Logic Regression

Ingo RUCZINSKI, Charles KOOPERBERG, and Michael LEBLANC

Logic regression is an adaptive regression methodology that attempts to construct predictors as Boolean combinations of binary covariates. In many regression problems a model is developed that relates the main effects (the predictors or transformations thereof) to the response, while interactions are usually kept simple (two- to three-way interactions at most). Often, especially when all predictors are binary, the interaction between many predictors may be what causes the differences in response. This issue arises, for example, in the analysis of SNP microarray data or in some data mining problems. In the proposed methodology, given a set of binary predictors we create new predictors such as "X_1, X_2, X_3, and X_4 are true," or "X_5 or X_6 but not X_7 are true." In more specific terms: we try to fit regression models of the form $g(E[Y]) = b_0 + b_1 L_1 + \cdots + b_n L_n$, where L_j is any Boolean expression of the predictors. The L_j and b_j are estimated simultaneously using a simulated annealing algorithm. This article discusses how to fit logic regression models, how to carry out model selection for these models, and gives some examples.

Key Words: Adaptive model selection; Boolean logic; Binary variables; Interactions; Simulated annealing; SNP data.

1. INTRODUCTION

In most regression problems a model is developed that only relates the main effects (the predictors or transformations thereof) to the response. Although interactions between predictors are sometimes considered as well, those interactions are usually kept simple (two- to three-way interactions at most). But often, especially when all predictors are binary, the interaction of many predictors is what causes the differences in response. For example, Lucek and Ott (1997) were concerned about analyzing the relationship between disease loci and complex traits. In the introduction of their article, Lucek and Ott recognized the importance of interactions between loci and potential shortcomings of methods that do not take those interactions appropriately into account:

Ingo Ruczinski is Assistant Professor, Department of Biostatistics, Bloomberg School of Public Health, Johns Hopkins University, 615 N Wolfe Street, Baltimore MD 21205-2179 (E-mail: ingo@jhu.edu). Charles Kooperberg and Michael LeBlanc are Members, Fred Hutchinson Cancer Research Center, Division of Public Health Sciences, P.O. Box 19024, Seattle WA 98109-1024 (E-mail: clk@fhcrc.org *and* mikel@crab.org).

Journal of Computational and Graphical Statistics, Volume 12, Number 3, Pages 475–511
DOI: 10.1198/1061860032238

> Current methods for analyzing complex traits include analyzing and localizing disease loci one at a time. However, complex traits can be caused by the interaction of many loci, each with varying effect.

The authors stated that, although finding those interactions is the most desirable solution to the problem, it seems to be infeasible.

> ... patterns of interactions between several loci, for example, disease phenotype caused by locus A and locus B, or A but not B, or A and (B or C), clearly make identification of the involved loci more difficult. While the simultaneous analysis of every single two-way pair of markers can be feasible, it becomes overwhelmingly computationally burdensome to analyze all 3-way, 4-way to N-way "and" patterns, "or" patterns, and combinations of loci.

The above is an example of the types of problems we are concerned about. Given a set of binary (true and false, 0 and 1, on and off, yes and no, ...) predictors X, we try to create new, better predictors for the response by considering combinations of those binary predictors. For example, if the response is binary as well (which is not a requirement for the method discussed in this article), we attempt to find decision rules such as "if $X_1, X_2, X_3,$ *and* X_4 are true," or "X_5 *or* X_6 *but not* X_7 are true," then the response is more likely to be in class 0. In other words, we try to find Boolean statements involving the binary predictors that enhance the prediction for the response. In the near future, one such example could arise from SNP microarray data (see Daly et al. 2001 for one possible application), where one is interested in finding an association between variations in DNA sequences and a disease outcome such as cancer. This article introduces the methodology we developed to find solutions to those kind of problems. Given the tight association with Boolean logic, we decided to call this methodology *logic regression.*

There is a wealth of approaches to building binary rules, decision trees, and decision rules in the computer science, machine learning and, to a lesser extent, statistics literature. Throughout this article we provide a literature review that also serves to highlight the differences between our methodology and other methods, which we hope will become clear in the following sections. To our knowledge, logic regression is the only methodology that is searching for Boolean combinations of predictors in the entire space of such combinations, while being completely embedded in a regression framework, where the quality of the models is determined by the respective objective functions of the regression class. In contrast to the methods discussed in the literature review, other models—such as the Cox proportional hazards model—can also be used for logic regression, as long as a scoring function can be defined. Making it computationally feasible to search through the entire space of models without compromising the desire for optimality, we think that logic regression models such as $Y = \beta_0 + \beta_1 \times [X_1 \text{ and } (X_2 \text{ or } X_3)] + \beta_2 \times [X_1 \text{ or } (X_4 \text{ or } X_5^c)]$ might be able to fill a void in the regression and classification methodology. Section 2 introduces the basics of logic regression. Sections 3 (search for best models) and 4 (model selection) explain how we find the best models. Section 5 illustrates these features in a case study using simulated data and data from the Cardiovascular Health Study, and describes the results obtained by analyzing some SNP array data used in the 12th Genetic Analysis Workshop. The article

concludes with a discussion (Section 6). Explanations of some technical details of logic regression have been deferred to the Appendix.

2. LOGIC REGRESSION

Before describing in the second part of this section which models we are interested in, we introduce some terminology that we will use throughout this article.

2.1 LOGIC EXPRESSIONS

We want to find combinations of binary variables that have high predictive power for the response variable. These combinations are Boolean logic expressions such as $L = (X_1 \wedge X_2) \vee X_3^c$; since the predictors are binary, any of those combinations of predictors will be binary as well. We assume familiarity with the basic concepts of Boolean logic. We closely follow Peter Wentworth's online tutorial "Boolean Logic and Circuits" (http://diablo.cs.ru.ac.za/func/bool/), which contains a thorough introduction to Boolean algebra.

- Values: The only two values that are used are 0 and 1 (true and false, on and off, yes and no, ...).
- Variables: Symbols such as X_1, X_2, X_3 are variables; they represent any of the two possible values.
- Operators: Operators combine the values and variables. There are three different operators: $\wedge$ (AND), $\vee$ (OR), ^{c} (NOT). X^c is called the conjugate of X.
- Expressions: The combination of values and variables with operators results in expressions. For example, $X_1 \wedge X_2^c$ is a logic (Boolean) expression built from two variables and two operators.
- Equations: An equation assigns a name to an expression. For example, using $L = X_1 \wedge X_2^c$ we can refer to the expression $X_1 \wedge X_2^c$ by simply stating L.

