Efficient Construction of Regression Trees with Range and Region Splitting [*]

Yasuhiko Morimoto
IBM Tokyo Research Laboratory
morimoto@trl.ibm.co.jp

Hiromu Ishii
University of Tokyo
hiromu@ims.u-tokyo.ac.jp

Shinichi Morishita
IBM Tokyo Research Laboratory
morisita@trl.ibm.co.jp

Abstract

We propose an efficient way of constructing regression trees in order to predict the objective numeric attribute values of given tuples. A regression tree is a rooted binary tree such that each internal node contains a test, which can be expressed as an RDB query, for splitting tuples into two disjoint classes and passing data in each class down to the left or right subtree. The mean of the objective attribute values at the leaf is used as the predicted value of the tuple.

To test a numeric attribute, traditional approaches use a guillotine-cut splitting that classifies data into those below a given value and others. Instead, we consider a family $\mathcal{R}$ of grid-regions in the plane associated with two given numeric attributes. We propose to use a test that splits data into those that lie inside a region R and those that lie outside.

The contributions of this paper are as follows. We present an efficient algorithm for computing $R \in \mathcal{R}$ that minimizes the mean squared error after the introduction of the test with the region R. Experiments confirmed that the use of region splitting gives a smaller mean squared error of regression trees. Our approach can also generate smaller regression trees.

[*]This research is partially supported by the Advanced Software Enrichment Project of the Information-Technology Promotion Agency.

**Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997**

1 Introduction

In recent years data mining, knowledge discovery from large databases, has attracted considerable attention in the database community, because many organizations have a strong interest in discovering unexpected and valuable rules in their huge databases. Efficient construction of association rules [AIS93, AS94, HF95, PS91] and decision trees [Qui86, Qui93] has been widely studied in the database literature, because data mining in large databases is such a challenging database query optimization problem. Another important reason is that association rules and tests used in decision trees can be directly expressed as standard relational database (RDB) queries, and are therefore efficiently executable by RDB systems. Thus automatic discovery of such rules and tests is highly promising as a means of enhancing the power of the decision support aspect of relational database systems.

Data Mining for Predicting Categorical Values

We treat one attribute as special, and call it the *objective attribute*. The other attributes are called *conditional attributes*. Our goal is to find some conditional attribute tests for characterizing the properties of the objective attribute.

Let X denote an attribute whose domain of data is categorical, that is, unordered discrete. We select X as the objective attribute. Let C denote a test on conditional attributes. An association rule has the form $C \Rightarrow (X = a)$, which means that if a tuple t satisfies C, A's value of t is equal to a with high probability. Some efficient ways of computing association rules have been proposed [AS94, PCY95], and some extended association rules with ranges and regions in the condition C have been studied [FMMT96a, FMMT96b, SA96]. A set of association rules of the form $C \Rightarrow (X = x)$ is useful for predicting the objective attribute value of X for given data.

A decision tree is a rooted binary tree structure for predicting the categorical values of the objective attribute for all data. Each internal node has a test on conditional attributes that splits data into two classes. A tuple is recursively tested at internal nodes and eventually reaches a leaf node. A good decision tree has the property that almost all the tuples arriving at every leaf node take a single value of the objective categorical attribute with a high probability, and therefore that value could be a good predictor of the objective attribute. Decision trees have been studied in the AI community for many years [PSF91], and recently efficient construction of decision trees from large databases has also been investigated in the database community [MAR96, FMMT96c].

Association rules and decision trees could be used for predicting the value of the objective numeric attribute, if we treat the attribute as a categorical one by sub-dividing its range into smaller sub-ranges and using those sub-ranges as unordered discrete items. This modification, however, does not directly handle the objective numeric attribute, and hence it is hard to guarantee high prediction accuracy.

Regression Tree for Predicting Ordered Numeric Values

In this paper, as a tool for predicting ordered numeric values, we will study regression trees, which are similar to decision trees in the sense that tests at internal nodes of regression trees are also expressible as standard RDB queries. In what follows, we therefore assume that the objective attribute is numeric. The mean of the objective attribute values at the leaf is used as the predicted value of the tuple.

We evaluate the quality of a regression tree by the mean squared error; that is, the average of the squared difference between the actual value and the predicted value of each tuple. Our goal is to construct a regression tree with a small mean squared error. The most common way of achieving this has been to greedily generate a new internal node which has a test that minimizes the mean squared error if the node is added to the current tree.

Breiman, Friedman, Olshen, and Stone [BFOS84] intensively investigated the usefulness of regression trees by applying them to many practical problems such as air pollution and criminal justice. According to their experiments [BFOS84], the accuracy of regression trees has been generally competitive with that of linear regression (e.g., the line estimator method). To be more precise, regression trees may be much more accurate in non-linear problems but tend to be less accurate in problems with good linear structure. This implies that tree-structured regression presents an interesting alternative for investigating regression-type problems. Furthermore, in practice, looking at real problems from different viewpoints allows us to make better predictions.

In this paper, therefore, we work on the improvement of Breiman et al.'s method for creating regression trees. Breiman et al.'s approach uses a test with a cut value that classifies data into those below the cut value and others, to minimize the mean squared error. Such a test is called a "guillotine-cut."

In practice we are often faced with various non-linear relationships. For instance, consider the case when two conditional numeric attributes have a strong correlation with respect to the objective numeric attribute. In this case, a test with a region for splitting data into two classes is natural and therefore much more effective as a means of reducing the mean squared error of the objective values than many guillotine-cut tests.

Y	M	W	BPS	GDM	...	GOLD	SP500
85	12	52	1.44353	0.40746	...	326.00	210.88
86	1	1	1.44612	0.40805	...	339.45	205.96
86	1	2	1.43794	0.40485	...	357.25	208.43
86	1	3	1.40470	0.40879	...	355.25	206.43
⋮	⋮	⋮	⋮	⋮	...	⋮	⋮
93	5	18	1.56875	0.63384	...	357.50	442.31

Table 1: Currency Data

Motivating Example

Table 1 shows a part of the relation that was collected from the New York currency markets every Monday from the last Monday in 1985 through the first Monday of May 1993. It contains 10 numeric attributes that are W (week), M (month), Y (year), BPS (British pound sterling, i.e., US$/pound), GDM (deutschmark, i.e., US$/mark), YEN (Japanese yen, i.e., US$/yen), TB3M (3-month Treasury bill yields), TB30Y (30-years Treasury bill yields), GOLD (US$/ounce), and SP500 (Standard and Poors index). Suppose we are interested in SP500, which is one of the indices for measuring the overall market performance on the New York Stock Exchange. To predict the value of SP500 from the value of other attributes, let us construct a regression tree.

