Machine Learning, 45, 235–259, 2001

Efficient Construction of Regression Trees with Range and Region Splitting*

YASUHIKO MORIMOTO morimoto@jp.ibm.com
IBM Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato, Kanagawa 242-8502, Japan

HIROMU ISHII† hiromu@kuis.kyoto-u.ac.jp
Kyoto University, Yoshida-Honmachi, Sakyo Ward, Kyoto 606-8501, Japan

SHINICHI MORISHITA†† moris@is.s.u-tokyo.ac.jp
University of Tokyo, 7-3-1 Hongo, Bunkyo Ward, Tokyo 113-0033, Japan

Editor: Douglas Fisher

Abstract. We propose a method for constructing regression trees with range and region splitting. We present an efficient algorithm for computing the optimal two-dimensional region that minimizes the mean squared error of an objective numeric attribute in a given database. As two-dimensional regions, we consider a class $\mathcal{R}$ of grid-regions, such as "x-monotone," "rectilinear-convex," and "rectangular," in the plane associated with two numeric attributes. We compute the optimal region $R \in \mathcal{R}$. We propose to use a test that splits data into those that lie inside the region R and those that lie outside the region in the construction of regression trees. Experiments confirm that the use of region splitting gives compact and accurate regression trees in many domains.

Keywords: regression tree, region splitting, range splitting

1. Introduction

We consider a regression problem for modeling and approximating the values of a continuous attribute in a relational table. In a table, we often treat one continuous attribute as special, and call it the *objective attribute*. The other attributes are called *conditional attributes*. Several models have been proposed for modeling and approximating the values of a continuous objective attribute by using the values of conditional attributes, such as statistical linear regression, spline functions, and neural networks. *Regression trees* are also a widely used method for such modeling tasks.

A regression tree is a rooted tree such that each intermediate node is associated with a test. At an intermediate node (initially at the root node), we check the test of the node to find out which branch a given record should move to. We assume a regression tree is a binary

---

*This paper is based on the extended abstract of "Efficient Construction of Regression Trees with Range and Region Splitting" in *Proceedings of the 23rd VLDB Conference* (Athens, Greece, 26–29 August). VLDB Endowment, 1997, pp. 166–175.

†This work was performed while the author was at University of Tokyo.

††This work was performed while the author was at IBM Tokyo Research Laboratory.

tree structure. Therefore, if the record satisfies the test, it goes down one branch. Otherwise, it goes down the other branch. The record is recursively checked at intermediate nodes and finally reaches a leaf node. Each leaf lists a predicted value for the objective attribute that is appropriate to the test conditions along the path to that leaf. We can predict the value of the objective attribute for a given record as the value given at the associated leaf.

Various proposals for constructing compact and accurate trees from a (training) table of data have been made in the AI community (Breiman et al., 1984; Quinlan, 1993; Piatetsky-Shapiro & Frawley, 1991) and recently efficient construction of classification trees from large tables has also been investigated in the database community (Mehta, Agrawal, & Rissanen, 1996; Shafer, Agrawal, & Mehta, 1996; Gehrke, Ramakrishnan, & Ganti, 1998).

This paper explores an improvement to the standard strategy of selecting internal-node tests during regression tree construction. In particular, conventional regression tree construction algorithms often use a test on a continuous, conditional attribute with a cut value within the range of the conditional attribute that splits data into those below the cut value and those above. Such a test is called a "guillotine-cut." However, consider the case when two conditional numeric attributes have a non-linear correlation with respect to the objective numeric attribute. In such a case, a test with a two-dimensional region for splitting data into two subsets may be more natural and effective.

As an example, consider Table 1, which shows part of the relational table that was collected from the New York currency markets every Monday, from the last Monday of 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, US$/pound), GDM (Deutschmark, US$/mark), YEN (Japanese Yen, US$/yen), TB3M (3-month Treasury bill yields), TB30Y (30-year Treasury bill yields), GOLD (US$/ounce), and SP500 (Standard and Poors Index). Suppose we are interested in predicting the SP500, which is one index of overall market performance, from the values of other attributes in the table, using 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 values of the SP500, which range from 190 to 460 and are divided into sub-ranges with a uniform width of ten. The data in the top histogram are then split by the region R in the middle of the figure. The 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 show the data distribution of records inside the region R and that of records outside the R,

Table 1. Currency data.

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

respectively. The number of records, and the mean and variance of SP500 values in those three histograms are:

	No. of tuples	Mean	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 intermediate nodes until we cannot significantly improve the mean squared error of the tree.

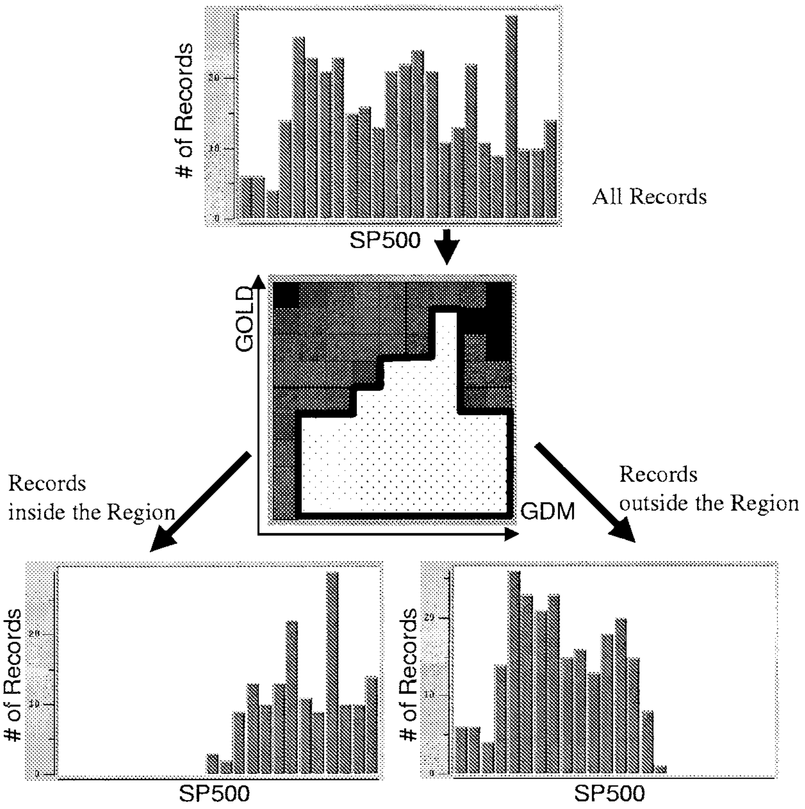


Figure 1. Region splitting.

In the remainder of the paper, we describe our variation of regression tree construction, which can form tests using two-dimensional regions and standard range cuts, as well as tests of categorical attributes. We show that the *optimal* region that minimizes the mean squared error can be efficiently computed, and through experiments in a variety of domains, show that a regression tree with region splitting tends to be more accurate and smaller than a conventional tree with guillotine-cuts only.

2. Regression tree construction

Let $\mathcal{D}$ denote a relational schema, which is a set of categorical or numeric attributes. We select a numeric attribute Y as the objective attribute. We call the other attributes in $\mathcal{D}$ conditional attributes.