Using brackets, any Boolean expression can be generated by iteratively combining two variables, a variable and a Boolean expression, or two Boolean expressions, as shown in Equation (2.1):

$$(X_1 \wedge X_2^c) \wedge [(X_3 \wedge X_4) \vee (X_5 \wedge (X_3^c \vee X_6))]. \tag{2.1}$$

Equation (2.1) can be read as an "and" statement, generated from the Boolean expressions $X_1 \wedge X_2^c$ and $(X_3 \wedge X_4) \vee (X_5 \wedge (X_3^c \vee X_6))$. The latter can be understood as an "or" statement, generated from the Boolean expressions $X_3 \wedge X_4$ and $X_5 \wedge (X_3^c \vee X_6)$, and so on. Using the above interpretation of Boolean expressions enables us to represent any Boolean expression in a binary tree format, as shown in Figure 1. The evaluation of such a tree, as a logic statement for a particular case, occurs in a "bottom-up" fashion via recursive substitution. The prediction of the tree is whatever value appears in the root (see the definition below) after this procedure.

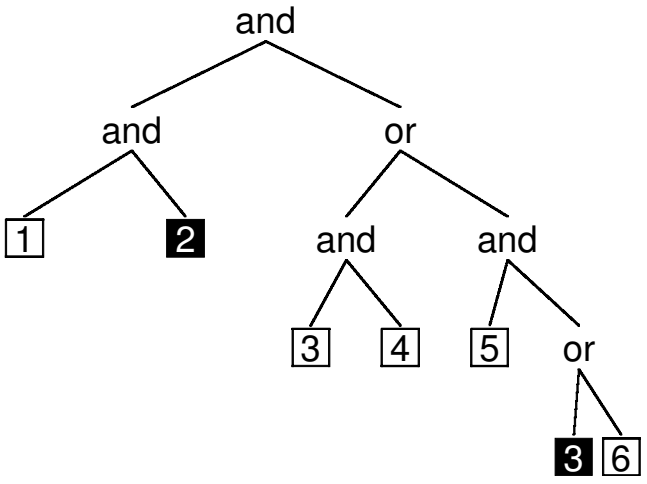


Figure 1. The logic tree representing the Boolean expression $(X_1 \wedge X_2^c) \wedge [(X_3 \wedge X_4) \vee (X_5 \wedge (X_3^c \vee X_6))]$. *For simplicity, only the index of the variable is shown. White letters on black background denote the conjugate of the variable.*

We use the following terminology and rules for logic trees [similar to the terminology used by Breiman, Friedman, Olshen, and Stone (1984) for decision trees]:

- The location for each element (variable, conjugate variable, operators $\wedge$ and $\vee$) in the tree is a knot.
- Each knot has either zero or two subknots.
- The two subknots of a knot are called its children, the knot itself is called the parent of the subknots. The subknots are each other's siblings.
- The knot that does not have a parent is called the root.
- The knots that do not have children are called leaves.
- Leaves can only be occupied by letters or conjugate letters (predictors), all other knots are operators ($\vee$'s, $\wedge$'s).

Note that since the representation of a Boolean expression is not unique, neither is the representation as a logic tree. For example, the Boolean expression in Equation (2.1) can also be written as

$$[(X_1 \wedge X_2^c) \wedge (X_3 \wedge X_4)] \ \vee \ [(X_1 \wedge X_2^c) \wedge (X_5 \wedge (X_3^c \vee X_6))].$$

Using the method described above to construct a logic tree from Boolean expressions, the result is a logic tree that looks more complex, but is equivalent to the tree in Figure 1 in the sense that both trees yield the same results for any set of predictors. See Ruczinski (2000) for a discussion of simplification of Boolean expressions in the context of logic regression.

In the above we described standard Boolean expressions, and introduced logic trees as a way to represent those Boolean expressions conveniently, because they simplify the implementation of the algorithm discussed in Section 3. Appendix A discusses the relation of those trees and other well-known constructs—for example, to Boolean expressions in

disjunctive normal form—that play a key role in many publications in the computer science and engineering literature (see, e.g., Fleisher, Tavel, and Yeager 1983), and to decision trees as introduced by Breiman et al. (1984) (see also Section 2.3).

2.2 LOGIC REGRESSION MODELS

Let $X_1, \ldots, X_k$ be binary predictors, and let Y be a response variable. In this article we try to fit regression models of the form

$$g(\mathsf{E}[Y]) = \beta_0 + \sum_{j=1}^{t} \beta_j L_j, \tag{2.2}$$

where L_j is a Boolean expression of the predictors X_i, such as $L_j = (X_2 \vee X_4^c) \wedge X_7$. We refer to those models as *logic models*. The above framework includes, for example, linear regression ($g(E[Y]) = E[Y]$) and logistic regression ($g(E[Y]) = \log(E[Y]/(1-E[Y]))$). For every model type we define a score function that reflects the "quality" of the model under consideration. For example, for linear regression the score could be the residual sum of squares and for logistic regression the score could be the binomial deviance. We try to find the Boolean expressions in (2.2) that minimize the scoring function associated with this model type, estimating the parameters β_j simultaneously with the search for the Boolean expressions L_j. In principle, other models such as classification models or the Cox proportional hazards model can be implemented as well, as long as a scoring function can be defined. We will come back to this in Section 6.

2.3 OTHER APPROACHES TO MODELING BINARY DATA

There are a large number of approaches to regression and classification problems in the machine learning, computer science, and statistics literature that are (sometimes) appropriate when many of the predictor variables are binary. This section discusses a number of those approaches. Boolean functions have played a key role in the machine learning and engineering literature in the past years, with an emphasis on Boolean terms in disjunctive normal form (e.g., Fleisher, Tavel, and Yeager 1983; Michalski, Mozetic, Hong, and Lavrac 1986; Apte and Weiss 1997; Hong 1997; Deshpande and Triantaphyllou 1998). Especially in the machine learning literature, the methods and algorithms using Boolean functions are generally based on either decision trees or decision rules. Among the former are some of the well-known algorithms by Quinlan, such as ID3 (Quinlan 1986), M5 (Quinlan 1992), and C4.5 (Quinlan 1993). CART (Breiman et al. 1984) and SLIQ (Mehta, Agrawal, and Rissanen 1996) also belong to that category. The rule-based methods include the AQ family (Michalski et al. 1986), the CN2 algorithm (Clark and Niblett 1989), the SWAP1 algorithms (Weiss and Indurkhya 1993a,b, 1995), RIPPER and SLIPPER (Cohen 1995; Cohen and Singer 1999), the system R^2 (Torgo 1995; Torgo and Gama 1996), GRASP (Deshpande and Triantaphyllou 1998), and CWS (Domingos 1996). Other approaches to finding optimal association rules have been recently proposed for the knowledge discovery and data

mining community by Bayardo and Agrawal (1999) and Webb (2000, 2001). Some authors proposed to combine some of the simpler, existing rule-based models into more complex classifiers (Liu, Hsu, Ma 1998; Meretakis and Wuthrich 1999). Also mentioned should be the very nice review article by Apte and Weiss (1997), which describes the use of decision trees and rule induction for some data mining examples.