Figure 1 illustrates a node in the regression tree generated by our method. The top histogram shows the data distribution of 384 records in Table 1 along with the values of SP500, which range from 190 to 460 and are divided into subranges with a uniform width of 10. The data in the top histogram are then split by the region R in the middle of Figure 1. R is a region in the Euclidean plane whose x-axis is the GDM (mark) price and whose y-axis is the GOLD price. The bottom-left histogram and bottom-right histogram respectively show the data distribution of records inside the region R and that of records outside R. The number of tuples, and the average and variance of SP500 values in those three histograms are:

	No. of Tuples	Average	Variance
Top	384	324	4430
Bottom-left	155	391	1117
Bottom-right	229	278	1535

The region R is chosen to minimize the sum of the mean squared errors of the two subsets after the split. From the shape of R, we can see that a lower gold price (lower than 368) is generally related to a higher SP500 index. Also, when the value of GDM is between 0.56 and 0.60, the SP500 index is likely to be high if gold is not too high. Although we only present a node in a regression tree here, in general we recursively generate internal nodes until we cannot significantly improve the mean squared error of the tree.

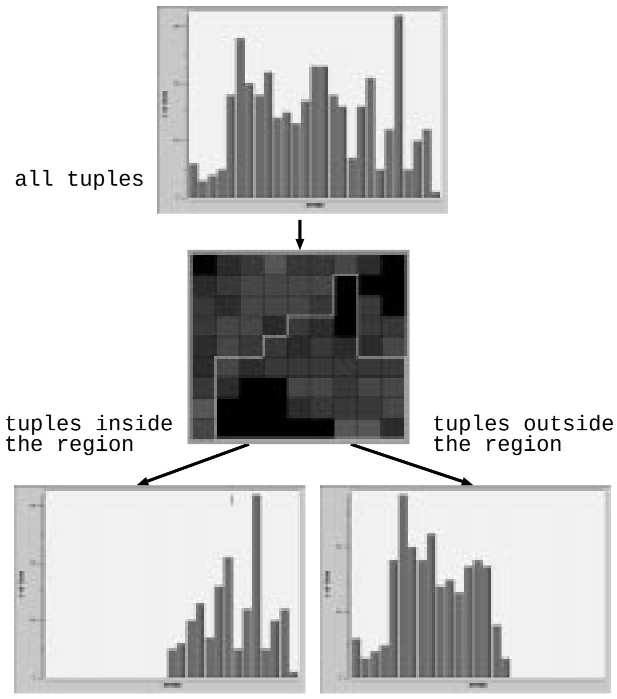

Figure 1: Region Splitting

Main Results

To confirm the effectiveness of splitting by regions, we provide positive answers to the following questions:

1. Can we efficiently compute the *optimal* region that minimizes the mean squared error after the addition of the test with the region?
2. In practice, does region-splitting give a more accurate regression tree than a guillotine-cut?
3. Is it easy to comprehend substantial non-linear relationships among attributes?

With regard to the first question, we begin by defining "regions," which are expressed as RDB queries. Let n be the total number of tuples in the database. In practice, n ranges from hundreds to millions. For each numeric attribute, we generate an equi-depth bucketing so that tuples are uniformly distributed into N ordered buckets. Next, for each pair of numeric attributes, we create an $N \times N$ pixel grid G according to the Cartesian product of the bucketing of each numeric attribute. A *grid region* is a union of pixels of G that are connected, and it is *x-monotone* if its intersection with each column of G is undivided. For instance, the grid region in the middle of Figure 1 is x-monotone. We can express an x-monotone region by a disjunction of expressions of the form `(a1 < x < a2) and (b1 < y < b2)` that shows the intersection of the region with a column. An x-monotone region is *rectilinear* if its intersection with each row of G is also undivided.

We consider the class of x-monotone regions, the class of rectilinear regions, and the class of rectangular regions. For each class, we present an algorithm for computing the optimal region that minimizes the mean squared error. The expected running time complexities of those three algorithms for x-monotone regions, rectilinear regions, and rectangular regions are $O(N^3 \log N)$, $O(N^4 \log N)$, $O(N^3 \log N)$, respectively. If we use $N \leq n^{1/3}$ buckets, the expected time complexities of the algorithms are $O(n \log n)$, $O(n^{4/3} \log n)$, and $O(n \log n)$, respectively.

With respect to the second question, we performed N-fold cross-validation analysis; that is, we randomly divided the given database into N equal-sized subsets, built a regression tree from the union of $N-1$ subsets, and evaluated the mean squared error of the regression tree against the remaining one subset of data. Tests showed that the mean squared error of regression trees with region splitting is smaller than that of conventional regression trees with guillotine-cut splitting. Our approach can also generate compact regression trees.

Figure 1 shows the answer to our last question. Without the region rules it is hard to grasp the substantial non-linear correlation, which affects the SP500 index.

2 Regression Tree

Tests on Conditional Attributes

Let $\mathcal{D}$ denote a relation scheme, which is a set of categorical or numeric attributes. Let t be a tuple over $\mathcal{D}$, and let $t[A]$ denote the value for attribute A. We select a numeric attribute A as special and call it the *objective* attribute. We call the other attributes in $\mathcal{D}$ *conditional attributes.*

For a categorical conditional attribute B, a *primitive* test has the form $B = b$, where b is an element in the domain of B. A tuple t satisfies a test of the form $B = b$ if $t[B]$ is equal to b. A test on a categorical conditional attribute is a Boolean combination of the primitive tests. For numeric conditional attributes B and C, we use primitive tests such as $B < b$, $B \in [b_1, b_2]$ and $(B, C) \in R$, where b, b_1, and b_2 are elements in the domain of B, and R is a connected region in the Euclidean plane associated with B and C. The value of t satisfies $B \in [b_1, b_2]$ if $b_1 \leq t[B] \leq b_2$, and satisfies $(B, C) \in R$ if $(t[B], t[C])$ is mapped to the region R.