We generate a regression tree T from a specific instance of relation, D, of $\mathcal{D}$. D is called the *training* data. Let t be a record of $\mathcal{D}$, and let $t[Y]$ denote the value of the objective attribute Y. Let w denote a node in a regression tree T, and let D_w denote the set of records in D that are classified by node w. We assume that D_w is not empty. Let $|D_w|$ denote the number of records in D_w, and let $\mu(D_w)$ denote the average value of the objective attribute Y in D_w; that is, $\mu(D_w) = (1/|D_w|)\sum_{t \in D_w} t[Y]$. By recursively following tests at internal nodes of T, classification of any record follows a single path from the root to a leaf, which we denote $\mathit{leaf}(t)$. We say that t is classified by the leaf node $\mathit{leaf}(t)$. Suppose that we are given another instance of the relation D' over $\mathcal{D}$. For each record t in D', we find the leaf node $\mathit{leaf}(t)$. We use $\mu(D_{\mathit{leaf}}(t))$ as the predicted value for t, whose actual value is $t[Y]$. To evaluate the prediction error of T against relation D', we use the mean squared error

$$MSE(T, D') = \frac{\sum_{t \in D'} \left(t[Y] - \mu\left(D_{\mathit{leaf}(t)}\right)\right)^2}{|D'|}.$$

When we are given training data D, we do not know what instances D' 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 data D, namely $MSE(T, D)$, is small.

2.1. Greedy tree construction

The problem of constructing a minimum regression tree from a given set of training records is known to be NP-hard (Hyafil & Rivest, 1976; Garey & Johnson, 1979). Therefore, we have to use an approximation algorithm for the problem.

The most common method has been the classic greedy recursive partitioning strategy, which is summarized as Algorithm 2.1. If a test splits a set of records D_w in a node w into two subsets D_w^{left} and D_w^{right}, the mean squared error of the node becomes

$$\frac{\sum_{t \in D_w^{\mathit{left}}} \left(t[Y] - \mu\left(D_w^{\mathit{left}}\right)\right)^2 + \sum_{t \in D_w^{\mathit{right}}} \left(t[Y] - \mu\left(D_w^{\mathit{right}}\right)\right)^2}{|D_w|}. \tag{1}$$

In the greedy tree construction, we find a test (step 5-6 in **SPLIT**) that minimizes Formula 1 at each split. Algorithm 2.1 is a greedy algorithm that constructs a regression tree T from D. We stop splitting a tree if the number of records in D_w becomes negligible.

ALGORITHM 2.1. **Regression Tree Construction**

```
MAIN ()
    1. Read training data D in a database.
    2. SPLIT ($D$)

SPLIT ($D_w$)
    1. IF ($D_w$ satisfies stopping condition) THEN stop.
    2. Examine all possible tests on each categorical attribute.
    3. Examine all possible guillotine cut tests on each numeric attribute.
    4. Examine all possible region tests on each pair of numeric attributes.
    5. Find the optimal test that most reduces MSE($T, D_w$).
    6. Split $D_w$ into $D_w^{left}$ and $D_w^{right}$ by using the optimal test.
    7. SPLIT ($D_w^{left}$)
    8. SPLIT ($D_w^{right}$)
```

2.2. *Tests on categorical conditional attributes*

For a categorical conditional attribute A, a *primitive* test has the form $A = a$, where a is an element in the domain of A. A record t satisfies a test of the form $A = a$ if $t[A] = a$. A test on a categorical conditional attribute is an arbitrary Boolean disjunction of such primitive tests, such as "$A = a \vee A = b \vee \cdots$."

Assume that the cardinality of A is m. We can order all the m primitive tests according to the average value of Y. The optimal test that minimizes Formula 1 must be one of the cuts of the ordered primitive tests. Therefore, we can find the optimal test in $O(m \log m)$ time. Breiman, Friedman, Olshen and Stone showed the $O(m \log m)$ algorithm for the GINI index criterion (Breiman et al., 1984). We will show, later in this paper, that the mean squared error criterion, which is defined by Formula 1, has the same property when applied to this algorithm.

2.3. *Tests on numeric conditional attributes*

For tests on numeric attributes, most regression trees use a test of the form $B < b$ using a numeric attribute B, i.e., *guillotine-cut* splitting. Suppose that we select a node w to partition. Among all the instances of guillotine-cut splitting, conventional methods select the optimal one, say $B < b$, that minimizes the mean squared error. $B < b$ partitions D_w into $D_w^{left} = \{t \in D_w \mid t[B] < b\}$ and $D_w^{right} = \{t \in D_w \mid b \leq t[B]\}$.

Our algorithm allows guillotine-cut splitting through tests of the form $B < b$, but we also allow tests of the form $B \in [b_1, b_2]$ and $(B, C) \in R$, where b, b_1, and b_2 are elements in the domain of conditional attributes B, and R is a connected region in the Euclidean plane associated with conditional attributes 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.

To repeat the motivation for region splits, there are many strong correlations among numeric attributes, which should be considered during test selection. Some work on classification and regression trees allow multivariate tests of the form $\Sigma_i a_i B_i < c$, where a_i and c are constants, to handle correlations (Breiman et al., 1984; Murthy, Kasif, & Salzberg, 1994; Brodley & Utgoff, 1995). Though such multivariate tests can handle combination of many numeric attributes, linear partitioning by those tests does not directly account for non-linear correlations. Figure 1 is a typical example of such a non-linear correlation. To handle non-linear correlations, Ittner and Schlosser (1996) considers composite attributes that include all possible pairwise products and squares of numerical attributes, and finds a linear partitioning on those composite attributes. This approach is an alternative to our use of regions.

2.4. *Pruning regression trees*

From training data, we can generate large regression trees through recursive partitioning. Larger regression trees can reduce the mean squared error in the training data but are likely to *overfit* the training data, giving a higher mean squared error against test data. To avoid this, we have to stop expanding leaves before a regression tree overfits the training data, or prune unnecessary leaves from an overfitted tree. The former approach is sometimes called *prepruning*, while the latter approach is called *postpruning*.

Though the prepruning methods can reduce computation time, the accuracy of the trees produced by postpruning is, in general, higher than that of prepruned trees. Therefore, postpruning methods are very popular. Since pruning methods have a great effect on the accuracy of trees, various postpruning methods have been intensively explored (Mingers, 1989). Nonetheless, we applied a variant of *cost-complexity* pruning, which is a popular prepruning method used in CART (Breiman et al., 1984), since our primary goal was to explore the utility of region splitting with respect to the accuracy of trees of different sizes.

Suppose that the set of records D_w at a node w is split into D_w^{left} and D_w^{right} so that the Formula 1 is minimized. Before the split, the mean squared error at node w is

$$\frac{\sum_{t \in D_w} (t[Y] - \mu(D_w))^2}{|D_w|}. \tag{2}$$

The split is effective if the difference between Formula 1 and Formula 2 at the node is large, because the split reduces the mean squared error of the regression tree. Let us call the difference the *profit* of the split. If the profit is small, the split is not effective and we prune the subtree of the node. To this end, we provide a threshold for the profit, and we prune the subtree of the node if we cannot find any split with profit that is more than the threshold.

We used the following threshold:

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

where D denotes the set of all the records in the training dataset, and α is a special parameter called a *weight parameter*. This threshold is equivalent to the AID criteria in Breiman et al. (1984).

The choice of the weight parameter α strongly affects the mean squared error against the test data. A larger value of the parameter is more likely to prune subtrees and hence to produce smaller regression trees, while a smaller value of the parameter generates a larger regression tree that might overfit the training data. We need to control the parameter value to yield a better regression tree with a smaller mean squared error. We use a separate dataset called the *validation set* to find an appropriate value of the parameter and use that value for pruning trees.