Within each of those two categories (trees versus rules), the methods differ by the aim classification versus regression (CART and MARS are exceptions which work in both classification and regression settings). The vast majority of algorithms published in the literature are concerned only with classification. Among the previously mentioned are ID3, C4.5, CN2, SLIQ, RIPPER/SLIPPER, and SWAP1. A myriad of other algorithms or derivations of the former methods exist for a variety of applications; see, for example, Apte, Damerau, and Weiss (1994). Weiss and Indurkhya (1993b) introduced an extension to their SWAP1 algorithm that learns regression rules in the form of ordered disjunctive normal form (DNF) decision rules. SWAP1R deals with regression by transforming it into a classification problem. Torgo and Gama (1996) extended this idea by proposing a processor that works on almost every classification method to be used in a regression framework. Methods that do not transform regression into classification are R^2, which searches for logical conditions in IF THEN format to build piece-wise regression models, and M5, which constructs tree-based piece-wise linear models. These models, trees with linear regression components in the terminal nodes, are known in the statistical literature as treed models (Chipman, George, and McCulloch 2002). The commercial package CUBIST developed by Quinlan learns IF THEN rules, with linear models in the conclusion. An example of such a CUBIST rule is "IF $(x_1 > 5)$ & $(x_2 < 3)$ THEN $y = 1 + 2x_1 - 8x_3 + x_5$." There is a similarity between this algorithm and the tree-based algorithm M5, which has linear functional models in the terminal nodes.

Another noteworthy difference in objectives across the methods is the desired format of the solutions, and in particular their interpretability. Although some authors strongly argue for compact, easily interpretable rules (e.g., Clark and Niblett 1989; Quinlan 1993; Weiss and Indurkhya 1995; Cohen 1995), others emphasize that they care about accuracy and predictive power the most. This is in particular the case when Boolean neural networks are the method of choice to find logic rules (Gray and Michel 1992; Thimm and Fiesler 1996; Lucek and Ott 1997; Anthony 2001). A nice review article by Wnek and Michalski (1994) compares a decision tree learning method (C4.5), a rule-learning method (AQ15), a neural net trained by a backpropagation algorithm (BpNet) and a classifier system using a genetic algorithm (CFS) with respect to their predictive accuracy and simplicity of solutions.

Multivariate adaptive regression splines (MARS; Friedman 1991) is not a methodology intended for binary predictors, but rather a regression methodology to automatically detect interactions between smooth, nonlinear spline-transformed continuous predictors. Because of its high amount of adaptivity, we will see in Section 5.1 that even for classification problems with binary predictors, a problem that MARS was not designed for, it can still produce quite reasonable results.

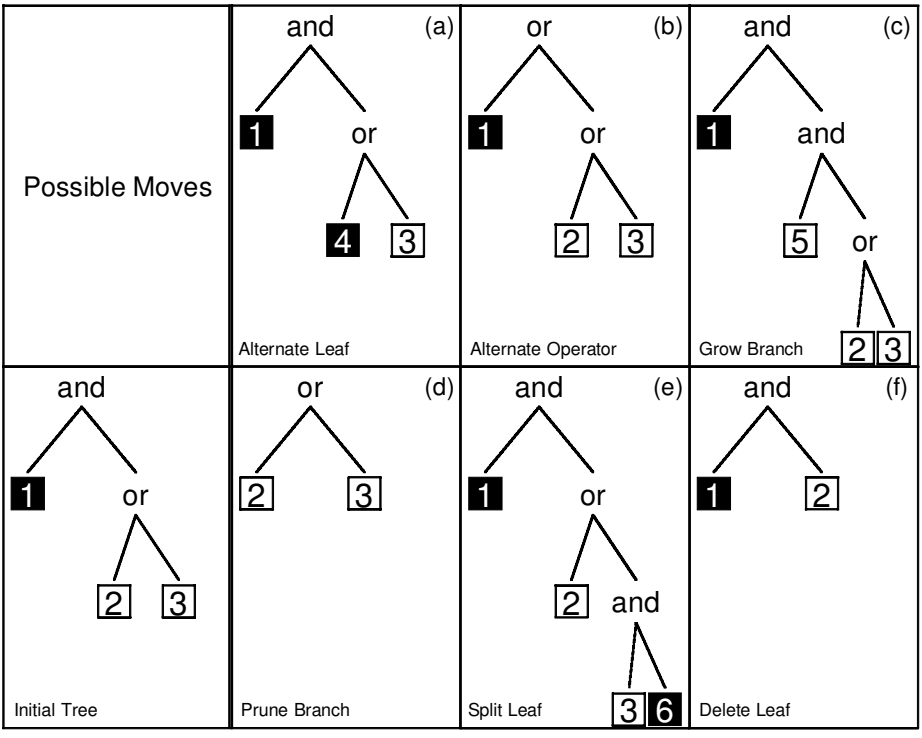


Figure 2. Permissible moves in the tree-growing process. The starting tree is in the panel in the lower left, the moves are illustrated in the panels (a)–(f). As in the previous figure, only the index of the variable is shown, and white letters on black background denote the conjugate of a variable.

3. SEARCH FOR BEST MODELS

The number of logic trees we can construct for a given set of predictors is huge, and there is no straightforward way to enlist all logic trees that yield different predictions. It is thus impossible to carry out an exhaustive evaluation of all different logic trees. Instead we use simulated annealing, a stochastic search algorithm (discussed in Section 3.3). We also implemented a greedy search algorithm which, together with an example, is shown in Section 3.2. But first we introduce the move set used in both search algorithms.