Regression Trees

A *regression tree* is a rooted binary tree such that each internal node is associated with a test. At an internal node (initially the root node), we check whether a given tuple meets the test of the node. If the tuple satisfies the test, it goes down to the left subtree. Otherwise it goes to the right subtree. A tuple is recursively checked at internal nodes and finally reaches a leaf node. Note that any tuple t reaches a unique leaf node, which we denote by $leaf(t)$. Then we say that t *belongs to* the leaf node $leaf(t)$.

We generate a regression tree T from a specific instance of relation D over $\mathcal{D}$. D is called a *training* data. Let w denote a node in a regression tree T, and let D_w denote the set of tuples in D that reach the node w. In what follows, we assume that D_w is not empty. Let $|D_w|$ denote the number of tuples in D_w, and let $\mu(D_w)$ denote the average of A's values in D_w; that is, $\mu(D_w) = (1/|D_w|)\sum_{t \in D_w} t[A]$.

Mean Squared Error

Suppose that we are given another instance of the relation D' over $\mathcal{D}$. For each tuple t in D', we find the leaf node $leaf(t)$, and we use $\mu(D_{leaf(t)})$ as a predictor of $t[A]$. In order to evaluate the prediction error of T against relation D', we use the *mean squared error*, the average of the squared difference between the actual objective attribute value and the prediction for all the tuples; that is,

$$\frac{\sum_{t\in D'}(t[A]-\mu(D_{leaf(t)}))^2}{|D'|},$$

which we will denote by $MSE(T, D')$.

When we are given a training relation D, we do not know what instances of $\mathcal{D}$ will be given as test data. Thus we try to generate a regression tree T such that the mean squared error of T against the training relation D, namely $MSE(T, D)$, is small. To construct such a tree T, Breiman et al. propose a greedy method that selects a leaf of T arbitrarily and generates further leaves from it by applying the test that minimizes the mean squared error of T. We start with a tree that has only a root, and recursively apply the above generation until we cannot significantly improve the mean squared error of T against D.

Guillotine Cut Splitting on Numeric Attributes

For a numeric attribute B, Breiman et al. consider only a test of the form $B < b$, which is called a *guillotine-cut splitting*. Suppose that we select a leaf w to partition. Among all the instances of guillotine cut splitting, Breiman et al.'s method selects the optimal one — say $B < b$ — that minimizes the mean squared error. $B < b$ partitions D_w into $D_w^{low} = \{t \in D_w \mid t[B] < b\}$ and $D_w^{high} = \{t \in D_w \mid b \le t[B]\}$, and it minimizes

$$\frac{\sum_{t\in D_w^{low}}(t[A]-\mu(D_w^{low}))^2+\sum_{t\in D_w^{high}}(t[A]-\mu(D_w^{high}))^2}{|D_w|}.$$

3 Range and Region Splitting

In this paper, we propose to consider a broader class of tests such as a range splitting of the form $B \in [b_1, b_2]$ and a region splitting of the form $(B, C) \in R$. Since ranges can be treated as special cases of regions, we focus on computing the region that minimizes the mean squared error. Suppose that we choose a leaf w to divide. Define

$$D_w^{in} = \{t \in D_w \mid (t[B], t[C]) \in R\}, \text{ and } D_w^{out} = D_w - D_w^{in}.$$

Following Breiman et al.'s greedy method, we will consider how to find the region R that minimizes

$$\frac{\sum_{t\in D_w^{in}}(t[A]-\mu(D_w^{in}))^2+\sum_{t\in D_w^{out}}(t[A]-\mu(D_w^{out}))^2}{|D_w|},$$

which we will denote by $U(R)$.

Grid Regions

We present three classes of regions that can be directly expressed as RDB queries. Later in this section, for each class, we present an efficient algorithm for computing the optimal region of the class that minimizes the mean squared error. To define those regions, we first distribute the values of B (resp. C) into N_B (N_C) equal-sized buckets. We then divide the Euclidean plane associated with B and C into $N_B \times N_C$ pixels (unit squares). For simplicity, we assume that $N_B = N_C = N$ without loss of generality as regards our algorithms. A *grid region* is a set of pixels. An *x-monotone* region is a grid region whose intersection with any vertical line is undivided. A *rectilinear* region is an x-monotone region such that its intersection with any horizontal line is also undivided. We will consider the class of x-monotone regions, the class of rectilinear regions, and the class of rectangular (grid) regions.

Interclass Variance

Let $\mathcal{R}$ be a class of grid regions. For easier computation of the region $R \in \mathcal{R}$ that minimizes $U(R)$, we introduce

$$V(R) = |D_w^{in}|(\mu(D_w^{in})-\mu(D_w))^2+|D_w^{out}|(\mu(D_w^{out})-\mu(D_w))^2,$$

which we call the *interclass* variance. The following lemma shows that the region R that maximizes $V(R)$ also minimizes $U(R)$.

Lemma 1 *Let $\mathcal{R}$ be a class of grid regions, and let R be an element in $\mathcal{R}$. The maximization of $V(R)$ is equivalent to the minimization of $U(R)$.*

Proof:

$$\begin{aligned} & V(R) \\ = \; & |D_w^{in}|(\mu(D_w^{in})-\mu(D_w))^2+|D_w^{out}|(\mu(D_w^{out})-\mu(D_w))^2 \\ = \; & -|D_w|\mu(D_w)^2+(|D_w^{in}|\mu(D_w^{in})^2+|D_w^{out}|\mu(D_w^{out})^2) \end{aligned}$$

Since $-|D_w|\mu(D_w)^2$ is invariant with respect to the choice of R, the maximization of $V(R)$ is equivalent to the maximization of $(|D_w^{in}|\mu(D_w^{in})^2+|D_w^{out}|\mu(D_w^{out})^2)$.

$$\begin{aligned} & U(R) \\ = \; & \frac{\sum_{t\in D_w^{in}}(t[A]-\mu(D_w^{in}))^2+\sum_{t\in D_w^{out}}(t[A]-\mu(D_w^{out}))^2}{|D_w|} \\ = \; & \frac{\sum_{t\in D_w} t[A]^2-(|D_w^{in}|\mu(D_w^{in})^2+|D_w^{out}|\mu(D_w^{out})^2)}{|D_w|} \end{aligned}$$

Since $\sum_{t\in D_w} t[A]^2$ and $|D_w|$ are independent of the choice of R, the minimization of $U(R)$ is equivalent to the maximization of $(|D_w^{in}|\mu(D_w^{in})^2+|D_w^{out}|\mu(D_w^{out})^2)$, and therefore the maximization of $V(R)$ is equivalent to the minimization of $U(R)$. $\square$

