# An Exact Dynamic Programming Algorithm for Regression Trees

Jacqueline J. Meulman, Elise Dusseldorp and Bart Jan van Os *

**Achtergrond:** Mijn belangstelling voor algoritmes voor combinatorische optimalisatie komt op de eerste plaats door mijn jarenlange samenwerking met Lawrence J. Hubert (University of Illinois at Urbana-Champaign) en Phipps Arabie (Rutgers University, Newark). Toen ik in 1994 een NWO-PIONIER subsidie kreeg, werd Bart Jan van Os een van mijn promovendi. Omdat hij een uitzonderlijk talent voor het ontwerpen en implementeren van algoritmes bleek te hebben, richtte het onderwerp van zijn proefschrift zich al gauw op globale optimalisatie d.m.v. dynamisch programmeren. Elise Dusseldorp, ook uit het PIONIER-project, promoveerde op het onderwerp regressiebomen. Omdat ik mijzelf met toepassingen van ensembles van kleine regressiebomen had beziggehouden, kwam het idee om gezamenlijk aan globaal optimale regressiebomen te werken bijna als vanzelf.

Tijdens het schrijven van zijn proefschrift heb ik Bart Jan aangeraden om een kursus bij Jan Karel Lenstra, toen nog in Eindhoven, te volgen. Jan Karel vertelde mij bij een latere gelegenheid dat hij nogal verbaasd was geweest om een promovendus uit de Faculteit Sociale Wetenschappen te Leiden in zijn collegezaal te zien. Nu vormden wij inderdaad een bijzondere groep binnen de FSW: de Facultaire vakgroep Datatheorie (het geesteskind van John P. van de Geer, de vader van Sara). Nadat een verregaande integratie van de Datatheoriegroep binnen een FSW departement mislukt was, kwam ik zelf tot mijn grote vreugde bij het Mathematisch Instituut terecht. Het artikel over optimale regressiebomen was echter nooit afgekomen. Na de uitnodiging om een bijdrage te leveren aan dit Liber Amicorum, kreeg ik het idee om het artikel ter ere van Jan Karel af te maken. Alleen bleek het toen als bijdrage aan het Liber veel te lang te zijn geworden. Daarom hier nu een sterk ingekorte impressie van ons werk, dat hopelijk te zijner tijd elders in volledige vorm gepubliceerd zal worden (maar in ieder geval al te lezen is op www.math.leidenuniv.nl/∼jmeulman). Wij danken hierbij Jan Karel voor zijn inspiratie.

*Jacqueline Meulman is Professor of Applied Statistics, Department of Mathematics, Leiden University, The Netherlands (e-mail: jmeulman@math.leidenuniv.nl), Elise Dusseldorp is Statistician, TNO Quality of Life Research, Leiden, The Netherlands (e-mail: elise.dusseldorp@tno.nl) and Bart Jan van Os is Statistician, 4os Consultancy, The Netherlands (e-mail: advies@4os.nl). This paper is based on work first presented in the PhD thesis Dynamic Programming for partitioning in multivariate data analysis by Bart Jan van Os, Leiden 2001, under supervision of Jacqueline J. Meulman and Lawrence J. Hubert.

## 1 Introduction

Although trees were introduced for the first time by Morgan and Sonquist (1963) in a method called Automatic Interaction Detection, the methodology was only fully developed in Breiman, Friedman, Olshen, and Stone (1984). Their CART algorithm complemented regression trees for the prediction of a continuous outcome variable, with classification trees that deal with a categorical (nominal) outcome variable. Since then a number of different methods and software packages have become available (for an overview, see Hand (1997)), especially after trees obtained renewed interest by research in data mining methods. The methodology has become popular for several reasons: trees automatically select predictors and find complicated interaction effects in the presence of noise variables, they deal with missing data without having to resort to imputation methods, and they can handle a large number of predictors of different measurement level (numeric, ordinal, binary and categorical variables are all handled with equal ease). Trees are immune to the effects of extreme outliers among the predictor variables, and are invariant to the scaling of predictors or any monotone transformation of them. Finally, a fitted tree mimics the way a hierarchically nested set of decision rules can be represented, and therefore has a clear interpretation.