3. Range and region splitting

In this paper, we propose considering a broader class of tests than guillotine cuts, such as range splits of the form $B \in [b_1, b_2]$ and region splits 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 node w to split. Define D_w^{in} and D_w^{out} to be the subsets of records D_w, that are inside and outside a region, respectively.

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

We consider how to find the region R that minimizes

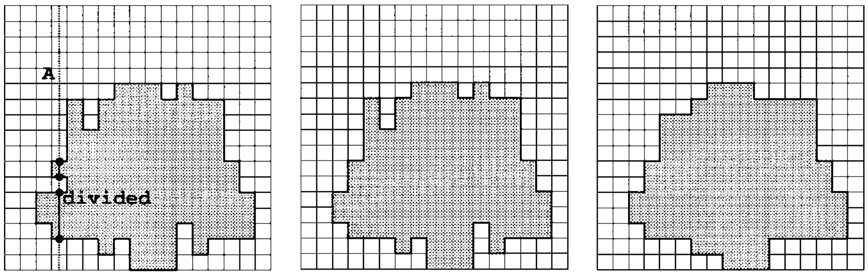
$$\frac{\sum_{t\in D_w^{in}}\left(t[Y]-\mu\left(D_w^{in}\right)\right)^2+\sum_{t\in D_w^{out}}\left(t[Y]-\mu\left(D_w^{out}\right)\right)^2}{|D_w|},$$

which we denote by $U(R)$.

3.1. Grid regions

We present three classes of regions that can be defined on a two-dimensional pixel grid plane G. To define those regions, we first distribute the values of B (resp. C) into N_B (N_C) buckets so that every bucket contains almost the same number of records. 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 union of pixels in G that are connected. An *x-monotone* region is a grid region whose intersection with any vertical line is undivided. For example, figure 2(a)



(a) Not X-Monotone (b) X-Monotone (c) Rectilinear

Figure 2. Region families.

shows a region that is not x-monotone, since the intersection of the vertical line A and the region is divided. In contrast, figure 2(b) shows an x-monotone region. We can express an x-monotone region by a disjunction of expressions, where each disjunct has the form `(a1 < x < a2)` and `(b1 < y < b2)`, where the union of disjuncts does not divide a vertical column. A *rectilinear* convex region is an x-monotone region such that its intersection with any horizontal line is also undivided. Figure 2(c) shows an example of a rectilinear convex region. A *rectangular* region is a rectangle on G, and is thus a rectilinear convex region. We will consider the class of x-monotone regions, the class of rectilinear convex regions, and the class of rectangular regions.

3.2. 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

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

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

Lemma 3.1. *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:

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

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

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

Since $\sum_{t\in D_w} t[Y]^2$ 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 $U(R)$. □

3.3. *The 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 $t[Y]$ by $t[Y] - \mu\left(D_w\right)$ 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

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

Let x denote $|D_w^{in}|$, let y be $\sum_{t\in D_w^{in}} t[Y]$, 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

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

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 $y = \sum_{t\in D_w^{in}} t[Y]$. Consider the convex hull of all those stamp points. Since the number of x-monotone regions (or rectilinear convex 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 3.1. *$f(x, y) = y^2(\frac{1}{x} + \frac{1}{M-x})$ is convex in the region $M > x > 0$; namely,*

$$(1-\gamma)f(x_1, y_1) + \gamma f(x_2, y_2) \geq f((1-\gamma)x_1 + \gamma x_2, (1-\gamma)y_1 + \gamma y_2)$$

for $0 \leq \gamma \leq 1$ and 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 $\delta_1, \delta_2 \neq 0$.

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

Next,

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

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

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

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

3.4. *Computing a stamp point on a convex hull*

A point of the convex hull and its corresponding region can be computed efficiently by using the "hand-probing" technique, invented by Asano, Chen, Katoh, and Tokuyama (Asano et al., 1996) for image segmentation and modified by Fukuda, Morimoto, Morishita, and Tokuyama (Fukuda et al., 1996) for extraction of the two-dimensional optimized association rules. Geometrically, hand-probing computes a tangential point of a line (with a certain slope θ) and the convex hull of all stamp points. A line with a slope θ can be characterized as $y - \theta x = a$, where a is a constant. The tangential point with the line maximizes (or minimizes) $y - \theta x$ among all stamp points.

Let $x(i, j)$ be a number of records in the (i, j)-th pixel in G, and let $y(i, j)$ be $\sum_{t \in (i,j)\text{-}th\ pixel} t[Y]$. Given a vector θ, we define $g(i, j) = y(i, j) - \theta x(i, j)$.

Let us consider an x-monotone region associated with the slope θ. For each $m = 1, 2, \ldots, N$, we precompute the indices $bottom_m(s)$ and $top_m(s)$ for all $1 \leq s \leq N$, where $bottom_m(s)$ and $top_m(s)$ are defined so that $\sum_{i=bottom_m(s)}^{s} g(i, m)$ and $\sum_{i=s}^{top_m(s)} g(i, m)$ are maximized (or minimized) respectively. This pre-computation takes $O(N)$ time.

For two indices s and s' (such that $s \leq s'$), we define

$$cover_m(s,\ s') = \sum_{i=bottom_m(s)}^{top_m(s')} g(i,\ m).$$

We define $F(i, \leq m)$ to be the partial x-monotone region that contains the $(m,\ i)$-th pixel and maximizes (or minimizes) the summation of $g(i,\ j)$ over pixels in the region on the left side of the m-th column of G. Then, we have the following formula:

$$F(i, \leq m+1) = max_{1 \leq j \leq N}\{F(j, \leq m) + cover_{m+1}(i,\ j)\}$$

or 0 if it takes negative value. From this formula, we can compute $max_m\{max_i F(i, \leq m)\}$ and the associated region, which must maximize (or minimize) the summation of $g(i,\ j)$. If $F(j, \leq m)$ for $1 \leq j \leq N$ are given, we can compute $F(i, \leq m+1)$ for $1 \leq i \leq N$ in $O(N)$ time. Thus, this dynamic programming can be performed in $O(N^2)$ time for a slope θ. Figure 3 shows a conceptual image of the dynamic programming. Detailed algorithms and proofs are in Fukuda et al. (1996).

A rectilinear convex region is also composed of connected vertical columns. Since its intersection with any horizontal line must be undivided, we use another dynamic programming to compute a grid region that maximizes (or minimizes) the summation of $g(i, j)$. There are four types of dynamic programming for concatenating a new column to the previous region: adding a wider column (W), adding a column slanting upward (U), adding a column slanting downward (D), and adding a narrower column (N). Figure 4 conceptually shows those dynamic programming and valid combinations of them. Dynamic programming gives us an $O(N^3)$ time solution for computing a rectilinear convex region. Detailed algorithms and proofs are in Yoda et al. (1997).

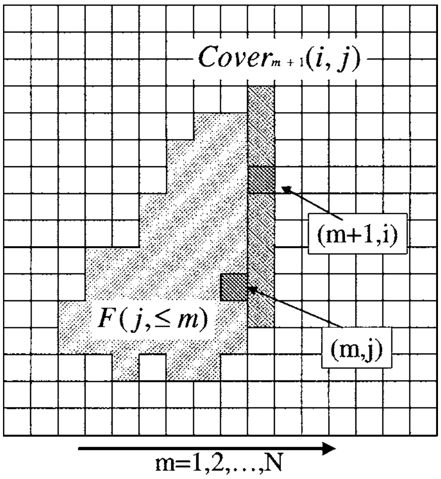


Figure 3. Dynamic programming for an X-monotone region.