Region Maximizing Interclass Variance

Next we consider how to compute the region $R \in \mathcal{R}$ that maximizes $V(R)$. Observe that $V(R)$ is invariant if we replace $t[A]$ by $t[A]-\mu(D_w)$ for each $t \in D_w$. Thus, after this replacement, the region maximizing $V(R)$ still gives the solution to the original problem. Furthermore, this modification makes $\mu(D_w) = 0$, and hence

$$V(R) = |D_w^{in}|\mu(D_w^{in})^2 + |D_w^{out}|\mu(D_w^{out})^2.$$

Let x denote $|D_w^{in}|$, let y be $\sum_{t \in D_w^{in}} t[A]$, and let M be $|D_w|$. Since $|D_w^{in}|\mu(D_w^{in}) + |D_w^{out}|\mu(D_w^{out}) = |D_w|\mu(D_w) = 0$, we have

$$V(R) = y^2(\frac{1}{x} + \frac{1}{M-x}),$$

which we will denote by $f(x,y)$.

For each $R \in \mathcal{R}$, we associate a *stamp point* (x,y), where $x = |D_w^{in}|$ and $y = \sum_{t \in D_w^{in}} t[A]$. Consider the convex hull of all those stamp points. Since the number of x-monotone regions (or rectilinear regions) is more than N^N, we cannot afford to calculate all the stamp points, and therefore we simply assume their existence. The following theorem guarantees the existence of the stamp point associated with the region $R \in \mathcal{R}$ that maximizes $V(R)$ on the boundary of the convex hull of all stamp points associated with regions in $\mathcal{R}$.

Theorem 1 *$f(x,y) = y^2(\frac{1}{x} + \frac{1}{M-x})$ is convex in the region $M > x > 0$; namely,*

$$\frac{f(x_1,y_1) + f(x_2,y_2)}{2} \geq f(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}),$$

for arbitrary points (x_1,y_1) and (x_2,y_2) such that $M > x_1, x_2 > 0$.

Proof: Let Δ denote a vector (δ_1, δ_2), and let V be $\delta_1 x + \delta_2 y$. It is sufficient to show that for any Δ, $\partial^2 f(x,y)/\partial V^2 \geq 0$. First let us consider the case in which $\delta_1, \delta_2 \neq 0$.

$$\begin{aligned} & \frac{\partial f(x,y)}{\partial V} \\ = & \frac{\partial f(x,y)}{\partial x} \cdot \frac{1}{\delta_1} + \frac{\partial f(x,y)}{\partial y} \cdot \frac{1}{\delta_2} \\ = & y^2(\frac{-1}{x^2} + \frac{1}{(M-x)^2})\frac{1}{\delta_1} + 2y(\frac{1}{x} + \frac{1}{M-x})\frac{1}{\delta_2} \end{aligned}$$

Next,

$$\begin{aligned} & \frac{\partial^2 f(x,y)}{\partial V^2} \\ = & \{y^2(\frac{2}{x^3} + \frac{2}{(M-x)^3})\frac{1}{\delta_1} + 2y(\frac{-1}{x^2} + \frac{1}{(M-x)^2})\frac{1}{\delta_2}\}\frac{1}{\delta_1} \\ & +\{2y(\frac{-1}{x^2} + \frac{1}{(M-x)^2})\frac{1}{\delta_1} + 2(\frac{1}{x} + \frac{1}{(M-x)})\frac{1}{\delta_2}\}\frac{1}{\delta_2} \\ = & \frac{2}{x}(\frac{y}{x\delta_1} - \frac{1}{\delta_2})^2 + \frac{2}{M-x}(\frac{y}{(M-x)\delta_1} + \frac{1}{\delta_2})^2 \\ \geq & 0 \end{aligned}$$

Next, when $\delta_1 \neq 0$ and $\delta_2 = 0$,

$$\frac{\partial^2 f(x,y)}{\partial V^2} = y^2(\frac{2}{x^3} + \frac{2}{(M-x)^3})\frac{1}{{\delta_1}^2} \geq 0.$$

We can similarly prove the case in which $\delta_1 = 0$ and $\delta_2 \neq 0$.
□

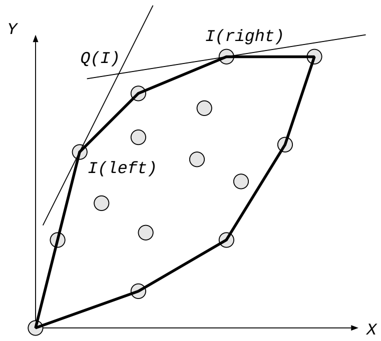

Figure 2: Guided Branch-and-Bound Search

Hand-Probing for Scanning the Convex Hull

To search for the stamp point associated with the region R that maximizes $V(R)$, thanks to Theorem 1, we can focus on scanning the convex hull of all the stamp points. To this end we first present a way of touching a point on the convex hull, using the "hand-probing" technique, invented by Asano, Chen, Katoh, and Tokuyama [ACKT96] for image segmentation and modified by Fukuda, Morimoto, Morishita, and Tokuyama [FMMT96b] for extraction of the optimized association rules. Hand probing is based on the "touching oracle"; that is, we select a slope θ and compute the tangent point at which a line with slope θ is tangent to the convex hull. When $\mathcal{R}$ is the set of x-monotone regions, Fukuda et al. [FMMT96b] provide an $O(N^2)$-time algorithm for generating the region corresponding to the tangent point. They also provide an $O(N^3)$-time algorithm when $\mathcal{R}$ is the set of rectangular regions. For the case in which $\mathcal{R}$ is the set of rectilinear regions, Yoda et al. [YFM$^+$97] present an $O(N^3)$-time algorithm.

By using the hand-probing technique repeatedly, we can scan all the points on the hull and find the optimal point associated with the region that maximizes $V(R)$. However, the number of points could be more than N^N if we consider x-monotone regions or rectilinear regions, and therefore in the worst case we may need to try N^N touching oracles in order to find the optimal stamp point, which is intractable in practice. To avoid searching for unnecessary points, we now introduce a guided branch-and-bound strategy.