Despite their success, trees are also known to have a major drawback: their limited prediction accuracy in applications, due to their instability. Instability means that a small change in the data can result in a very different tree, and the latter makes the interpretability unreliable. Because of the hierarchical nature of the process, the effect of an error in the top split is propagated down to all of the splits below it, and this constitutes an inherent instability (Hastie, Tibshirani, and Friedman (2001), p. 274). Therefore various approaches have tried to extend greedy algorithms, with successful advancements obtained by combining multiple tree structures in ensembles, obtaining a stronger predictor or classifier. Examples are Boosting (Freund & Schapire, 1996; Drucker & Cortes, 1996; Friedman, 2001), Bagging (Breiman, 1996), Random Forests (Breiman, 2001) and Rule Ensembles (Friedman & Popescu, 2008), which are competitive with other predictive learning methods such as Support Vector Machines (Vapnik, 1996), Kernel methods and local regression (see Loader, 1999), and Neural Networks (see Ripley, 1996).

Unfortunately, the result of an ensemble method is deprived of one of the most important advantages of a single tree, i.e., its high interpretability. At the expense of higher prediction accuracy, we lose the representation of the prediction rules in a simple tree structure. Because prediction using ensemble methods works more or less as a black box, considerable efforts have been invested to obtain secondary tools for interpretation, such as the importance of a variable by looking at partial dependence (for example, Friedman 2001). However, with respect to interpretability, the overview by looking at many partial dependence plots cannot compete with looking at a single tree. Consequently, adapting greedy tree methods remains an active domain of research, also because of their ability to handle large datasets that arise in data mining. Research in this area includes the introduction of more complex splits that combine several predictors through a discriminant analysis in each node (Loh & Vanichesetakul, 1988), finding more advanced split rules to overcome variable selection bias (Loh, 2002), and combining the tree model with standard regression models (Alexander & Grimshaw, 1996; Dusseldorp & Meulman, 2001, 2004; Dusseldorp, Conversano, & Van Os, 2010).

The present paper aims at strengthening tree methods at their heart, i.e., at the way trees are constructed (fitted). A major way to remove the inherent instability, and still to obtain a single tree representation, is to use an algorithm that finds a *globally optimal* tree. However, from the early days on, simultaneous optimization over all possible splits in the complete tree structure has been considered computationally infeasible. Nowadays, the problem is believed to be most likely intractable, because parts of the problem and the related problem of constructing minimum size regression trees from decision tables are proven to be NP-complete (Hyafil & Rivest, 1976; Naumov, 1994). The effort of finding a globally optimal tree immediately raises a crucial question: is optimizing the performance of prediction rules on the training data actually worth the effort, or does it induce over-fitting so that the expected prediction error (obtained for the test data) actually increases? We believe that the latter is not the case, because the algorithm aims at improving the accuracy without increasing the complexity, or possibly even decreases the complexity.

## 2 Regression Trees

In regression problems, one has a system consisting of a random "outcome", "response", or "dependent" variable $Y$ and a set of random "explanatory" ,"predictor", or "independent" variables $X = \{X_1, \cdots, X_J\}$, where $J$ denotes the number of predictors. The problem defines a "training" sample, $\{y_i, \mathbf{x}_i\}_1^N$ of known values for $Y$ and $\mathbf{X}$, where $(y_i, \mathbf{x}_i)$ links the predictor variables of the $i$th object with the $i$th value of the outcome variable, and where $i = 1, \ldots, N$. The space spanned by the predictor variables, denoted by $\mathbf{X}$, is called the *measurement space*. The regression tree problem consists of finding a set of rules that partition the measurement space into regions such that within each region $R_k$ we are able to predict a measurement $\hat{y}_i$ as close as possible to $y_i$ for all objects that are included into that region (for an example, see Hastie et al. (2001), p. 268). The measurement space representation has the advantage that it provides a concise description of the regression tree problem as a partitioning task, and immediately suggests relations to other problems that were solved globally, such as hierarchical clustering (Hubert, Arabie, & Meulman, 2001; Van Os, 2001; Van Os & Meulman, submitted). Consequently, the measurement space representation will be used to present the global optimization approach and the proposed algorithm.