3.1 MOVING IN THE SEARCH SPACE

We define the neighbors of a logic tree to be those trees that can be reached from this logic tree by a single "move." We stress that each move has a counter move (i.e., a move to potentially get back from the new tree to the old tree), which is important for the underlying Markov chain theory in simulated annealing, discussed later. We allow the following moves:

- *Alternating a leaf:* We pick a leaf and replace it with another leaf at this position. For example, in Figure 2(a) the leaf X_2 from the initial tree has been replaced with

the leaf X_4^c. To avoid tautologies, if the sibling of a leaf is a leaf as well, the leaf cannot be replaced with its sibling or the complement of the sibling. It is clear that the counter move to alternating a leaf is by changing the replaced leaf back to what it was before the move (i.e., alternating the leaf again).

- *Changing operators:* Any $\wedge$ can be replaced by a $\vee$, and vice versa (e.g., the operator at the root in the initial tree in Figure 2 has been changed in Figure 2(b)). These two moves complement each other as move and counter move.
- *Growing and pruning:* At any knot that is not a leaf, we allow a new branch to grow. This is done by declaring the subtree starting at this knot to be the right side branch of the new subtree at this position, and the left side branch to be a leaf representing any predictor. These two side trees are connected by a $\wedge$ or $\vee$ at the location of the knot. For example, in the initial tree in Figure 2 we grew a branch in the knot that the "or" occupied (panel (c)). The counter move to growing is called pruning. A leaf is trimmed from the existing tree, and the subtree starting at the sibling of the trimmed leaf is "shifted" up to start at the parent of the trimmed leaf. This is illustrated in Figure 2(d), where the initial tree has been pruned at the "and."
- *Splitting and deleting:* Any leaf can be split by creating a sibling, and determining a parent for those two leaves. For example, in Figure 2(e) the leaf X_3 from the initial tree in Figure 2 has been split, with leaf X_6^c as its new sibling. The counter move is to delete a leaf in a pair of siblings that are both leaves, illustrated in Figure 2(f), where X_3 has been deleted from the initial tree.

Given this move set, a logic tree can be reached from any other logic tree in a finite number of moves, referred to irreducibility in Markov chain theory. This is true even if one, for example, omits some moves such as pruning and growing. In this sense, these moves are not absolutely necessary in the move set. But inclusion of those additional moves in the move set enhances the performance of the search algorithm. Section 4 discusses how to choose the optimal model from all candidates. For now we simply consider the maximum number of trees fixed. But note that if a model does not have the maximum of trees allowed, a permissible move is to add another tree with a single leaf. Vice versa, if a model has a tree with a single leaf, a permissible move is to delete this tree from the model.

3.2 GREEDY SEARCH

Similar to the search algorithm in CART (Breiman et al. 1984), a greedy algorithm can be used to search for "good" logic models. In the context of logic regression, the first step is to find the variable that, used as a single predictor, minimizes the scoring function (without loss of generalization, lower scores are better). After this predictor is found, its neighbors (states that can be reached by a single move from the given state) are investigated, and the new state is chosen as the state that

1. has a better score than the original state; and
2. has the best score among the considered neighbors.

If such a state does not exist, the greedy algorithm stops, otherwise the neighbors of the new state are examined and the next state is chosen according to the above described criterion. There is no guarantee that this algorithm finds the best scoring state possible. This does happen if the search gets "stuck," for example, if a better tree can be reached in two moves, but not in one move. Another potential problem is that in the presence of noise in the data it can happen that, even though the tree representing the correct underlying model has been reached in the search, there exist one or more additional moves that improve the score, and hence the final model over-fits the data. In contrast to the greedy search for CART, a greedy move for logic trees might actually result in a tree of lower or equal complexity (e.g., by deleting a leaf or changing an operator, respectively).

As an example, Figure 3 shows parts of the outcome of a greedy search for a logic tree in a classification problem on a simulated dataset. The data were generated by simulating 20 binary predictors, with a total of 1,000 cases each, with each value of each predictor being an independent sample from a Bernoulli random variable with probability $1/2$. The underlying Boolean equation was chosen to be

$$L = X_1 \wedge (X_2^c \vee X_3) \wedge [X_4 \vee (X_5^c \wedge (X_6 \vee X_7))]. \tag{3.1}$$

For a certain case i, where the Boolean equation L was true, the response Y_i was sampled from a Bernoulli random variable with probability $2/3$, otherwise it was sampled from a Bernoulli random variable with probability $1/3$. The score in the greedy search was chosen to be the number of misclassified cases (i.e., how often a proposed tree predicted the wrong response).

The best single predictor turned out to be predictor X_1, having a misclassification rate of 43.4% (434 out of 1,000 cases). The second step was splitting the first leaf into $X_1 \wedge X_3$, reducing the misclassification rate to 38.7%. After seven steps, the correct tree was visited (lower left panel in Figure 3). The true misclassification rate in this example is 33.1% However, the algorithm did not stop. There were possible moves from this tree that improved the score, the best being splitting leaf X_4 into $X_4 \wedge X_{12}$, which resulted in a tree having five fewer misclassifications than the tree representing the true Boolean expression. After that, the greedy algorithm took four more steps (not displayed as trees in Figure 3) until it stopped, yielding a low misclassification rate of 31.4%. These misclassification rates are displayed in the lower right panel in Figure 3 as solid squares. The true misclassification rates for the respective trees were calculated using model (3.1) and displayed in the panel as open squares. Through tree number seven (which represents the true underlying Boolean expression) the true misclassification rate is decreasing with the tree number in the greedy search. After that, however, the following trees also fit noise, and the true misclassification rate increases, as expected. This emphasizes the need for statistical tools for model selection, since in real-life problems the truth is unknown and the subject of the search. The emphasis in this article is on model selection using simulated annealing, as discussed in the next section. However, we also developed randomization tests, similar to those described in Section 4.3 for simulated annealing, for the greedy search. See Ruczinski (2000) for details of these tests.