Guided Branch-and-Bound Search

While searching for the optimal region, we maintain the current maximum V_{max} corresponding to the points examined so far. Suppose we have examined two tangent points, say $I(left)$ and $I(right)$, and consider the interval $I = [I(left), I(right)]$ in Figure 2. Let $Q(I) = (x_{Q(I)}, y_{Q(I)})$ be the point of intersection of the two tangent lines that are used to compute $I(left)$ and $I(right)$. We compute the interclass variance value of $Q(I)$, $f(Q(I)) = f(x_{Q(I)}, y_{Q(I)})$.

Corollary 1 *For any point $Q' = (x', y')$ inside the triangle $I(left)$, $I(right)$, and $Q(I)$,*

$$f(x', y') \leq max\{f(x_{Q(I)}, y_{Q(I)}), V_{max}\}$$

Proof: Immediate from Theorem 1. □

This corollary gives an upper bound between $I(left)$ and $I(rignt)$. Hence, we can find the optimal region effectively by the hand probing together with a branch and bound strategy guided by the values $f(Q(I))$. We examine the subinterval with the maximum value of $f(Q(I))$ first. In addition, subintervals whose $f(Q(I))$ is less than V_{max} are pruned away. During the process, V_{max} is monotonically increased while each $f(Q(I))$ is monotonically decreased. Most of subintervals are expected to be pruned away during the computation, and therefore the number of touching oracles is expected to be $O(\log n)$, where n is the number of points on the convex hull.

Since both of the number of x-monotone regions and the number of rectilinear regions are bounded by N^{2N}, the expected number of touching oracles is $O(N \log N)$. On the other hand, the number of rectangular regions is bounded by N^4, and hence the number of touching oracles in this case is expected to be $O(\log N)$. Recall that the time complexities of performing one touching oracle for x-monotone regions, rectilinear regions, and rectangular regions are $O(N^2)$, $O(N^3)$, and $O(N^3)$, respectively. Thus we can experimentally compute the region minimizing the mean squared error in time proportional to $O(N^3 \log N)$, $O(N^4 \log N)$, and $O(N^3 \log N)$ for x-monotone regions, rectilinear regions, and rectangular regions, respectively.

4 Experimental Results

4.1 Prediction Accuracy

Ten fold Cross-Validation Test

We performed the following ten-fold cross-validation test:

- First, randomly divide the original dataset into ten subsets of almost equal sizes.
- Take the union of nine subsets and use it as the training dataset to generate a regression tree that splits data by guillotine cuts, x-monotone regions, rectilinear regions, or rectangular regions.
- Then, use the remaining subset as the test dataset to evaluate the regression tree, and compute the mean squared error *against* the test dataset. The mean squared error varies depending on the sample dataset. In order to compare the mean squared errors of different datasets, in what follows, we use a "normalized" version called the *relative mean squared error* of a regression tree, which is defined as the mean squared error of the tree divided by the variance of the test data.
- Repeat the above steps ten times, and then calculate the average of all the relative mean squared errors.

How to Avoid Overfitting

From a training dataset, we can generate large regression trees by expanding leaves as much as we want. Larger regression trees can reduce the mean squared error in the training dataset but are likely to *overfit* the training dataset, giving a higher mean squared error against the test dataset. To avoid such an overfitting, we need some criteria for when to stop expanding a regression tree.

Let A denote the objective numeric attribute. Suppose that the set of tuples D_w at same node, say w, is divided into D_w^{in} and D_w^{out} by splitting it with region R. In Section 3, we present an algorithm for computing the optimal region R that minimizes the following mean squared error after the partition by R:

$$\frac{\sum_{t \in D_w^{in}} (t[A] - \mu(D_w^{in}))^2 + \sum_{t \in D_w^{out}} (t[A] - \mu(D_w^{out}))^2}{|D_w|} \cdots (*)$$

Before the partition by R, the mean squared error is

$$\frac{\sum_{t \in D_w} (t[A] - \mu(D_w))^2}{|D_w|} \cdots (**)$$

The splitting by R is effective if the difference of the above two mean squared errors, $(**) - (*)$, is large, because the splitting contributes to the reduction of the mean squared error of the regression tree. Let us call the difference the *profit* of the splitting with region R. If the profit is small, the partition by R is not effective for expanding the node and should therefore be avoided. To this end, we provide a threshold for the profit of a region splitting, and we do not expand the node w if we cannot find any region splitting whose profit is less than the threshold. Actually we used the following threshold:

$$\alpha \times \frac{\sum_{t \in D} (t[A] - \mu(D))^2}{|D|},$$

where D denotes the set of all the tuples in the training dataset, and α is a special parameter called a *pruning parameter*. This threshold is equivalent to the AID criteria mentioned in [BFOS84]. The choice of pruning parameter α strongly affects the mean squared error against the test data. A larger pruning parameter is more likely to prune subtrees and hence to produce smaller regression trees, while a smaller pruning parameter generates a larger regression tree that might overfit the training data. We need to control the pruning parameter nicely to yield a better regression tree with a smaller mean squared error. We will discuss this issue later in more detail.

Pixel Density

The average number of tuples in each pixel, which we call the *pixel density*, tends to affect the mean squared error. Using a lower pixel density is likely to make regression trees overfit the training dataset, while a coarse grid with a higher pixel density often fails to find various-shaped regions. We empirically found that a pixel density ranging from 5 to 10 gives a lower mean squared error for test datasets, and we therefore use a pixel density of 5 or 10.

In the process of generating regression trees, the number of tuples becomes smaller at nodes lower in the tree. In order to guarantee a pixel density of 5 or 10, we are forced to use a grid of, say 2×2, which is too coarse to generate interesting regions. If we have to use a coarse grid of less than 5×5, we employ one-dimensional (range) splitting instead.

Dataset	#tuples	#attrs
add10	9792	10
abalone	4177	8
kin-8fh	8192	8
kin-8fm	8192	8
kin-8nh	8192	8
kin-8nm	8192	8
pumadyn-8fh	8192	8
pumadyn-8fm	8192	8
pumadyn-8nh	8192	8
pumadyn-8nm	8192	8

Table 2: Dataset Summary

Test Datasets

We used several public datasets, summarized in Table 2, which were acquired from the following WWW site:

`http://www.cs.utoronto.ca/~ delve/data/datasets.html.`

We chose these datasets because almost all the attributes are numeric. As regions for splitting the datasets, we used x-monotone regions, rectilinear regions, and rectangular regions, and we assigned a value of 5 or 10 for the pixel density.