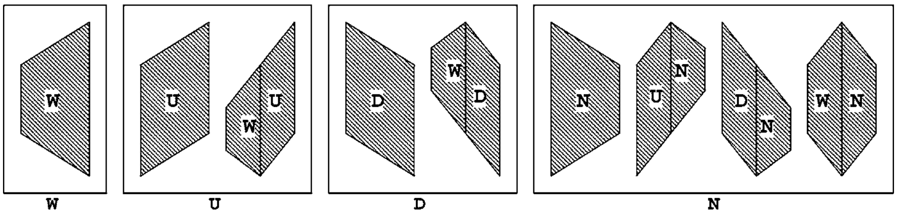


Figure 4. Dynamic programming for a rectilinear convex region.

The time complexity of computing a rectangular region is also $O(N^3)$.

3.5. Searching on the convex hull

To search for the stamp point associated with the region R that maximizes $V(R)$, Theorem 3.1 implies that we can focus on scanning the convex hull of all the stamp points. 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, when the number of records is n, the number of points on the hull is at most n if we consider x-monotone regions or rectilinear convex regions. Therefore, in the worst case, we may need to try hand-probing n times in order to find the optimal stamp point, which is costly in practice. To avoid searching for unnecessary points, we use a guided branch and bound strategy.

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 $I(\mathit{left})$ and $I(\mathit{right})$, and consider the interval $I = [I(\mathit{left}), I(\mathit{right})]$ in figure 5. Let $Q(I) = (x_{Q(I)}, y_{Q(I)})$ be the point of intersection of the two tangent lines that are used to compute $I(\mathit{left})$ and $I(\mathit{right})$. We can compute the interclass variance value of $Q(I)$, $f(Q(I)) = f(x_{Q(I)}, y_{Q(I)})$.

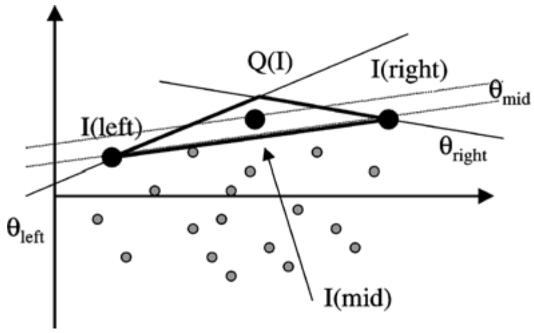


Figure 5. Guided branch-and-bound search.

Corollary 3.1. *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 3.1. □

This corollary gives an upper bound of points on the convex hull between these two points. Hence, we can find the optimal region effectively by 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 the subintervals are expected to be pruned away during the computation, and therefore the number of hand-probings is expected to be $O(\log n)$, where n is the number of points on the convex hull. Algorithm 3.1 sketches the branch and bound search, which is substantially the same as we used for finding the optimal region that minimizes entropy in the two class classification problem (Morimoto et al., 1997).

ALGORITHM 3.1 **Branch and Bound Search**

```
MAIN ()
  1. Initialize the set of candidate intervals $S$.
  2. Compute the initial interval $I_{init} = [I_{init}(left), I_{init}(right)]$ using $\theta = \infty$.
  3. $f(Q(I_{init})) = \infty$
  4. $V_{max} = max\{f(I_{init}(left)), f(I_{init}(right))\}$
  5. Insert $I_{init}$ into $S$.
  6. Choose $I$ that has the maximal $f(Q(I)$ from $S$.
  7. IF $V_{max} < f(Q(I))$, THEN call (BranchBound (I) and goto Step 6, ELSE stop
     searching.

BranchBound (I)
  1. Let $\theta_{mid}$ be the slope of the line containing $I(left)$ and $I(right)$.
  2. Compute a new point $I(mid)$ using $\theta_{mid}$.
  3. If $V_{max} > f(I(mid))$ then $V_{max} = f(I(mid))$.
  4. Divide $I$ into $I_{left} = [I(left), I(mid)]$ and $I_{right} = [I(mid), I(right)]$ and compute
     $f(Q(I_{left}))$ and $f(Q(I_{right}))$.
  5. Insert $I_{left}$ and $I_{right}$ into $S$.
```

Recall that the time complexities of performing hand-probing for an x-monotone region, a rectilinear convex region, and a rectangular region 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 $O(N^2 \log n)$, $O(N^3 \log n)$, and $O(N^3 \log n)$ for x-monotone regions, rectilinear convex regions, and rectangular regions, respectively.

3.6. Record density of a pixel

We uniformly distribute each numeric attribute into N ordered buckets. Next, for each pair of numeric attributes, we create an $N \times N$ pixel grid G. The average number of records in each pixel on G, which we call the *record density*, influences the accuracy of the regression trees. Using a lower record density is likely to make the regression trees overfit the training dataset, while a coarse grid with a higher record density often fails to find variously shaped regions.

We empirically found that a record density ranging from five to ten gives a lower mean squared error for test datasets, and we therefore use a record density of five or ten in the later experimental section. The experimental results for a record density of five show the effects of relatively fine grid regions, while those for a record density of ten show the effects of coarse grid regions.

In the process of generating regression trees, the number of records becomes smaller at nodes lower in the tree. In order to guarantee a record density of five or ten, we are forced to use a coarse grid, say 2×2, which is too rough to generate interesting regions. If we have to use a coarse grid whose size is less than 5×5, we employ guillotine-cut splitting instead.

3.7. Spline interpolation of grid regions

We compute the optimal region on a discretized pixel grid. However, such grid regions tend to be too coarse to describe regions if there are not enough records. In such cases, we use an interpolation technique to get smooth regions. Since the number of records becomes smaller at nodes lower in regression trees, interpolation techniques are indispensable.

The optimal region on a pixel grid is expressed by a disjunction of the form `(a1 < x < a2)` and `(b1 < y < b2)`. Therefore, we can plot two points $(\frac{a1+a2}{2}, b1)$ $(\frac{a1+a2}{2}, b2)$ for each column of the grid region. Then, we compute a smooth region from those points.

We compute two kinds of spline curves, spline1 and spline2, of interpolated regions from a set of points on a pixel grid. Figure 6 shows examples of the two kinds of interpolated regions. In each pixel grid, the gray pixel grid region shows the optimal region that is computed by our algorithm. The points on the border of the gray region are used to compute the interpolated region.

Figure 7 illustrates how to compute a spline curve from a set of points. We first order points of the set by x values and make a point sequence. Then, we split the point sequence into several intervals as in the figure. For spline1, we cut the sequence at the x value for each point. For spline2, we use the middle value of each two consecutive points in the sequence. For each interval i, we define a polynomial function of the form $f_i(x) = ax^3 + bx^2 + cx + d$ where a, b, c, and d are constants, and $x_i \le x \le x_{i+1}$ for spline1, and $\frac{x_i + x_{i+1}}{2} \le x \le \frac{x_{i+1} + x_{i+2}}{2}$ for spline2.

Each polynomial function, say $f_i(x)$, of spline1 is defined so that the following conditions are satisfied.