## 2.1 Formal Definition of the Optimal Tree Problem

In the sequel we will show how regression trees can be formulated as a combinatorial optimization problem. Without loss of generality we will restrict ourselves to binary trees.

Each tree results in a partition $\mathcal{P}$ of the measurement space $\mathbf{X}$ into a number of $|\mathcal{P}|$ regions $R_k \in \mathcal{P}$ that correspond to the induced terminal nodes. Let $\Omega_d(\mathbf{X})$ denote the set of all admissible partitions of the measurement space $\mathbf{X}$ with binary trees of at most $K$ regions (terminal nodes) and depth $d$, where $d$ denotes the maximum number of layers from the top to any terminal node in the tree. Note that the definition of $\Omega_d(\mathbf{X})$ may include additional admissibility restrictions that are widely applied in tree methods, such as the required minimum number of observations in a terminal node, or a minimum impurity criterion that stops further splitting if the value of the loss function (that measures the discrepancy between the training data and the representation) is close (or equal) to zero.

We now can formulate regression trees as special cases of the mathematical problem:

$$\begin{cases} f(\mathbf{y}, \mathbf{X}) = \underset{\mathcal{P}}{\operatorname{opt}} \sum_{R_k \in \mathcal{P}} h(\mathbf{y}, \mathbf{X}, R_k) \\ \text{subject to} \\ \mathcal{P} \in \Omega_d(\mathbf{X}), \\ d \leq D, \\ |\mathcal{P}| \leq K. \end{cases} \tag{1}$$

The *impurity* function $h(\cdot)$ can be defined in many ways without violating the Dynamic Programming principles explained below. For regression trees that minimize the sum of squared error by estimating a constant $a_k$ in each region $R_k$, $h(\cdot)$ is defined by

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i (y_i - a_k)^2 I(\mathbf{x}_i \in R_k), \tag{2}$$

but $h(\cdot)$ also includes other loss functions such as absolute deviations

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i |y_i - a_k| I(\mathbf{x}_i \in R_k), \tag{3}$$

or even so-called Treed Regression (Alexander & Grimshaw, 1996) that estimates a *simple* linear regression model in each region by defining impurity as

$$h(\mathbf{y}, \mathbf{X}, R_k) = \sum_i (y_i - (a_k + \mathbf{b}_k'\mathbf{x}))^2 I(\mathbf{x}_i \in R_k), \tag{4}$$

where $\mathbf{b}_k$ only has one non-zero element corresponding to the best linear predictor in that region, excluding predictors that are used for splits up to the particular terminal node. Note that in all these cases the required additional parameters can be estimated *independently* from other regions.

The mathematical problem formulation also explicitly includes two types of size constraints. The first type of constraint ($d \leq D$) imposes the maximum depth $D$ on the tree, and corresponds to the maximum order of interaction between predictors (Friedman, 2001). Let an *np*-split (a new predictor split) be a split that is defined on a predictor that has not been used in previous splits, if existing. The maximum order of interaction is then equal to the largest number of np-splits from the top to any terminal node. The second type of constraint ($|\mathcal{P}| \leq K$) imposes a maximum $K$ on the number of terminal nodes ($|\mathcal{P}|$ in the tree. These two restrictions are obviously related by $K \leq 2^D$, and only one of the two may be in effect for a given problem. Both constraints define the admissible *maximum* size rather than the required maximum size, because additional admissibility conditions, such as the minimum number of observations in a terminal node, may actually prevent larger trees in some cases. Because in general the best prediction accuracy obtained by the least complexity of the tree is preferred, the smallest tree that optimizes (1) under the size constraints is accepted.