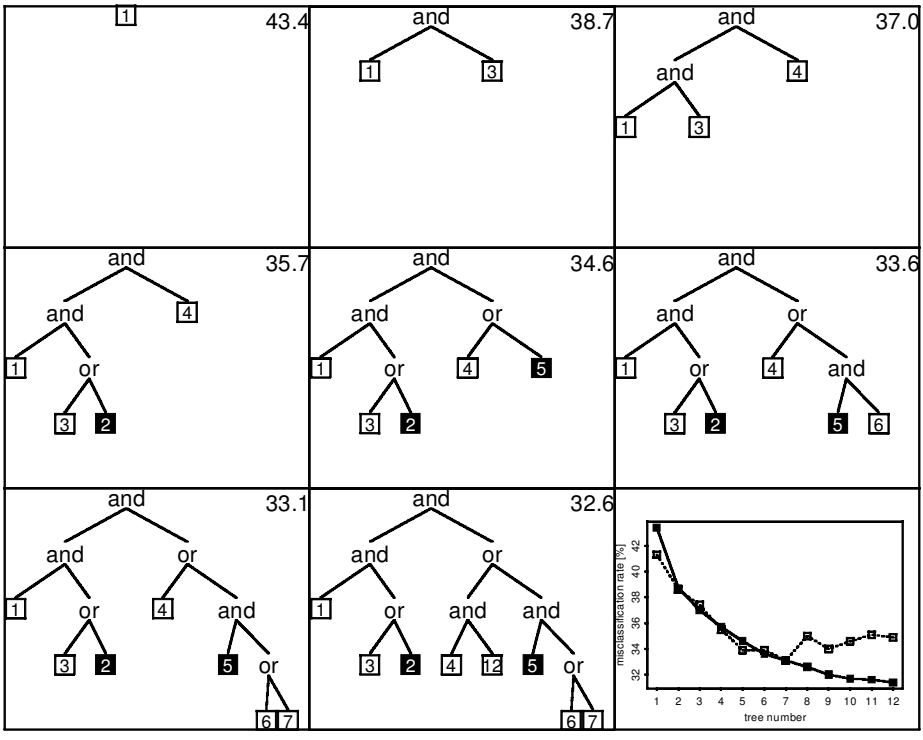


Figure 3. The sequence of the first eight trees visited in the greedy search. The number in the upper right corner is the percentage of misclassified observations in the simulated data (1,000 cases), using the respective tree as predictor. The panel on the lower right shows those mis-classification rates (solid squares), together with the true misclassification rates calculated from model (3.1) (open squares).

Unlike in the above example, it is possible that the correct tree is not visited at all. This can happen when the search gets stuck or, as shown below, if an incorrect variable gets chosen. This happens frequently, particularly when some of the predictors are correlated. To show this, we generated a dataset exactly as above, except this time we substituted variable X_8 (not in the model) by a surrogate of $X_3 \vee X_4$. For every case i we chose X_8^i to be a Bernoulli random variable with probability 0.9 if $X_3^i \vee X_4^i$ was true, and Bernoulli with probability 0.1 otherwise. The outcome of the search is displayed in Figure 4. Now in the the second step the variable X_8 is selected, and remains in the model until the very end. The true misclassification rate takes a big dip in Step 2, and the resulting tree is basically a surrogate for $(X_1 \wedge (X_3 \vee X_4))$ which is a better tree than any other tree with two leaves.

3.3 SEARCH VIA SIMULATED ANNEALING

The simulated annealing algorithm and its statistical properties are discussed in numerous publications such as the books by Otten and van Ginneken (1989) and van Laarhoven and Aarts (1987). Below, we briefly outline the basics of simulated annealing and describe how we use it to find good-scoring logic models.

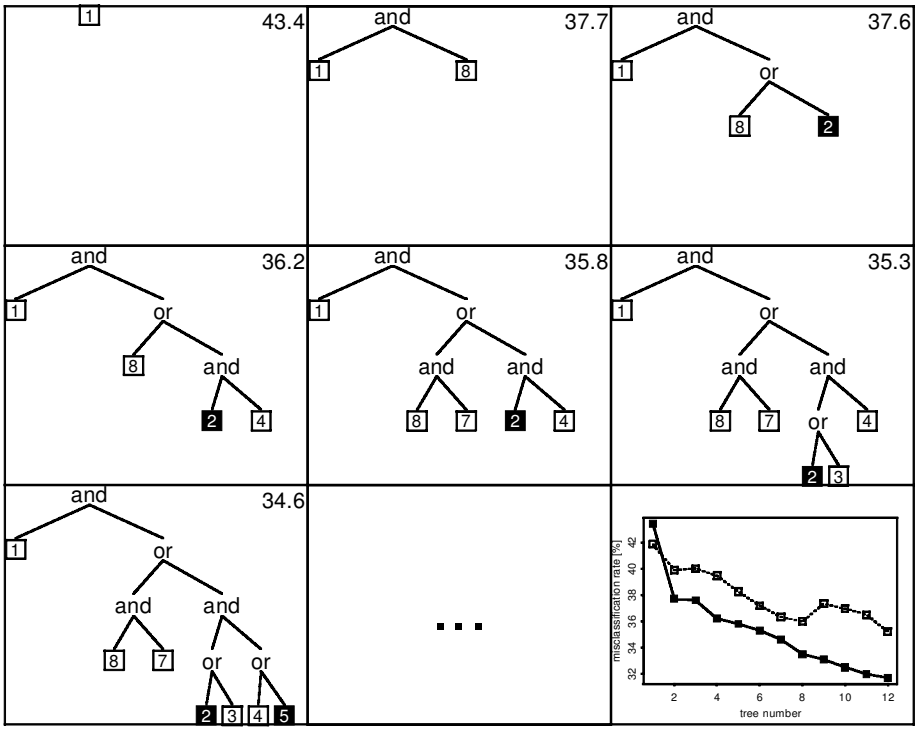


Figure 4. The sequence of the first seven trees visited in the greedy search on the data in which the predictors are not independent. Again, the number in the upper right corner is the percentage of misclassified observations in the simulated data (1,000 cases), using the respective tree as predictor. The panel on the lower right shows those misclassification rates (solid squares), together with the true misclassification rates calculated from the correct model (open squares).

The annealing algorithm is defined on a state space S, which is a collection of individual states. Each of these states represents a configuration of the problem under investigation. The states are related by a neighborhood system, and the set of neighbor pairs in S defines a substructure M in $S \times S$. The elements in M are called moves. Two states s, s' are called adjacent, if they can be reached by a single move (i.e., $(s, s') \in M$). Similarly, $(s, s') \in M^k$ are said to be connected via a set of k moves. In all our applications, the state space is finite.

The basic idea of the annealing algorithm is the following: given a certain state, pick a move according to a selection scheme from the set of permissible moves, which leads to a new state. Compare the scores of the old and the new state. If the score of the new state is better than the score of the old state, accept the move. If the score of the new state is not better than the score of the old state, accept the move with a certain probability. The acceptance probability depends on the score of the two states under consideration and a parameter that reflects at which point in time the annealing chain is (this parameter is usually referred to as the temperature). For any pair of scores, the further ahead we are in the annealing scheme, the lower the acceptance probability, if the proposed state has a score worse than the score of the old state. This algorithm generally leads to good-scoring states, see Otten and van

Ginneken (1989) and van Laarhoven and Aarts (1987).