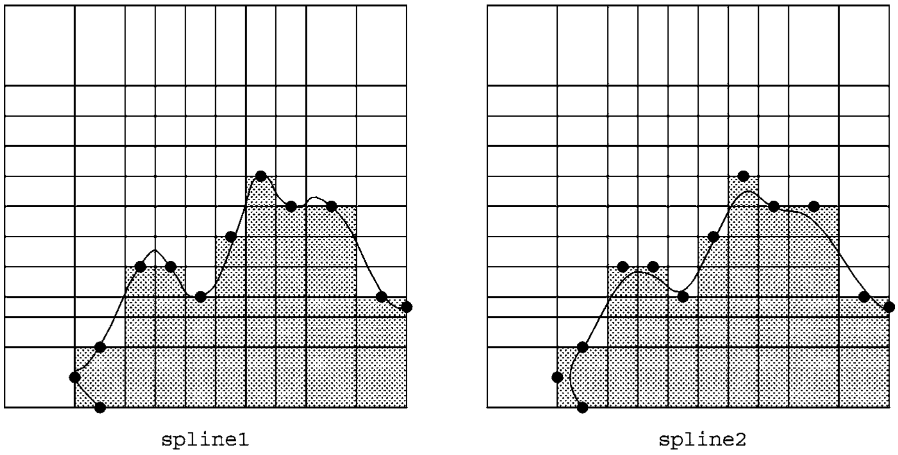


Figure 6. Interpolated regions.

- $f_i(x)$ must pass through $\mathbf{x_i}$ and $\mathbf{x_{i+1}}$ · ($\mathbf{x_i} = (x_i,\ y_i)$ is the i-th point of a point sequence.)
- $f_i'(x_{i+1}) = f_{i+1}'(x_{i+1})$ · ($f_i'(x)$ is the 1st derivative of $f_i(x)$.)
- $f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$ · ($f_i''(x)$ is the 2nd derivative of $f_i(x)$.)

Each polynomial function of spline2 satisfies the following conditions.

- $f_i(x)$ must pass through $\frac{\mathbf{x_i}+\mathbf{x_{i+1}}}{2}$ and $\frac{\mathbf{x_{i+1}}+\mathbf{x_{i+2}}}{2}$.
- $f_i'(\frac{x_{i+1}+x_{i+2}}{2}) = f_{i+1}'(\frac{x_{i+1}+x_{i+2}}{2}) = \frac{y_{i+2}-y_{i+1}}{x_{i+2}-x_{i+1}}$.

Though interpolated regions may be worse than the original pixel grid regions with respect to the mean squared error of training datasets, those interpolated regions are much more accurate for test datasets, as we will show in the experimental results.

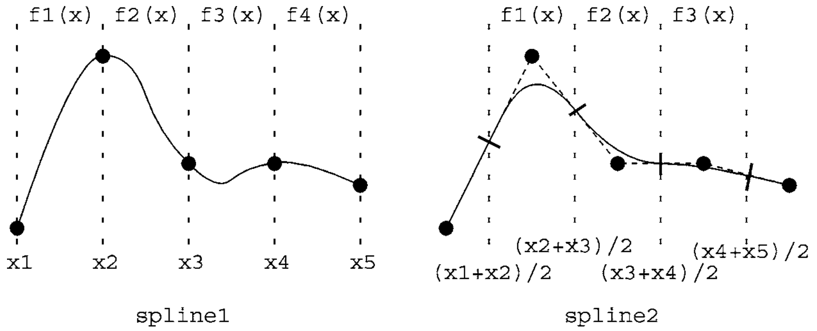


Figure 7. Spline interpolation.

4. Experimental results

To examine prediction accuracy, 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 and there are enough records for effective region splitting.

4.1. Minimal potential error

To examine the minimal potential error of trees, we performed the following ten-fold cross-validation tests for each of the 10 datasets:

1. We randomly divided the original dataset into ten almost-equal-sized subsets.
2. We excluded one subset and used the other nine subsets as the training dataset to generate regression trees with each of 4 types of splitting: guillotine cuts, x-monotone regions, rectilinear convex regions, and rectangular regions.
3. We used the excluded subset as the test dataset to evaluate the regression tree, and computed the mean squared error against the test dataset. The absolute value of the mean squared error varies depending on the sample dataset. To compare the mean squared errors of different dataset, we used a "normalized" version called the *relative mean squared error* of a regression tree, which is defined as the mean squared error divided by the variance of the objective attribute.
4. We returned to Step 2 and excluded a different subset. If there is no remaining subset for the test dataset, then we calculated the average and standard deviation of the ten results.

To examine the effectiveness of region splitting and eliminate the effect of pruning, we generated a number of regression trees for various values of the weight parameter, then we

Table 2. Dataset summary.

Dataset	#records	#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

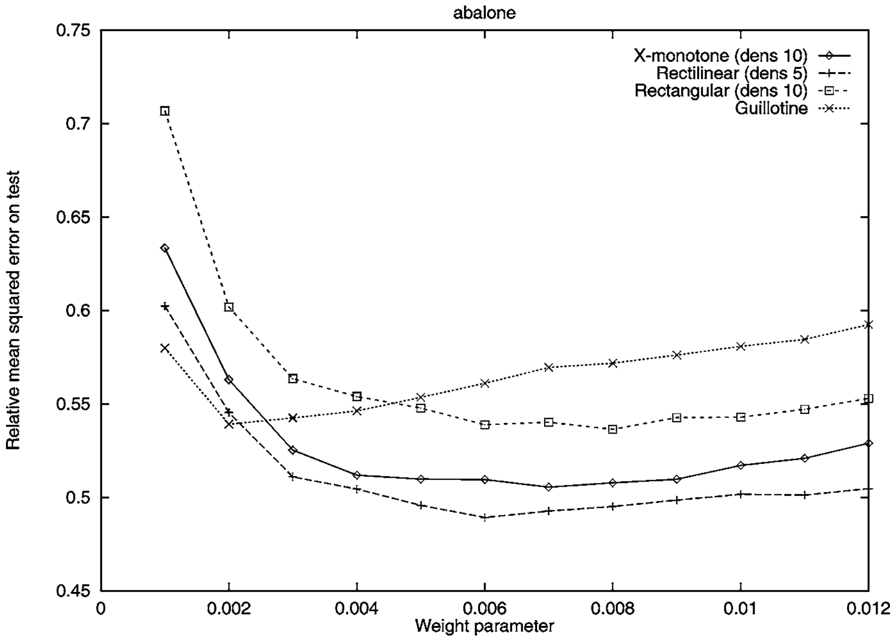


Figure 8. Accuracy for “abalone”.

compared the best accuracy of those various-sized regression trees, and call it the minimal potential error.

For instance, let us consider the “abalone” dataset. Figure 8 shows how the relative mean squared error changes according to the value of the weight parameter. The graph for “Rectilinear (dens 5)” shows the relative mean squared errors of the regression trees with rectilinear convex regions for a record density of five, and we see that the relative mean squared error is smallest when the weight parameter is 0.006. Observe that the value increases for weight parameters less than 0.006, 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.

For the other datasets, we performed the same analysis, which shows the same general trends as figure 8 (i.e., generally, a lower error for rectilinear (dens 5) over alternatives across weight values, and similar shape curves for each method due to under and overfitting). Tables 3–5 summarize the results, showing the pairs of the average relative mean squared error and the average number of leaves for trees constructed with the best parameter values, i.e., weight parameter α and pixel density. The number in each parenthesis is the standard deviation over the ten results of cross validation. We underlined the smallest relative mean squared error among all split strategies (x-monotone, rectilinear convex, rectangular, and guillotine) for each dataset. Almost universally, the rectilinear, spline2 method yields the lowest error rates across all domains. While we do not indicate the significance of differences

Table 3. Minimal potential error (X-monotone).