Note that in our approach these size constraints do not necessarily have to be chosen a priori. The mathematical problem (1) is solved for a series of values for $K$ and $D$, and fortunately, the algorithm introduced below will provide such a series of solutions with relatively little additional effort.

## 3 The Dynamic Programming recursion

To formulate our Dynamic Programming algorithm some more definitions are needed that take into account the hierarchical nesting of the splits. Let $v$ be any node (including the root node) of a regression tree $T(R)$ of the complete measurement space $R$, and let $R'$ and $T(R')$ denote the region ($R' \subseteq R$) and the (sub)tree associated with $v$; let $l(R', s, j)$ and $r(R', s, j)$ be a pair of left and right functions that split the region $R'$ into a left and right measurement space using split $s \in \mathcal{S}_j(R')$, where $\mathcal{S}_j(R')$ denotes the set of admissible splits of $R'$ by predictor $X_j$. Furthermore, let $h(R')$ be a shorthand notation for $h(\mathbf{y}, \mathbf{X}, R')$, and let $t(R')$ denote the value of the objective function for obtaining a (sub)tree $T(R')$ for region $R'$, such that

$$t(R') = \sum_{R_k \in \mathcal{P}(R')} h(R_k). \tag{5}$$

Finally, let $T^*(R')$ be an *optimal* (sub)tree and $t_d(R')$ the associated (optimal) value of the objective function. We have the following *principle of optimality* (Bellman, 1957):

**Proposition 1** *Any subtree $T(R')$ starting at some node $v$ of an optimal tree $T^*(R)$ is itself an optimal tree of the measurement space $R'$ associated with its root node $v$.*

PROOF. The objective function can be split into the sum of two parts

$$t^*(R) = \sum_{R_k \in \mathcal{P}(R)} h(R_k) = \sum_{R_{k_1} \in \mathcal{P}(R')} h(R_{k_1}) + \sum_{R_{k_2} \in \mathcal{P}(R-R')} h(R_{k_2}), \tag{6}$$

where the first part contains the regions that belong to the subtree $T(R')$, and the other part contains all other regions. Hence, we can always replace any subtree of an optimal tree with a better subtree if that would exist, but this would contradict the optimality of the original tree $T$.
This particular splitting of the objective function is often referred to as the *separability* of the objective function. In this case, it stems from the fact that we can evaluate the part of the objective function for the subtree independently from the way in which the tree partitions the rest of the measurement space.

This principle quite naturally leads to a recursion for optimal regression trees. In fact, the transition from a recursion to a tree and vice versa is so transparent that trees are often used to explain DP algorithms. Moreover, these type of recursions for trees are well-known in computer science, where the classical example is the use of DP for the minimal size "Optimal Binary Search Tree", a problem that is not NP-hard, and were DP is very efficient. In contrast, DP for NP-hard problems is often not considered because of the explosion of computer resources needed for increasing problem sizes, with the major exception of Hubert et al. (2001), Hubert, Arabie, and Meulman (2006), who present DP solutions (and software) for finding exact solutions to a number of combinatorial data analysis problems.

In formulating the recursion, however, we have to take into account the size restrictions of the mathematical problem (1) that are implicit in this principle of optimality. In other words, the principle of optimality only holds if the size constraints for the subtree conform to the overall size constraints of the complete tree. To define practical recursions that can readily be converted into algorithms, it is more convenient to explicitly include the size constraints. For maximum-depth trees, where only the depth-constraint is imposed, we have the following recursion:

$$\begin{cases} t^*_d(R') = & \underset{s,j}{\text{opt}} \left[t^*_{d-1}(l(R',s,j)) + t^*_{d-1}(r(R',s,j)\right] \\ & \text{subject to } s \in \mathcal{S}_j(R'); j = 1,\ldots,J; \\ t^*_d(R') = & h(R') \\ & \text{if } d = 1 \text{ or} \\ & \text{other admissibility restrictions prevent further splitting of } R'. \end{cases} \tag{7}$$

