 3:

**Corollary 3**: The volume of the minimum volume tree $T^*(i,j,r)$ is defined as follows:

$$D^*(i,j,r) = \text{intradist}(i,j) + \min_{b \leq k < \text{sub}} \{QD(i,j,r-1,k)\}$$

The proof follows from the definition.

To calculate the minimum volume tree $T^*(i,j,r)$, we use these corollaries. This is the basis of the dynamic programming solution. In the next section, we develop two algorithms, one to compute the tree volume and one to build the concept tree.
4.2 The algorithms

In any backward DP algorithm, in addition to recursion, we also need to compute the final transition cost, which we will call base case in our problem. While constructing the concept tree, we showed how to calculate minimum volume using DP in the last section. If the depth of the tree is equal to R (the maximum depth), then it’s the root level. i.e. all the data points are present in one node (root). The bottom-most level (the leaf level) corresponds to r=0, which is our base case. Once we compute the various volume terms for the base level, we can apply the DP equations to construct the whole concept tree. The whole algorithm can be written as follows:

1. For $1 \leq i, j \leq n$, compute $\text{intradist}(i,j)$
2. For $1 \leq i, j \leq n$, compute $\text{interdist}^A(i,j)$ and $\text{interdist}^B(i,j)$
3. For $1 \leq i, j \leq n$, compute $D^*(i,j,0)$ and $QD(i,j,0,1)$
4. For $1 \leq i, j \leq n$, compute $QDM(i,j,0,1)$
5. For $k=2$ to $ub$
   a. For $1 \leq i, j \leq n$, compute $QDM(i,j,0,k)$
   b. For $1 \leq i, j \leq n$, compute $QD(i,j,0,k)$
6. For $r=1$ to $R$
   a. For $1 \leq i, j \leq n$, compute $D^*(i,j,r)$
   b. For $1 \leq i, j \leq n$, compute $QDM(i,j,r,1)$
   c. For $1 \leq i, j \leq n$ and $2 \leq k \leq ub$, compute $QDM(i,j,r,k)$
   d. For $1 \leq i, j \leq n$ and $2 \leq k \leq ub$, compute $QD(i,j,r,k)$

Steps 3 to 5 corresponds to the base case ($r=0$). At this level (leaf level), the following relations hold by definition:

- $D^*(i,j,0) = \text{intradist}(i,j)$
- $QD(i,j,0,1) = \text{intradist}(i,j)$
- $QDM(i,j,0,1) = D^*(i,j,0) - \text{interdist}^B(i,j)$

To compute $QDM(i,j,0,k)$ and $QD(i,j,0,k)$ in step 5, we exploit the corollary 1 and 2. The terms in step 6 can be computed using the same corollaries. We save all $D^*$’s, $QD$’s and $QDM$’s in a data structure as they are needed to build the concept tree.

To build concept trees from the results above, we need to define some other symbols. Let $QDM(i,j,r,k).u$, $QD(i,j,r,k).u$ denote the values of $u$ achieving the minimum volume in corollary 1 and 2, and let $D'(i,j,r,k)$ denote the value of $k$ achieving the minimum volume in corollary 3. The following procedure shows how to build a concept tree from $D'(i,j,r)$. To make the procedure easier to understand, we show the relations in Fig. 3.5, where we assume $k=5$.

1. If $r=0$ then build the node containing the data from data $i$ to data $j$ and exit (base case).
2. Let $k=D'(i,j,r).k$, which is the number of subtrees in level $r-1$.
3. Output $u_i=QD(i,j,r-1,k).u$, which is the right end of the first node in level $r-1$.
4. Recursively output the concept tree for $D'(i,u_i,r-1)$.
5. For $z=2$ to $k-1$
6. Output $u_c=QD'(u_c+1,j,r-1,k-z+1).u$, which is the right end of the $z^{th}$ node in level $r-1$.
7. Recursively output the concept tree for $D'(u_c+1,u_c,r-1)$.
8. Endfor
9. Recursively output the concept tree for $D'(u_k+1,j,r-1)$.

4.3 Time complexity

The most time-consuming computation in the algorithm is the computation of $QDM(i,j,r,k)$ and $QD(i,j,r,k)$ in steps 6c and 6d for $1 \leq i, j \leq n$ and $2 \leq k \leq ub$ and $1 \leq r \leq R$. In total, we need to compute at most $O(n^2 \times ub \times R)$ instances of $QD$ or $QDM$. Furthermore, from corollary 1 and 2, we see that each such computation can be done in time $O(n)$. As a result, the time complexity of the algorithm is $O(n^2 \times ub \times R)$. Note that here ‘n’ is the number of distinct data points, which is much less than the total number of data points.