	Pixel Grid		Spline1		Spline2	
Dataset	Err	Size	Err	Size	Err	Size
abalone	0.523 (0.0397)	10.0 (0.63)	0.523 (0.0481)	9.1 (0.83)	0.505 (0.0490)	8.6 (.917)
add10	0.141 (0.00854)	117.6 (4.34)	0.135 (0.00870)	147.7 (12.1)	0.127 (0.00969)	126.5 (9.87)
kin-8fh	0.450 (0.0331)	49.5 (3.20)	0.449 (0.0281)	38.8 (2.32)	0.444 (0.0190)	39.0 (1.55)
kin-8fm	0.224 (0.0132)	157.8 (8.35)	0.222 (0.0172)	95.2 (3.46)	0.211 (0.0127)	95.2 (3.54)
kin-8nh	0.661 (0.0538)	35.7 (1.79)	0.648 (0.0520)	43.0 (4.88)	0.625 (0.0374)	32.2 (1.89)
kin-8nm	0.497 (0.0270)	57.0 (3.19)	0.487 (0.0212)	63.4 (4.92)	0.465 (0.0206)	62.8 (4.02)
pumadyn-8fh	0.410 (0.0187)	8.0 (0.00)	0.405 (0.0232)	8.0 (0.00)	0.403 (0.0206)	8.0 (0.00)
pumadyn-8fm	0.0607 (0.00726)	15.9 (0.49)	0.0588 (0.00587)	16.0 (0.00)	0.0574 (0.00585)	17.0 (0.77)
pumadyn-8nh	0.350 (0.0272)	7.8 (0.40)	0.344 (0.0260)	8.3 (0.64)	0.341 (0.0250)	10.3 (0.90)
pumadyn-8nm	0.0526 (0.00415)	20.0 (0.63)	0.0487 (0.00362)	39.4 (3.72)	0.0480 (0.00338)	35.1 (2.70)

between methods within a domain because cross validation violates sample independence assumptions, the number of "wins" across domains for rectilinear, spline2 with respect to any other method is significant by a sign test at the 0.05 level ($N = 10$), though if kin-8XX and pumadyn-8XX are not considered independent, then the number of wins for rectilinear, spline2 is only marginally significant by a sign test ($N = 4$) with respect to each alternative strategy.

4.2. Error rates of pruned trees

For minimal potential error, the weight parameter value and record density value giving the minimum error ratio greatly depends on the dataset and construction method. In

Table 4. Minimal potential error (rectilinear).

	Pixel Grid		Spline1		Spline2	
Dataset	Err	Size	Err	Size	Err	Size
abalone	0.522 (0.0436)	8.8 (0.98)	0.500 (0.0427)	9.9 (0.94)	0.489 (0.0463)	8.8 (0.75)
add10	0.121 (0.00618)	121.3 (8.00)	0.118 (0.00719)	127.4 (6.83)	0.114 (0.00458)	123.9 (6.76)
kin-8fh	0.433 (0.0241)	59.8 (2.14)	0.419 (0.0274)	55.7 (1.85)	0.412 (0.0223)	55.3 (2.93)
kin-8fm	0.196 (0.0106)	138.7 (4.45)	0.197 (0.00865)	115.2 (5.31)	0.193 (0.00793)	150.6 (6.59)
kin-8nh	0.613 (0.0437)	30.1 (2.07)	0.608 (0.0422)	33.7 (2.57)	0.610 (0.0388)	47.7 (2.41)
kin-8nm	0.450 (0.0365)	93.1 (3.27)	0.451 (0.0234)	51.7 (1.19)	0.440 (0.0265)	67.1 (3.91)
pumadyn-8fh	0.401 (0.0208)	8.0 (0.00)	0.400 (0.0199)	8.2 (0.40)	0.400 (0.0178)	8.0 (0.00)
pumadyn-8fm	0.0600 (0.00628)	15.8 (0.40)	0.0581 (0.00545)	16.3 (0.64)	0.0572 (0.00575)	19.5 (1.36)
pumadyn-8nh	0.339 (0.0287)	8.1 (0.30)	0.335 (0.0260)	8.0 (0.00)	0.333 (0.0274)	8.3 (0.46)
pumadyn-8nm	0.0501 (0.00373)	26.6 (1.36)	0.0471 (0.00428)	40.5 (4.15)	0.0468 (0.00456)	39.4 (2.33)

Table 5. Minimal potential error (conventional).

	Rectangular		Guillotine	
Dataset	Err	Size	Err	Size
abalone	0.536 (0.0352)	10.1 (0.700)	0.539 (0.0480)	38.9 (2.51)
add10	0.156 (0.00932)	148.9 (5.36)	0.185 (0.00746)	540.2 (7.19)
kin-8fh	0.460 (0.0256)	69.9 (3.33)	0.479 (0.0299)	128.3 (3.41)
kin-8fm	0.257 (0.0186)	162.1 (3.88)	0.249 (0.0164)	919.9 (7.91)
kin-8nh	0.618 (0.0398)	37.9 (1.81)	0.655 (0.0389)	88.4 (6.23)
kin-8nm	0.477 (0.0217)	59.5 (3.47)	0.541 (0.0499)	203.6 (7.49)
pumadyn-8fh	0.409 (0.0223)	19.8 (1.36)	0.410 (0.0184)	26.2 (2.09)
pumadyn-8fm	0.0655 (0.00494)	62.6 (3.61)	0.0632 (0.00683)	153.9 (5.09)
pumadyn-8nh	0.353 (0.0304)	27.3 (2.00)	0.355 (0.0317)	44.1 (1.45)
pumadyn-8nm	0.0541 (0.00335)	99.9 (2.47)	0.0535 (0.00367)	185.0 (4.94)

comparison of the error rate, we use a separate dataset, designated as the validation set, to find the adequate weight and density values for pruning.

We reserved one subset as the validation set, which is arbitrarily chosen from the nine training subsets during each trial of the 10-fold cross validation. That is, we used eight subsets for tree building and used one subset for finding the parameters, i.e., the weight parameter and the record density. We pruned a regression tree by the best parameter setting for the validation set and examined its error rate for the remaining test sets.

Tables 6–8 summarizes the results of the 10-fold cross validation experiment.

The cross-validation results can be summarized as follows:

- X-monotone and rectilinear convex regions attain more accurate and smaller trees than conventional trees (at marginal significance for error rate by a sign test).
- Spline interpolation makes region splitting more accurate. Even though the spline2 sacrifices training accuracy (see Section 3.7), it attains the most accurate results (for tests) in most of the cases.
- Rectilinear regions are robust even if we use a low record density. Therefore, the results of a record density of five were more accurate than those of a record density of ten in most of the cases. As a result, rectilinear convex regions won over all of other splitting methods on all datasets.
- In case of x-monotone regions, on the other hand, using a low record density tends to give a higher error ratio because of overfitting. Therefore, the results for a record density of ten were more accurate than those for a record density of five in most of the cases.
- By using a validation set, we found parameter values that attained to satisfactory results as regards accuracy.

Table 6. Error of pruned tree (X-monotone).