There are various options how to implement the annealing algorithm and fit the logic models. We fit all trees in the model simultaneously. This requires for computational reasons that we preselect the maximum number t of trees, which can be chosen arbitrarily large (hardware permitting), which is a conservative choice if we have no a priori idea of how many trees we maximally want to fit. If t larger than necessary, and the model can be trimmed down if needed. In Section 4 this will be discussed in detail. For now, we assume that t is known and fixed.

We usually select one tree in the current logic model, and then randomly pick (following a predetermined distribution) a move from the move set for this tree. We refit the parameters for the new model, and determine its score, which we then compare to the score of the previous state (logic model), as described above. More details about simulated annealing in general and our implementation in particular is given in Appendix B.

3.4 FITTING OTHER MODELS WITH BINARY DATA

The methods discussed in Section 2.3 can further be distinguished by their mechanism that guides the search to find solutions. In general, scalability is one of, if not the, highest priority in analyzing data to find Boolean functions that have predictive power. Especially in data mining problems, fast and efficient search algorithms are crucial. Therefore, greedy type algorithms, similar to the one discussed in Section 3.2 are standard, and often give satisfactory results (Murthy and Salzberg 1995). In recent years, other search methods have gained popularity. As is the case for logic regression, greedy algorithms do not necessarily find a global optimum, and it has been recognized that alternatives should be considered if their computational expense is not prohibitive. Among those alternatives proposed are genetic algorithms (Vafaie and DeJong 1991; Bala, DeJong, Pachowicz 1991; Giordana and Saitta 1993; Wnek and Michalski 1994), Simulated Annealing (Fleisher et al. 1985; Sutton 1991; Lutsko and Kuijpers 1994), and a thermal training procedure that is somewhat similar to simulated annealing (Frean 1990). The recent statistics literature has seen a wealth of alternatives to straight greedy tree searches. Buntine (1992) proposed a Bayesian approach as a stochastic search tool for classification trees, and Chipman, George, and McCulloch (1998) and Denison, Mallick, and Smith (1998) published Bayesian algorithms for CART trees. Other approaches were proposed using the EM algorithm (Jordan and Jacobs 1994), bootstrapping based techniques such as bagging (Breiman 1996), bumping (Tibshirani and Knight 1999), and iterative reweighting schemes such as boosting (Freund and Schapire 1996), and randomized decision trees (Amit and Geman 1997; Dietterich 1999). Comparisons of some of those methods were, for example, carried out by Quinlan (1996), Dietterich (1999), and Breiman (1999). Another alternative to greedy searches is the "patient" rule induction method (PRIM) used by Friedman and Fisher (1999).

4. MODEL SELECTION

Using simulated annealing gives us a good chance to find a model that has the best or

close to best possible score. However, in the presence of noise in the data, we know that this model likely overfits the data. This section introduces some model selection tools for the simulated annealing search algorithm. Similar tools for the greedy search are discussed in Ruczinski (2000).

In this article, we use the total number of leaves in the logic trees involved in a model as a measure of model complexity, and call it the model size. Different measures are certainly possible and easily implemented, but not further discussed in this article. The first part of this section describes how we ensure finding the best models of a given size, which is necessary in certain types of model selection. The other two parts of this section describe the model-selection techniques we used in the case studies. Examples of these model selection techniques are found in Section 5. A potential alternative method of model selection is to penalize the score function for the size of the model, in the spirit of AIC, BIC, and GCV. One difference between our models and traditional setups where these measures are used is that for logic regression more complicated models do not necessarily have more parameters. We plan to investigate such measures in future work.

4.1 MODELS OF FIXED SIZE

In certain situations it is of interest to know the best scoring tree or the best model of a certain size. This is the case, for example, when cross-validation is used to determine the best overall model size, as we will see below. If the simulated annealing run is carried out with the move set as described in Section 3, the tree or model size changes constantly, and we cannot guarantee that the final model is of the desired size. To determine the best overall model of a fixed size, we prohibit moves that increase the tree when its desired size has been reached. In other words, we can carry out the simulated annealing as before, except we do not suggest branching a tree or splitting a leaf if the tree has already the desired size. Strictly speaking, this guarantees that we find the best of *up to* the desired size. However, smaller tree sizes are desirable in general, so this is not be a problem. In reality, the maximum (desired) tree size almost always is reached anyway, provided this size is not too large. An alternative approach would be to alter the move set discussed in Section 3.1 to include only moves that keep the size of the model unchanged (this requires the definition of some new moves). However, in our experience the altered move set approach is computationally considerably less efficient, both for the programmer (it complicates the code considerably) and for the computer (as the simulated annealing algorithm converges slower), than the approach described above.

4.2 TRAINING/TEST SET AND CROSS-VALIDATION

We can use a training and test-set or a cross-validation approach to determine the size of the logic tree model with the best predictive capability. When sufficient data are available, we can use the training set/test set approach. That is, we randomly assign the cases to two groups with predetermined sizes, using one part of the data as the training set, and the other

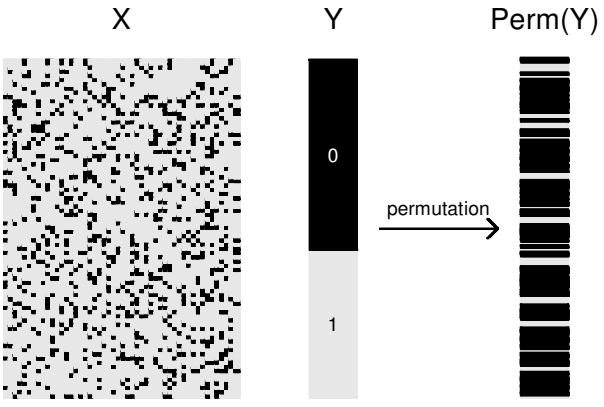


Figure 5. The setup for the null model randomization test, shown for binary response Y. In this test, we randomly permute the entire response.

part as test set. Thus, instead of using the entire data in the model fitting and model evaluation process as described above, we fit models of fixed size using the training set, and then pick a model size by scoring those models using the independent test set. When sufficient data for an independent test set is not available, we can use cross-validation instead. Assume we want to assess how well the best model of size k performs in comparison to models of different sizes. We split the cases of the dataset into m (approximately) equally sized groups. For each of the m groups of cases (say group i), we proceed as follows: remove the cases from group i from the data. Find the best scoring model of size k (as described in Section 3), using only the data from the remaining $m - 1$ groups, and score the cases in group i under this model. This yields score ϵ_{ki}. The cross-validated (test) score for model size k is $\epsilon_k = (1/m) \sum_i \epsilon_{ki}$. We can then compare the cross-validated scores for models of various sizes.