To create a better regression tree with a smaller mean squared error, we generate a number of regressions for various values of the pruning parameter. For instance, let us consider the "add10" dataset. Figure 3 shows how the relative mean squared error changes according to the value of the pruning parameter. The graph for "Rectilinear (dens5)" shows the relative mean squared errors of regression trees with rectilinear regions for a pixel density of 5, and we see that the relative mean squared error is smallest when the pruning parameter is 0.0002. Observe that the value increases for pruning parameters less than 0.0002, which indicates that the regression tree becomes larger and overfits the training data. Similarly, for x-monotone region splitting, rectangular region splitting, and guillotine cut splitting, we can find the respective regression tree with the smallest relative mean squared error. Observe that the use of rectilinear regions is the most effective.

For the other datasets, we performed the same analysis (See Figures 4-12). Table 3 summarizes the results, showing the pair of the smallest relative mean squared error and the average number of leaves for each splitting. In the table, we underline the smallest relative mean squared error for four types of splitting. Note that rectilinear region splitting with a pixel density of 5 is always better than the others, and tends to yield smaller regression trees.

4.2 Performance Results

The performance in constructing regression trees depends on the execution time needed to compute the optimal regions for splitting data. Thus the cost of computing one optimal region gives us an idea of the overall performance in generating one regression tree.

Computing One Optimal Region

We generated our test data, in the form of an $N \times N$ grid, as follows: We first made random numbers uniformly dis-

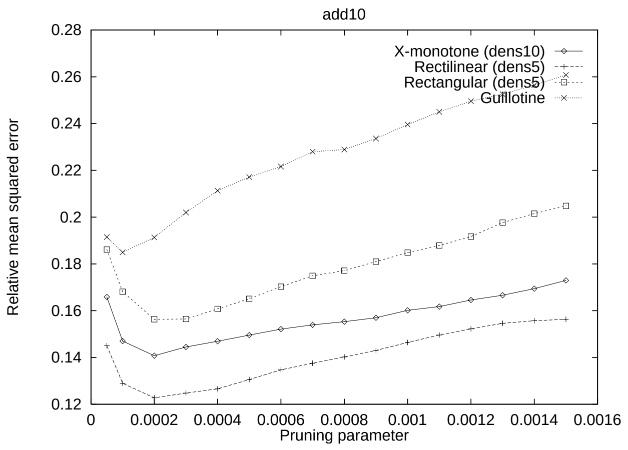

Figure 3: Accuracy Results for "add10"

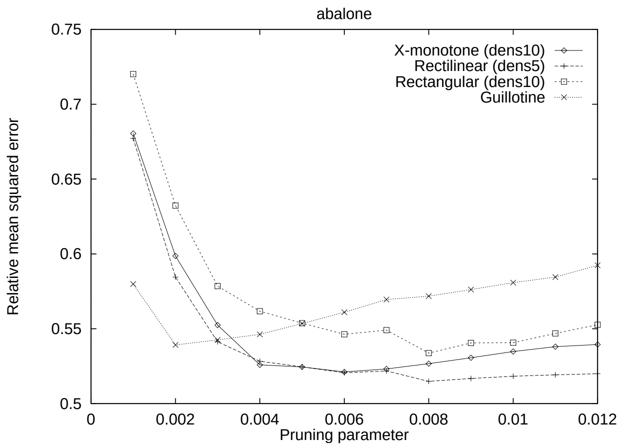

Figure 4: Accuracy Results for "abalone"

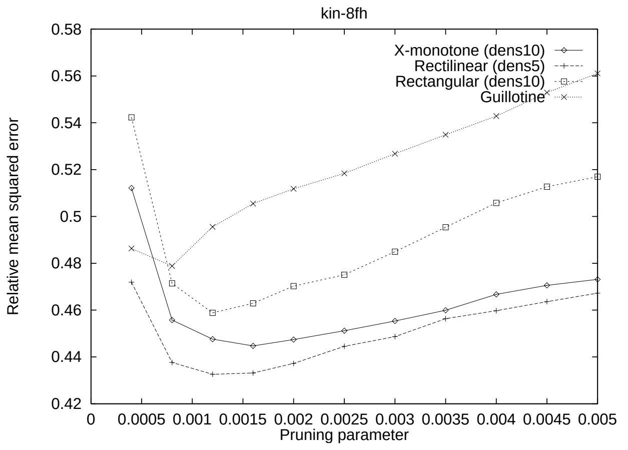

Figure 5: Accuracy Results for "kin-8fh"

	X-monotone		Rectilinear		Rectangular		Guillotine	
Dataset	Err	Size	Err	Size	Err	Size	Err	Size
add10	1.41e-1	132.4	1.23e-1	120.9	1.56e-1	222.8	1.85e-1	540.2
abalone	5.21e-1	10.0	5.15e-1	8.7	5.34e-1	10.2	5.39e-1	38.9
kin-8fh	4.47e-1	34.1	4.33e-1	59.7	4.59e-1	69.9	4.79e-1	128.3
kin-8fm	2.25e-1	110.3	1.97e-1	138.3	2.57e-1	162.7	2.49e-1	919.9
kin-8nh	6.49e-1	29.9	6.18e-1	25.8	6.19e-1	37.7	6.55e-1	88.4
kin-8nm	4.94e-1	44.9	4.49e-1	92.2	4.78e-1	59.3	5.41e-1	203.6
pumadyn-8fh	4.12e-1	8.0	4.02e-1	8.0	4.09e-1	19.9	4.10e-1	26.2
pumadyn-8fm	6.04e-2	15.7	5.95e-2	23.3	6.53e-2	51.4	6.32e-2	153.9
pumadyn-8nh	3.47e-1	8.0	3.37e-1	8.0	3.53e-1	26.7	3.55e-1	44.1
pumadyn-8nm	5.30e-2	28.0	4.96e-2	38.2	5.50e-2	100.1	5.35e-2	185.0

Table 3: Summary of Cross-Validation Results

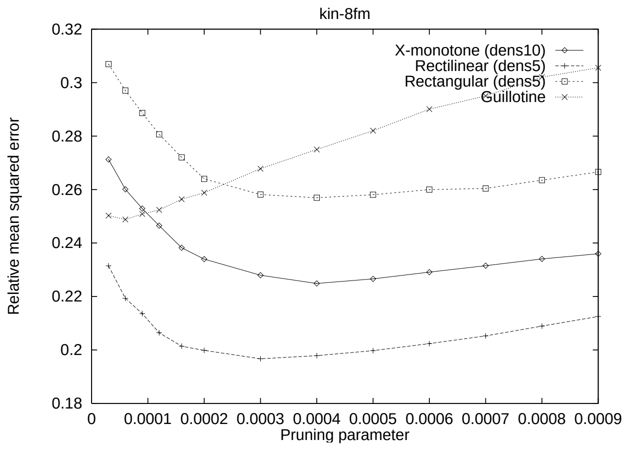

Figure 6: Accuracy Results for “kin-8fm”