	Pixel grid		Spline1		Spline2	
Dataset	Err	Size	Err	Size	Err	Size
abalone	0.535 (0.0573)	10.6 (4.27)	0.530 (0.0615)	10.5 (4.30)	0.519 (0.0432)	8.6 (1.56)
add10	0.141 (0.00986)	119.7 (26.1)	0.138 (0.0145)	134.1 (23.9)	0.131 (0.0108)	108.7 (18.2)
kin-8fh	0.465 (0.0233)	45.8 (18.0)	0.453 (0.0187)	42.5 (9.70)	0.448 (0.0148)	47.1 (20.8)
kin-8fm	0.240 (0.0194)	148.1 (89.9)	0.240 (0.0129)	122.4 (34.8)	0.234 (0.0126)	144.8 (63.4)
kin-8nh	0.683 (0.0348)	31.7 (11.4)	0.655 (0.0303)	32.8 (10.0)	0.644 (0.0393)	34.5 (21.3)
kin-8nm	0.503 (0.0315)	46.4 (11.9)	0.493 (0.0322)	53.6 (17.5)	0.476 (0.0241)	70.1 (36.6)
pumadyn-8fh	0.416 (0.0243)	7.90 (1.14)	0.410 (0.0246)	8.90 (1.04)	0.402 (0.0232)	8.30 (0.46)
pumadyn-8fm	0.0611 (0.00348)	17.9 (3.75)	0.0601 (0.00515)	19.1 (6.04)	0.0584 (0.00398)	16.4 (2.54)
pumadyn-8nh	0.349 (0.0242)	7.90 (1.04)	0.343 (0.0190)	9.1 (2.66)	0.343 (0.0195)	9.1 (2.07)
pumadyn-8nm	0.0546 (0.00568)	31.2 (9.31)	0.0508 (0.00529)	41.3 (11.3)	0.0503 (0.00409)	39.7 (16.0)

4.3. *Performance results*

The CPU time taken to construct a regression tree depends on the execution time needed to compute the optimal region for splitting data. Thus, the cost of computing one optimal region gives us an idea of the overall performance in generating one regression tree.

4.3.1. Computing one optimal region. We generated our test data, in the form of an $N \times N$ grid, as follows: We first generated random numbers uniformly distributed in $[N^2, 2N^2]$ and assigned them to the number of records in each pixel. We then assigned $1, \ldots, N^2$ as the sum of the target values in a pixel, from a corner pixel to the central one, proceeding in

Table 7. Error of pruned tree (rectilinear).

	Pixel grid		Spline1		Spline2	
Dataset	Err	Size	Err	Size	Err	Size
abalone	0.511 (0.0585)	9.00 (1.73)	0.501 (0.0584)	9.90 (4.46)	0.511 (0.0472)	14.9 (13.5)
add10	0.122 (0.00880)	115.6 (12.9)	0.121 (0.00634)	111.8 (17.0)	0.117 (0.00673)	124.3 (19.5)
kin-8fh	0.439 (0.0268)	55.6 (20.7)	0.437 (0.0209)	46.1 (10.8)	0.431 (0.0256)	52.8 (16.8)
kin-8fm	0.213 (0.0130)	137.2 (58.3)	0.208 (0.0168)	115.8 (26.1)	0.211 (0.0195)	130.2 (57.2)
kin-8nh	0.620 (0.0246)	36.9 (10.0)	0.620 (0.0169)	40.8 (18.1)	0.615 (0.0182)	37.9 (8.67)
kin-8nm	0.459 (0.0218)	54.6 (17.3)	0.461 (0.0142)	63.2 (14.2)	0.456 (0.0273)	60.3 (10.1)
pumadyn-8fh	0.406 (0.0253)	8.10 (0.54)	0.402 (0.0257)	8.30 (0.90)	0.400 (0.0231)	7.80 (0.60)
pumadyn-8fm	0.0605 (0.00396)	17.3 (2.05)	0.0595 (0.00378)	22.6 (6.30)	0.580 (0.00345)	17.5 (1.86)
pumadyn-8nh	0.338 (0.0208)	8.2 (0.75)	0.338 (0.00540)	9.1 (1.58)	0.336 (0.0228)	10.3 (2.05)
pumadyn-8nm	0.0524 (0.00546)	37.1 (12.5)	0.0483 (0.00540)	33.9 (9.36)	0.0482 (0.00453)	33.7 (9.00)

Table 8. Error of pruned tree (conventional).

	Rectangular		Guillotine	
Dataset	Err	Size	Err	Size
abalone	0.554 (0.0859)	15.0 (3.13)	0.545 (0.0701)	30.2 (9.31)
add10	0.164 (0.00747)	177.6 (36.4)	0.191 (0.00748)	425.6 (124.2)
kin-8fh	0.473 (0.0353)	60.0 (14.5)	0.490 (0.0200)	121.7 (62.2)
kin-8fm	0.257 (0.0193)	156.6 (43.2)	0.252 (0.0133)	800.0 (289.3)
kin-8nh	0.633 (0.0290)	36.4 (12.9)	0.641 (0.0210)	87.2 (20.3)
kin-8nm	0.490 (0.0281)	58.2 (10.6)	0.533 (0.0326)	125.7 (55.8)
pumadyn-8fh	0.414 (0.0307)	18.8 (2.75)	0.408 (0.0239)	24.9 (4.18)
pumadyn-8fm	0.0653 (0.00409)	56.8 (5.60)	0.0646 (0.00467)	99.8 (21.4)
pumadyn-8nh	0.353 (0.0188)	23.9 (4.59)	0.365 (0.0203)	46.6 (14.9)
pumadyn-8nm	0.0584 (0.00592)	84.7 (18.9)	0.0566 (0.00512)	190.1 (68.3)

a spiral fashion. These test data were generated so that the number of points on the convex hull increased sub-linearly to N, by the square root of the number of pixels. We examined the CPU time taken to compute the optimal region and the number of hand probing needed to find the region. We performed all experiments on an IBM RS/6000 workstation with a 604e3 332 MHz CPU and 768 MB of main memory, running under the AIX 4.2 operating system.

Table 9 shows the time (sec.) and number of hand probing that were required in the guided branch and bound algorithm to find the optimal x-monotone (or respectively rectilinear convex, or rectangular) region that minimizes the mean squared error. It shows that the number of hand probing increases very slowly thanks to the guided branch and bound algorithm. Figure 9 confirms that the CPU time follows our scale estimation.

At the root of a regression tree, we may need a large number of pixels to guarantee the specified record density. The number of records, however, decreases in the lower parts of the tree, and the number of pixels soon becomes less than 20×20 for most datasets; hence computing the optimal rectilinear convex region is generally not costly according to Table 10.

Table 9. Time for computing the optimal region (1).

	X-monotone		Rectilinear		Rectangular	
#Pixel	Time	#Touch	Time	#Touch	Time	#Touch
10^2	0.0117	29	0.0148	22	0.00282	17
20^2	0.0522	36	0.120	30	0.0146	23
30^2	0.112	35	0.444	33	0.0429	24
40^2	0.193	34	1.04	32	0.0951	24
50^2	0.372	42	2.27	35	0.199	27

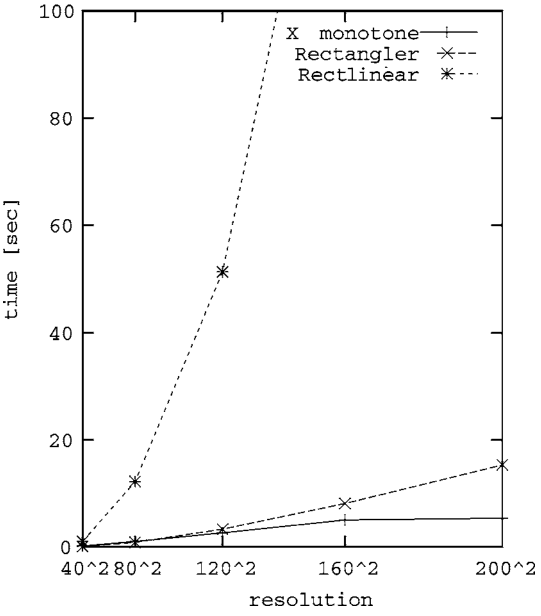


Figure 9. Time for computing an optimal region (2).