4.3 RANDOMIZATION

We implemented two types of randomization tests. The first one, referred to as "null model test," is an overall test for signal in the data. The second test, a generalization of this test, is a test for model size which can be used to determine an optimal model size. This section introduces the ideas behind those tests.

4.3.1 Null Model Test: A Test for Signal in the Data

For any model class we consider in our methodology (linear regression, logistic regression, etc.) we first find the best scoring model, given the data. Then, the null hypothesis we want to test is: "There is no association between the predictors X and the response Y." If that hypothesis was true, then the best model fit on the data with the response randomly permuted, as indicated in Figure 5 for a binary Y, should yield about the same score as the best model fit on the original data (Y does not have to be binary in general; it was

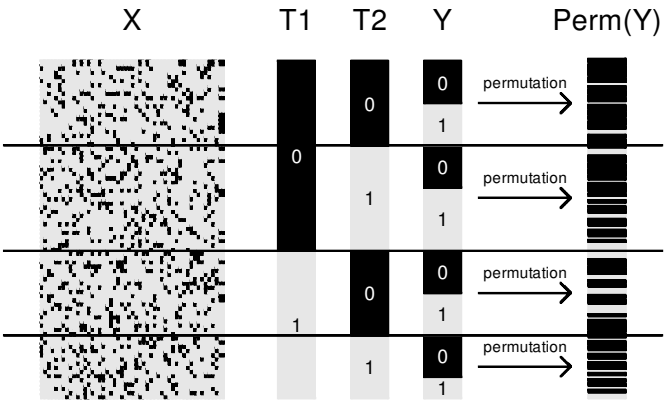


Figure 6. The setup for the sequential randomization test: in this test, we permute the response within the group of cases with the same fitted values for all existing trees (here T1 and T2).

chosen binary solely to illustrate the randomization technique in Figure 5). We can repeat this procedure as often as desired, and claim the proportion of scores better than the score of the best model on the original data as an exact p value, indicating evidence against the null hypothesis.

4.3.2 A Test to Detect the Optimal Model Size

If the above described test showed the existence of signal in the data, we want to determine the model that best describes the association between predictors and response. Assume that the best-scoring model has score ϵ^* and size k. We also find the best scoring models of sizes 0 through k. To find out which model size is optimal, we carry out a sequence of randomization tests. The null hypothesis in each test is: "The optimal model has size j, the better score obtained by models of larger sizes is due to noise," for some $j \in \{0, \ldots, k\}$. Assume the null hypothesis was true, and the optimal model size was j, with score ϵ_j. We now "condition" on this model, considering the fitted values of the logic model. For a model with p trees, there can be up to 2^p fitted classes. Figure 6 shows the setup for models with two trees.

We now randomly permute the response within each of those classes. The exact same model of size j considered best still scores the same (ϵ_j), but note that other models of size j potentially could now score better. We now find the overall best model (of any size) on the randomized data. This model will have a score (ϵ_j^{**}) that is at least as good, but usually better than ϵ_j. If the null hypothesis was true, and the model of size j was indeed optimal, then ϵ^* would be a sample from the same distribution as ϵ_j^{**}. We can estimate this distribution as closely as desired by repeating this procedure many times. On the other hand, if the optimal model had a size larger than j, then the randomization would yield on average worse scores than ϵ^*.

We carry out a sequence of randomization tests, starting with the test using the null model, which is exactly the test for signal in the data as described in the previous subsection.

We then condition on the best model of size one and generate randomization scores. Then we condition on the best model of size two, and so on. In general, model selection is now best carried out by comparing the successive histograms of randomization scores ϵ_j^{**}. If we need an automatic rule, we pick the smallest model size where a fraction less than p of the randomization scores have scores better than ϵ^*. Typically we choose p about 0.20, that is, larger than 0.05, as we do not want to exclude any possibly useful association.

4.4 APPROACHES TO MODEL SELECTION FOR OTHER MODELS WITH BINARY DATA

Model selection differs greatly among the methods discussed in Section 2.3. Especially in the early works on Boolean functions, it was often assumed that there was no noise in the data, and the authors strived for minimal complete and consistent rules, that is, the simplest Boolean functions that perfectly classify all examples (e.g., Michalski et al. 1986; Quinlan 1986; Hong 1997). In real-life examples, this "no-noise" assumption usually does not hold, and the above methods result in over-fitting. To account for that, a variety of algorithms were proposed. Some of those were simply modifications of previously mentioned algorithms. For example, Clark and Niblett (1989) pointed out that ID3 can easily modified and showed that extensions exist that circumvent this "no-noise" assumption, and Zhang and Michalski (1989) proposed a method called SG-TRUNC, which they implemented into AQ15 and released as version AQ16. Cross-validation is also commonly used for model selection; for example, in M5 (Quinlan 1992) and R^2 (Torgo and Gama 1996). Some other pruning techniques are based on the minimum description length principle, such as the one used in SLIQ (Mehta, Agrawal, Rissanen 1996), or based on cost-complexity consideration, for example, used in CART (Breiman et al. 1984). More pruning techniques and a comparison between those can be found in Mehta, Rissanen, and Agrawal (1995).

5. EXAMPLES

5.1 A SIMULATION STUDY

As discussed in Section 2.3, there exist many algorithms in the fields of machine learning, computer science, and statistics to model binary data. In the statistics literature, the two best-known adaptive algorithms are CART (Breiman et al. 1984), which is especially useful for modeling binary predictor data, and MARS (Friedman 1991), which was actually designed for continuous predictors, but can still be applied to binary data. As the type of models considered for each of these approaches is different, it is not hard to come up with an example where logic regression outperforms MARS and CART, or vice versa. The intent of the simulation study in this section is therefore not to show that logic regression is better or as least as good (in some sense) as CART and MARS, but rather to provide an example that shows that there exist situations in which the underlying model is particularly complicated,

Table 1. The Results for the Classification Part of the Simulation Study. The results are averages over ten runs. Logic regression can fit the true model using four terms, CART needs seven terms, and MARS needs two terms. The error rate is calculated using the differences between the true signal L and the predictions of the classifier, $\hat{L}$.