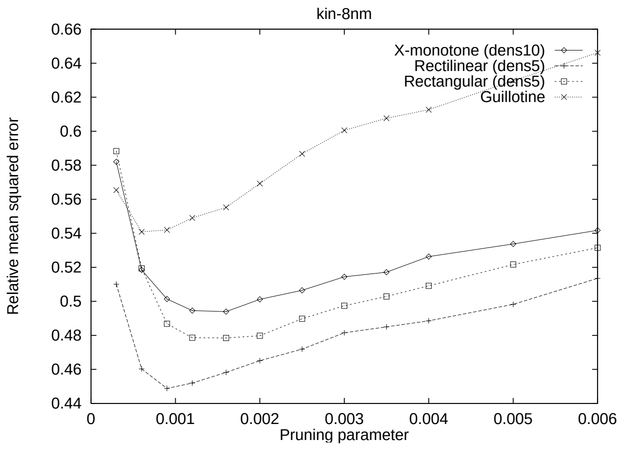

Figure 8: Accuracy Results for “kin-8nm”

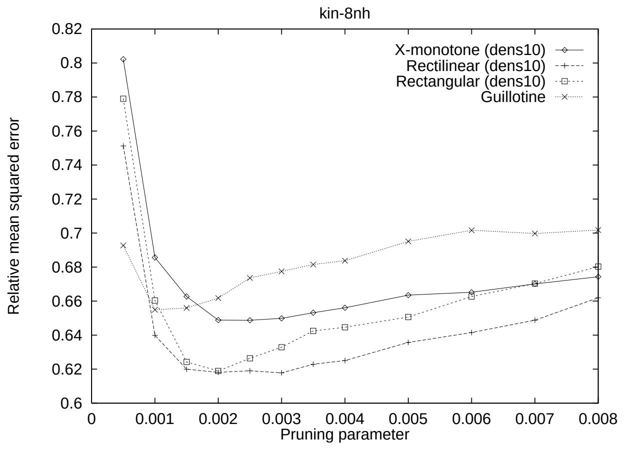

Figure 7: Accuracy Results for “kin-8nh”

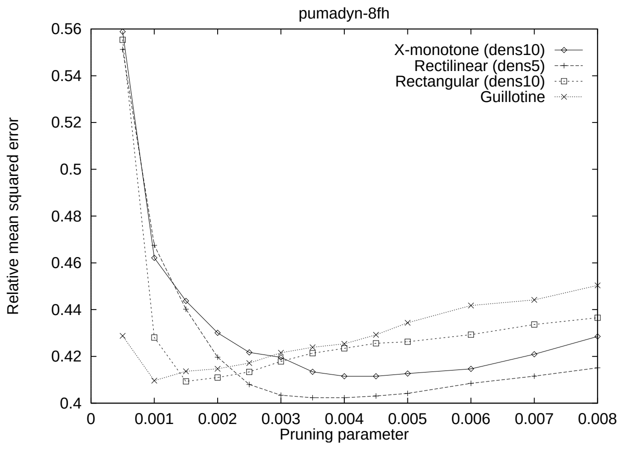

Figure 9: Accuracy Results for “pumadyn-8fh”

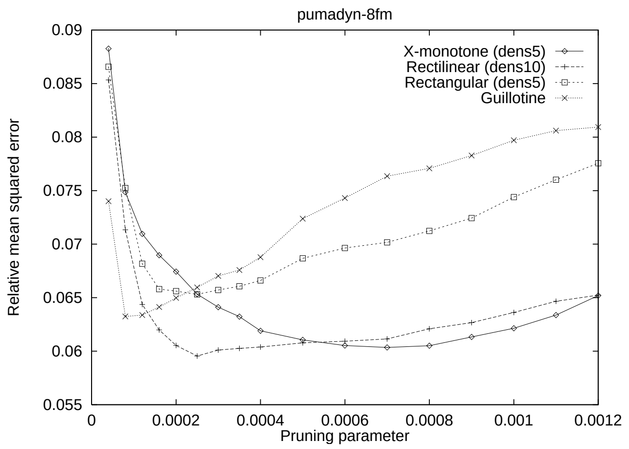

Figure 10: Accuracy Results for "pumadyn-8fm"

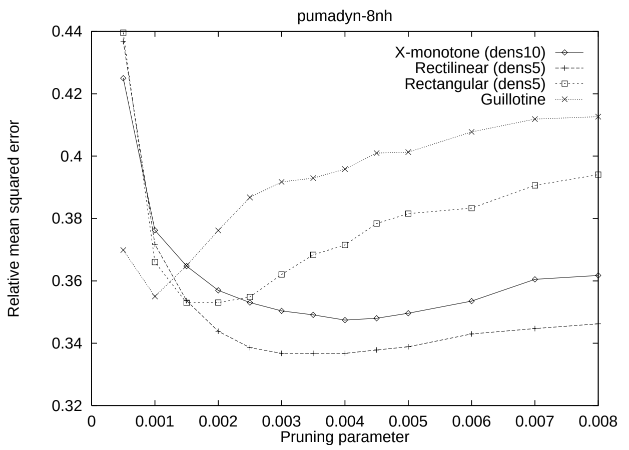

Figure 11: Accuracy Results for "pumadyn-8nh"

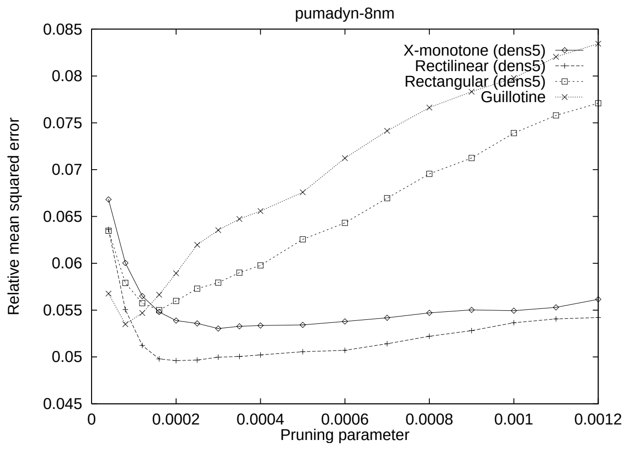

Figure 12: Accuracy Results for "pumadyn-8nm"