5. EXPERIMENTAL EVALUATION

As a part of the project, I implemented the DP algorithm described in Section 4. All the experiments are performed on a 3000 MHz Intel Pentium 4 CPU with 2 GB main memory, running Suse Linux 10.1 with kernel 2.6.16.54. The algorithms were implemented in C++ with STL and compiled with gnu g++ compiler version 4.1.0.

5.1 Data Sets

The dataset used in [5] is not available publicly. So, we couldn’t compare our results with those in [5]. We use Iris flower dataset [12] on the attribute Sepal length scaled by the factor of 10. There are total of 150 data points in the dataset and 35 distinct values in the dataset. The data distribution is shown in Fig. 5.1. For comparing run-time, we use an artificial dataset containing 100,000 data points, created using random number generation algorithm. This dataset contains 100 distinct data points.
5.2 Results
The main focus of the project is on the Dynamic Programming element for the hierarchical discretization of continuous attributes. So, we didn’t invest much time in implementing the other approaches discussed in related work section. Thus, we restricted our experimental comparison study with Equal-frequency algorithm only.

Fig 5.1 Iris flower dataset distribution for Sepal Length
The results are shown in Fig. 5.2, revealing that DP algorithm is much more effective than Equal-frequency algorithm. The parameters $\beta$, $lb$ and $ub$ is fixed to 5, 2 and 2 respectively and the parameter $R$ (the maximum depth of the tree) is varied from 2 to 5. Clearly, the volume of the concept tree is lower in case of DP, as it is globally optimized concept tree.

Fig 5.2 The volumes of the trees constructed for different depth values ($R$)
Next, we keep the parameters $R$, $\beta$ fixed to 3 and 5 respectively. The results are shown in Fig 5.3. The parameter $lb$, $ub$ are varied from 2 to 5.

Fig 5.3 The volumes of the trees constructed for different boundary values ($k=lb=ub$)
We also analyze the effect of the parameter $\beta$ on the volume of the concept tree for both the algorithms. Fig 5.4 shows the results obtained. As $\beta$ increases, the significance of inter-distances increases compared to intra-distances, resulting in the drop of volumes. As clear from the figure, the increase in $\beta$ results in widening the gap between volumes obtained from DP and Equal-Frequency. It is an interesting result as it proves that when we increase the significance of inter-distances, the nodes in the concept tree obtained from DP algorithm are more compact compared to Equal-Frequency algorithm.

Fig 5.4 The concept hierarchy tree for attribute sepal length built by DP algorithm ($\beta=5$, $lb=0$, $ub=4$, $R=4$)
To demonstrate how DP algorithm gives us the optimal branching, a concept tree for the dataset is shown in Fig 5.5. The parameters $lb$, $ub$ and $R$ is set to 0, 4 and 4 respectively.

Fig 5.5 The concept hierarchy tree for attribute sepal length built by DP algorithm ($\beta=5$, $lb=0$, $ub=4$, $R=4$)
Finally, we see the effect of parameters $R$ and $ub$ on the runtime in Fig. 5.6. As expected, the runtime increases with $R$ and $ub$. E.g. when $ub=4$ and $R=4$, the time taken to build the concept tree is 79 sec.

6. CONCLUSION
A number of Machine Learning and Data Mining algorithms can be applied only to data described by discrete numerical or nominal attributes. In the case of continuous attributes, there is a need for a discretization algorithm that transforms continuous attributes into discrete ones. Continuous-valued attributes are transformed into discrete ones by splitting the range of the attribute values in
a finite number of intervals. Since a large number of possible attribute values slows and makes pattern discovery ineffective, one of the main goals of a discretization algorithm is to significantly reduce the number of discrete intervals of a continuous attribute. At the same time, the algorithm should minimize the information loss due to discretization. A satisfactory trade off between these two goals needs to be achieved. Many studies show induction tasks can benefit from discretization: rules with discrete values are normally shorter and more understandable and discretization can lead to improved predictive accuracy.

In this project, we studied various approaches to solve the problem of discretization. Most of the traditional approaches are local optimum, thus the resultant concept tree in not globally optimized. We then exploited Dynamic Programming to construct global optimal concept tree as proposed in [5]. We also implemented the algorithm and studied its performance. Although the complexity is cubic, its runtime is not bad because it’s cubic in term of the number of distinct data points, not the total number of data points.

Currently, the algorithm works for integer attributes only. As a future work, it would be interesting to see how it can be extended to real numbers. One way to tackle real numbers is to scale them to integers (as we did for Iris dataset for our experiments). In addition, the time complexity of the proposed algorithm reaches cubic order and may not suitable for very large datasets, which commonly occur in data mining. Thus, future research could propose improved algorithms to build hierarchies in very large databases.

7. REFERENCES