4.3.2. Computing one regression tree. The next experiment examines the overall performance in tree construction. At each node of a regression tree, we first prepared grid regions for each combination of numeric attributes, and then computed the optimal region. The grid preparation can be done by scanning all the records at a node just once. However, we have to examine all pairs (permutations for x-monotone regions) of two numeric attributes to find the optimal region. Thus, in general, the number of attributes dramatically affects the performance if we use region splitting.

Table 10 compares the time (sec.) taken to construct trees by using datasets with different numbers of records. We randomly selected records from the "add10" dataset to generate datasets having different numbers of records, and used those datasets to construct a regression tree by performing region splitting with a record density of five. We compared the time taken to construct unpruned large trees using x-monotone (spline2) regions, rectilinear

Table 10. Tree construction time (1).

#Records	X-monotone	Rectilinear	Guillotine
2000	33.3	31.5	4.10
4000	80.2	80.1	9.65
6000	130	136	15.9
8000	180	188	22.8

Table 11. Tree construction time (2).

#Attributes	X-monotone	Rectilinear	Guillotine
4	58.7	64.4	17.7
6	85.8	91.8	19.3
8	128	136	20.8
10	180	186	22.3

convex (spline2) regions, and guillotine cuts. The result shows that tree construction time is a little more than our scale estimation, because trees from larger datasets tend to become bigger.

Table 11, on the other hand, compares the performance using datasets with different numbers of attributes. We used 8000 records from the "add10" dataset, and constructed trees using N numerical attributes in the dataset. Though the time complexity for finding the optimal region in each split is almost linear of the combinations (permutations) of attributes, the overall tree construction time does seems to increase sub-linearly relative to the number of combinations. One of the main reasons is that the cost of computing the optimal region is not dominant if there are few attributes.

5. Conclusion and discussion

Experiments using diverse datasets confirmed that regression trees with region splitting appear to be more accurate and smaller than conventional ones. However, in order to use region splitting, we have to spend additional computation time that is proportional to the number of combinations of numeric conditional attributes. In the experiments, region splitting trees generally reduced error ratio by around 10 to 20%. One case of which achieves 48% reduction. Those reductions suggest the advantages of region splitting trees. In many applications, the improvements will be worth the computational cost if there are not too many numeric attributes.

Another important advantage of region splitting is comprehensibility of correlations among conditional attributes, which we did not evaluate objectively. Though the optimal region in each node of a regression tree itself is complicated, arguably we can easily understand two-dimensional visual representations like that of figure 1.

By introducing region splitting in regression trees, we have to consider additional parameters: record density, type of regions, and type of splines. Choice of those parameters affects construction time and prediction accuracy of the trees.

As for the types of splines, the computational costs of the three types, including no interpolation, are small and negligible. The spline2 achieved better results than the others in almost all datasets and experimental conditions, though the experiments to date are only suggestive of the advantage of spline2.

As for the other experimental conditions, there are tradeoffs. If we use lower record density, we can find finer pixel regions. However, it takes much time and it tends to overfit if record density is too fine or if the number of records is too small. According to the experimental results, we should use a record density of more than five. Since the number of records becomes smaller in the lower nodes of a tree, the pixel grid becomes too coarse with less than five by five. In such cases, we abandoned region splitting and used conventional guillotine cutting, though an alternative might be single-variable regressions as described by Lubinsky (Lubinsky, 1996).

An x-monotone region can express more variety of shapes than a rectilinear region. However, it often overfits, especially when we use a low record density. In the experiments, because of overfitting, the accuracy results of x-monotone regions is worse than that of rectilinear regions. If we use an x-monotone region, we should use a record density of more than ten. A notable advantage of using x-monotone regions is their fast computation time. If the number of records becomes huge, say more than a million, it is better to use x-monotone regions.

For classification problems, we applied similar region splitting for constructing decision trees in Morimoto et al. (1997). Though trees with region splitting also reduce classification errors, those reductions are not so large compared with the regression cases. The spline interpolations were not so effective in the classification problems, which was one of the reasons for the differences. In k-class classification problems, we have to consider a k-dimensional guided branch and bound search, while we can handle regression problems in two-dimensional plane. In such cases, the time and space complexities for classification become large, and we have to compromise on optimality.

In this paper, we proposed region splitting in regression trees and presented the methods and their effectiveness. As discussed above, there are issues that have to be explored, such as parameter adjustment, handling tables with many numerical attributes, and handling tables with a small number of records.

Acknowledgments

This work was partly supported by the Advanced Software Enrichment Project of the Information-Technology Promotion Agency, Japan. We thank Professor Douglas H. Fisher and anonymous reviewers for their suggestions and detailed comments that helped improve this paper.

References

Asano, T., Chen, D., Katoh, N., & Tokuyama, T. (1996). Polynomial-time solutions to image segmentations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (pp. 104–113).

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Brodley, C. E. & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning, 19*, 45–77.

Fukuda, T., Morimoto, Y., Morishita, S., & Tokuyama, T. (1996). Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (pp. 13–23).

Garey, M. R. & Johnson, D. S. (1979). *Computer and Intractability. A Guide to NP-Completeness*. San Francisco, CA: W. H. Freeman.

Gehrke, J. E., Ramakrishnan, R., & Ganti, V. (1998). RainForest—A framework for fast decision tree construction of large datasets. In *Proceedings of the International Conference on Very Large Data Bases* (pp. 416–427).

Hyafil, L. & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters, 5*, 15–17.

Ittner, A. & Schlosser, M. (1996). Non-linear decision trees—NDT. In *Proceedings of the International Conference on Machine Learning* (pp. 252–258).

Lubinsky, D. (1996). Tree structured interpretable regression. *Learning from Data: AI and Statistics V* (pp. 387–398).

Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In *Proceedings of the International Conference on Extending Database Technology* (Vol. 1057, pp. 18–32). Berlin: Springer.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4*, 227–243.

Morimoto, Y., Fukuda, T., Morishita, S., & Tokuyama, T. (1997). Implementation and evaluation of decision trees with range and region splitting. *Constraints, an International Journal, 2* (3/4), 401–427.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research, 2*, 1–32.

Piatetsky-Shapiro, G. & Frawley, W. J. (Eds.) (1991). *Knowledge Discovery in Databases*. Menlo Park, CA: AAAI Press.

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

Shafer, J. C., Agrawal, R., & Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the International Conference on Very Large Data Bases* (pp. 544–555).

Yoda, K., Fukuda, T., Morimoto, Y., Morishita, S., & Tokuyama, T. (1997). Computing optimized rectilinear regions for association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining* (pp. 96–103).

Received April 7, 1998
Revised January 17, 2001
Accepted January 17, 2001
Final manuscript February 19, 2001