	X-monotone		Rectilinear		Rectangular	
#pixel	time	#t	time	#t	time	#t
10^2	0.08	26	0.04	21	0.01	18
20^2	0.36	28	0.34	27	0.05	26
30^2	0.89	31	1.35	32	0.16	27
40^2	1.89	35	3.47	32	0.37	29
50^2	2.69	32	7.01	32	0.75	31

Table 4: Time for Computing an Optimal Region(1)

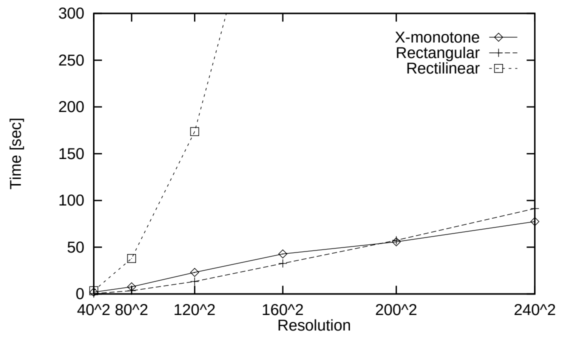

Figure 13: Time for Computing an Optimal Region(2)

tributed in $[N^2, 2N^2]$ and assigned them to the number of tuples in each pixel. We then assigned $1, \ldots, N^2$ to the sum of the target values in a pixel, from a corner pixel to the central one, proceeding in a spiral fashion. These test data were generated so that the number of points on the convex hull increased sub-linearly to N, the square root of the number of pixels. We examined the CPU time taken to compute the optimal regions and the number of touching oracles needed to find the regions. We performed all experiments on an IBM RS/6000 workstation with a 112 MHz PowerPC 604 chip and 512 MB of main memory, running under the AIX 4.1 operating system.

Table 4 shows the time (sec.) and number of touching oracles, denoted "#t", that were required in the guided branch-and-bound algorithm to find the optimal x-monotone (resp. rectilinear, or rectangular) region that minimizes the mean squared error. It shows that the number of touching oracles increases very slowly thanks to the guided branch-and-bound algorithm. Figure 13 confirms that the CPU time follows our scale estimation. Although the asymptotic time complexity for computing the optimal x-monotone region is better than that for computing the optimal rectilinear region, in practice the optimal rectilinear region is computed faster because the constant factor is smaller.

At the root of a regression tree, we may need a large number of pixels to guarantee the specified pixel density. The number of tuples, however, decreases in lower parts of the tree, and the number of pixels soon becomes less than 30×30 for most datasets; hence computing the optimal rectilinear region is not costly according to Table 4.

#tuples	X-monotone	Rectilinear
2000	849	341
4000	2172	945
6000	3819	1620
8000	5477	2304

Table 5: Tree Construction Time (1)

#attributes	X-monotone	Rectilinear
4	901	599
6	1897	1028
8	3416	1430
10	5477	2304

Table 6: Tree Construction Time (2)

Computing One Regression Tree

The next experiment examines the overall performance in tree construction. At each node of a regression tree, we first prepare the grid, and then compute the optimal region. The grid preparation is not expensive, because it can be done by scanning all the tuples at a node just once. The problem is that we have to calculate the optimal region for all pairs (permutations for x-monotone regions) of two distinct numeric attributes. Thus the number of attributes dramatically affects the overall performance.

Table 5 compares the time (sec.) taken to construct trees by using datasets with different numbers of tuples. We randomly selected tuples from the "add10" dataset to generate datasets having different numbers of tuples, and used those datasets to construct regression trees by performing region-splitting with a pixel density of 5. We set the pruning parameter α to 0.0002 because, as we have seen in Figure 3, the value yields the tree with the smallest mean squared error. The result shows that tree construction time is a little more than our scale estimation, because trees from larger datasets become bigger.

Table 6, on the other hand, compares the performance using datasets with different numbers of attributes. We used 8000 tuples from the "add10" dataset, and constructed trees using the first N numerical attributes. Observe that the time complexity is almost linear in the square of the number of attributes.

5 Discussion

Experiments using diverse datasets confirmed that region-splitting trees are more accurate than conventional ones. However, in order to use the region-splittings, we have to spend additional computation time which is proportional to the number of numeric conditional attributes. In the experiments, region-splitting trees reduced error ratio by around 10%. One case of which achieves dramatic 34% reduction. Those reductions show substantial capability of our method. If we compare the error ratio of compact trees which are small enough to be appreciated, the error reduction is much more significant as you can see in Figures 3 to 12. In many applications, the improvements will be worth the computational cost if there are not too many numeric attributes. Another important advantage is size of the trees. The size is much smaller than conventional ones. It makes easy to grasp rules which affect values of the target attribute. Furthermore, we can recognize many non-linear correlations among conditional attributes which could not be found without region-splittings.

References

[ACKT96] Tetsuo Asano, Danny Chen, Naoki Katoh, and Takeshi Tokuyama. Polynomial-time solutions to image segmentations. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 104–113, 1996.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arum Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 207–216, May 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pages 487–499, 1994.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[FMMT96a] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Mining optimized association rules for numeric attributes. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 182–191, June 1996.

[FMMT96b] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 13–23, June 1996.

[FMMT96c] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Constructing efficient decision trees by using optimized association rules. In *Proceedings of the 22nd VLDB Conference*, pages 146–155, 1996.

[HF95] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st VLDB Conference*, pages 420–431, 1995.

[MAR96] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In *Proceedings of the Fifth International Conference on Extending Database Technology*, 1996.

[PCY95] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 175–186, May 1995.

[PS91] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In *Knowledge Discovery in Databases*, pages 229–248, 1991.

[PSF91] G. Piatetsky-Shapiro and W. J. Frawley, editors. *Knowledge Discovery in Databases.* AAAI Press, 1991.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, June 1996.

[YFM+97] Kunikazu Yoda, Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Computing optimized rectilinear regions for association rules. In *Proceedings of KDD'97*, August 1997.