In words, the process of finding an optimal tree can be described by: i. search through all possible splits at the current node; ii. for each split, find independently an optimal subtree at the left and the right node; iii. a. if a node is allowed to be split further, the search process is recursively started for this new node; iv. b. if a node is an end node, we simply use the objective function value for this node; v. the objective function value for the current split at the current node is the sum of the objective function values of the optimal subtrees; vi the split that results in the best combination of subtrees gives the optimal (sub)tree at the current node and is returned; vii if the current node is the root node, we have found the complete optimal tree; otherwise we return to the previous level in the search.

When we also impose the overall size constraint $|\mathcal{P}| \leq K$, the recursion becomes more elaborate, because we generally do not know in advance the size of the left and the right subtree for any split:

$$\begin{cases} t^*_{dk}(R') = & \underset{s,j,k_l}{\text{opt}} \left[ t^*_{d-1,k_l}(l(R',s,j)) + t^*_{d-1,k-k_l}(r(R',s,j) \right] \\ & \text{subject to } s \in \mathcal{S}_j(R'); j = 1, \ldots, J; k_l = 1, \ldots, k-1; \\ t^*_{dk}(R') = & h(R') \\ & \text{ if } d = 1 \text{ or } k = 1 \text{ or} \\ & \text{other admissibility restrictions prevent further splitting of } R'. \end{cases} \tag{8}$$

The search process is now extended to not only include an optimal choice of the split $s$ and the predictor $j$, but also the optimal split of the size constraint $k$ at the node into two size constraints $k_l$ and $k_r, k = k_l + k_r$ for the left and right subtrees. Consequently, imposing the size constraint $|\mathcal{P}| \leq K$ additionally to imposing the depth constraint increases the workload of completely searching the recursion. For an algorithm that finds optimal trees for a range of size constraints, however, the recursion can be reformulated such that the number of possible nodes visited for finding a range of depth-constrained optimal trees is of the same order of magnitude as for finding a range of both depth and size constrained trees.

## 4 Algorithm Implementation

Given the recursions (7) and (8), we can immediately formulate recursive algorithms. For example, Algorithm 4.1 gives a DP algorithm for finding an optimal maximum-depth tree. This algorithm explicitly uses a heap to store and retrieve previously searched nodes, and only gives the (optimal) objective function value of an optimal tree. To retrieve the optimal tree itself, the tuple $(R', (s,j))$ that corresponds to the optimal split must be stored in the heap as well. When the algorithm is finished, a simple recursive routine can restore the optimal tree by retrieving all optimal $(R', (s,j))$ from the heap, starting at the root node for $R'$, applying the split $(s,j)$ giving $R'^{(l)}$ and $R'^{(r)}$, retrieving the optimal splits for those measurement spaces, and so forth.

The current algorithm constrains the tree size by maximum depth. For finding optimal trees with a maximum number of terminal nodes $K$, we have to extend the algorithm. Taken the above recursion, a straightforward implementation is to induce an additional loop that circles over all possible sizes of

**Algorithm 4.1** $h \leftarrow$ OptimalTree$(R', d)$
Find the minimal objective function value $h$ of the best tree, springing from a node defined by region $R'$, with depth $d$ (recursive)

```
 1: if $\exists(R', h)$ then
 2:     retrieve $(R', h)$
 3: else if $(d = 1) \vee (\mathcal{S}_j(R') = \emptyset, \forall j) \vee (R'$ is homogeneous enough$)$ then
 4:     $h \leftarrow h(R')$
 5: else
 6:     $h \leftarrow \inf$
 7:     for $j = 1$ to $J$ do
 8:         for $s \in \mathcal{S}_j(R')$ do
 9:             $h^{(l)} \leftarrow$ OptimalTree$(l(R', s, j), d - 1)$
10:             $h^{(r)} \leftarrow$ OptimalTree$(r(R', s, j), d - 1)$
11:             $h \leftarrow \min(h, h^{(l)} + h^{(r)})$
12:         end for
13:     end for
14:     store $(R', h)$
15: end if
Ensure: $\exists h$
```

(sub)trees before stepping into a lower level. However, a more efficient approach is given by Algorithm 4.2 that extends the search by simultaneously searching at each node for all optimal trees of size one up to $K - 1$ rather than *the* optimal tree. To do so, we have to search for all optimal combinations of subtrees such that $2 \leq |P(R'^{(l)})| + |P(R'^{(r)})| \leq K$, and store the optimal combinations for all sizes $1, \ldots, K$. (The 'optimal combination' of size one is no split). The practical details as well as the limits for using a (partial heap) are discussed in the full paper. Effective implementations have so far mostly not implemented a heap, but instead simply fully exploit the recursive algorithm.

## 5 Discussion

The present paper proposes an exact Dynamic Programming algorithm for finding optimal regression trees, where optimality is defined as the quality of the predictions at the leaves. DP is often dismissed a priori as a candidate for such hard problems, because of the enormous computational resources involved. Practical limitations are indeed imminent, but due to the particular implementation, our algorithm can find (guaranteed) optimal trees for sample sizes of about 500 observations and 15 predictors, with a maximum size of the tree, the *depth*, of seven. Note that these numbers applied at the time the current algorithm was implemented, which was some time ago, and a depth of seven is usually more than sufficient. So ExactTree can find sufficiently large optimal trees for moderately sized datasets, and thus it can deal with nontrivial statis-

**Algorithm 4.2** $H \leftarrow$OptimalSCTrees$(R', d, K)$
Find the minimal objective function values $H = \{h_1, \ldots, h_K\}$ of all Size Constrained trees with number of terminal nodes not exceeding $2, \ldots, K$, springing from a node defined by region $R'$, with depth $d$ (recursive)

```
1: if $\exists(R', H)$ then
2:     retrieve $(R', H)$
3: else if $(d = 1) \vee (\mathcal{S}_j(R') = \emptyset, \forall j) \vee (R'$ is homogeneous enough$)$ then
4:     $h_k \leftarrow h(R'), k = 1, \ldots, K$
5: else
6:     $H \leftarrow \inf$
7:     $h_1 \leftarrow h(R')$
8:     for $j = 1$ to $J$ do
9:         for $s \in \mathcal{S}_j(R')$ do
10:            $H^{(l)} \leftarrow$ OptimalSCTrees$(l(R', s, j), d - 1, K - 1)$
11:            $H^{(r)} \leftarrow$ OptimalSCTrees$(r(R', s, j), d - 1, K - 1)$
12:            $h_k \leftarrow \min(h_k, h^{(l)}_{k'} + h^{(r)}_{k-k'}), k = 2, \ldots, K,$ and $k' = 1, \ldots, k$
13:        end for
14:    end for
15:    $h_k \leftarrow \min(h_{k'}), k = 2, \ldots, K,$ and $k' = 1, \ldots, k$
16:    store $(R', H)$
17: end if
Ensure: $\exists H$
```

tical applications in a lot of domains. Moreover, ExactTree can do this while using a plethora of optimality criteria that are beyond the reach of alternative strategies such as linear programming and branch and bound; it uses a simple interface and does not require special expertise of the user (a Matlab version of the software is available from the third Author).

Our applications so far have shown that using the ExactTree algorithm indeed improves upon the prediction of greedy methods; compared to the greedy algorithm, ExactTree both finds smaller trees with similar prediction error, and larger trees with smaller prediction error (while having a smaller standard error of the prediction error). These results might indicate that using an exact algorithm within the bounds of size constraints on the tree structure, indeed leads to a more stable and accurate estimation of the tree structure without over-fitting on the training data. Moreover, restricting the search space of splits for each variable by optimally digitizing the predictors into a moderate number of categories appears to improve the stability of the prediction (and this seems to be also true for greedy methods). Further study to confirm these results is obviously needed.

ExactTree cannot deal with the size of datasets that are common in large-scale data mining, nor can it be used to replace automatic variable selection in the presence of a large number of variables. In the latter case, however, it is worthwhile to use ExactTree to estimate an optimal tree for given predictors that were selected by another method. Another application of ExactTree is as a benchmark tool, for truly evaluating the performance of more elaborate heuristic approaches as compared to greedy methods. Finally, an obvious application of ExactTree would be as a part of a so-called Boosting Algorithm, such as MART, where the ensemble is made up by many *small* trees, to obtain a really strong predictor.

## References

Alexander, W. P., & Grimshaw, S. D. (1996). Treed regression. *Journal of Computational and Graphical Statistics.*, *5*, 156–175.

Bellman, R. E. (1957). *Dynamic programming.* Princeton, NJ: Princeton University Press.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *26*, 123–140.

Breiman, L. (2001). Random forrest. *Machine Learning*, *45*, 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth.

Drucker, H., & Cortes, C. (1996). Boosting decision trees. In D. Touretzky, M. Mozer, & M. Hasselmo (Eds.), *Advances in neural information processing systems 8* (pp. 479–485). MIT Press.

Dusseldorp, E., Conversano, C., & Van Os, B. J. (2010). Combining an additive and tree-based regression model simultaneously: Stima. *Journal of Computational and Graphical and Statistics*, *19*, 514–530.

Dusseldorp, E., & Meulman, J. J. (2001). Prediction in medicine by integrating regression trees into regression analysis with optimal scaling. *Methods of Information in Medicine*, *5*, 403–409.

Dusseldorp, E., & Meulman, J. J. (2004). The regression trunk approach to discover treatment covariate interaction. *Psychometrika*, *69*, 355–374.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proc. 13th international conference on machine learning* (pp. 148–156). Morgan Kaufman.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Friedman, J. H., & Popescu, B. E. (2008). Predictive learning via rule ensembles. *Annals of Applied Statistics*, *2*, 916–954.

Hand, D. J. (1997). *Construction and assessment of classification rules.* Chichester: Wiley.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction.* New York: Springer.

Hubert, L. J., Arabie, P., & Meulman, J. J. (2001). *Combinatorial data analysis: Optimization by dynamic programming.* Philadelphia: SIAM.

Hubert, L. J., Arabie, P., & Meulman, J. J. (2006). *The structural representation of proximity matrices with matlab.* Philadelphia: ASA-SIAM Series on Statistics and Applied Probability.

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

Loader, C. (1999). *Local regression and likelyhood.* New York: Springer-Verlag.

Loh, W.-Y. (2002). Regression trees with and unbiased variable selection and interaction detection. *Statistica Sinica*, *12*, 361–386.

Loh, W.-Y., & Vanichesetakul, N. (1988). Tree structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, *83*, 715–725.

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data,and a proposal. *Journal of the American Statistical Association.*, *58*, 415–434.

Naumov, G. E. (1994). Np-completeness of problems of construction of optimal decision trees. *Soviet Physics, Doklady*, *36*, 270–271.

Ripley, B. D. (1996). *Pattern recognition and neural networks.* Cambridge: Cambridge University Press.

Van Os, B. J. (2001). *Dynamic programming for partitioning in multivariate data analysis.* Unpublished doctoral dissertation, Leiden University.

Van Os, B. J., & Meulman, J. J. (submitted). *Dynamic programming for globally optimal hierarchical cluster analysis.*

Vapnik, V. (1996). *The nature of statistical learning theory.* New York: Springer-Verlag.