Method		*Logic*		*CART*	*MARS*
Selection	*Truth*	*CV*	*Random*	*CV*	*GCV*
true model size	—	4.0	4.0	7.0	2.0
fitted model size	—	3.4	3.9	3.2	3.7
number of predictors used	4	3.4	3.9	2.1	4.1
number of predictors $X_1, \ldots, X_4$ used	4	3.3	3.3	1.0	2.9
number of predictors $X_5, \ldots, X_{10}$ used	0	0.1	0.6	1.1	1.2
fraction of times X_{10} used	0	0.0	0.1	0.9	0.8
error rate relative to truth	0.0%	5.5%	8.4%	23.7%	15.4%

and logic regression outperforms CART and MARS. Coming up with reverse examples would be straightforward as well. One simulation considers an example that is especially difficult for CART. It was designed in a way that CART tends to choose a wrong initial split, from which it can not easily "recover." The true model is a fairly simple logic regression rule. Although there exists a CART tree that is equivalent to the logic regression rule used in the model, CART does not produce that tree in practice. MARS fares better in this study than CART, but not as well as logic regression.

We generated ten datasets, with 250 cases and 10 binary predictors each. Predictors X_1 through X_9 had an independent Bernoulli(0.5) distribution. Let $L = (X_1 \wedge X_2) \vee (X_3 \wedge X_4)$. Predictor X_{10} was equal to L with probability 0.5, and otherwise was a Bernoulli(7/16) random variable (note that $P(L = 1) = 7/16$ in our simulation). For each of the ten datasets, we generated a response for classification and a response for linear regression. For classification, the response for each case was sampled from a Bernoulli(0.7) distribution if L was true, and from a Bernoulli(0.3) distribution otherwise. In the regression case, the model was chosen as $Y = 5 + 2L + N(0, 1)$. For both classification and regression, we picked the best logic model for each of the ten datasets, separately using the cross-validation and the randomization procedure. The best model for CART was chosen by minimizing the 10-fold cross-validation estimates of the prediction error. The MARS models were chosen to minimize the GCV score with penalty = 3. MARS models were restricted to have no more than fourth order interactions and a maximum model size of 15 terms. The results are shown in Tables 1 and 2. We used the S-Plus program `tree` to run CART, and a version of MARS written by Tibshirani and Hastie available from http://www.stats.ox.ac.uk/pub/. For classification using MARS, we fit a regression model using binary response, and thresholded the fitted values at 0.5. The smallest model that logic regression can fit to model L has size four; the smallest CART model for L is shown in Figure A.1 (p. 506), and has seven terminal nodes; the smallest MARS model for L is $\beta_a X_1 X_2 + \beta_b X_3 X_4 - \beta_c X_1 X_2 X_3 X_4$, which has three terms. For the classification problem, MARS can produce the correct fitted values with the model $\beta_a X_1 X_2 + \beta_b X_3 X_4$, however. In Tables 1 and 2 we refer to the number of terminal nodes for CART models and the number of terms in a MARS model as "model size."

For logic regression, model selection using randomization yields models of similar size

Table 2. The Results for the Regression Part of the Simulation Study. The results are averages over ten runs. Note: Logic regression can fit the true model using four terms, CART needs seven terms, and MARS needs three terms. The root mean squared error is calculated using the squared differences between the true signal $5 + 2L$ and the predictions of the fitted model, $\hat{\beta}_0 + \hat{\beta}_1 \hat{L}$.

Method		*Logic*		*CART*	*MARS*
Selection	*truth*	*CV*	*Random.*	*CV*	*GCV*
true model size	—	4.0	4.0	7.0	3.0
fitted model size	—	4.0	4.0	10.0	5.0
number of predictors used	4	4.0	4.0	5.4	4.4
number of predictors $X_1, \ldots, X_4$ used	4	4.0	4.0	4.0	3.9
number of predictors $X_5, \ldots, X_{10}$ used	0	0.0	0.0	1.4	0.5
fraction of times X_{10} used	0	0	0	1.0	0.2
root mean squared error relative to truth	0.00	0.07	0.07	0.47	0.34

as cross-validation. This is using the automated procedure, as described in Section 4.3.2, with a cut-off of $p = 0.2$. When we visually examined plots like Figure 11 in Section 5.2, we sometimes selected different models than the automated procedure, and actually ended up with results for the randomization procedure that are better than those using cross-validation. CART selects much larger models, and often includes the "wrong" predictors. In general it is very much tricked by predictor X_{10}. For classification, MARS often substitutes one of the correct predictors by X_{10}. Although MARS often has the correct predictors in the selected model, the model is usually larger than the smallest model that is needed, leading to less interpretable results. For the regression part of the simulation study all methods perform better as the signal was stronger. Both logic regression approaches get the correct model each time; CART still is tricked by X_{10}; MARS actually picks the correct predictors seven out of ten times, but usually with a too-complicated model. For the regression part of the simulation study, CART always ends up with models with considerably more terminal nodes than needed.

Tables 1 and 2 also provide the error rate of the fitted model relative to the true model $L = (X_1 \wedge X_2) \vee (X_3 \wedge X_4)$ for classification and $5+2L$ for regression over the design points (see table captions). For the classification problem we note that logic regression yields error rates of under 10%, while MARS and CART have error rates of 15% and 23%, respectively. This should be compared with the 30% noise that we added to the truth. For the regression problem, both logic regression approaches always find the correct model, but since there is random variation in the estimates of the coefficients, the root mean squared error (RMSE) relative to the truth is not exactly 0. Again, for the regression problem MARS does much better than CART. The RMSE for these approaches can be compared to the noise, which had standard deviation 1.

5.2 THE CARDIOVASCULAR HEALTH STUDY

The Cardiovascular Health Study (CHS) is a study of coronary heart disease and stroke in elderly people (Fried et al. 1991). Between 1989 and 1990, 5,201 subjects over the age

Table 3. The 23 Regions of the Cardiovascular Health Study Brain Atlas. The predictor number will be used later to display logic trees and to describe linear models we fit. The counts are the number of CHS patients for whom an infarct was diagnosed. The letters in the first column indicate the standard anatomical clusters of the above 23 locations. They are Cerebral Cortex (Cluster A), Cerebellar Cortex (Cluster B), Cerebellar White Matter (Cluster C), Basal Ganglia (Cluster D), Brain Stem (Cluster E) and Cerebral White Matter (Cluster F).

Cluster	*Predictor*	*Region*	*Counts*
A	1	Anterior Cerebral Artery (frontal lobe)	16
	2	Anterior Cerebral Artery (parietal lobe)	4
	3	Middle Cerebral Artery (frontal lobe)	62
	4	Middle Cerebral Artery (parietal lobe)	43
	5	Middle Cerebral Artery (temporal lobe)	64
	6	Posterior Cerebral Artery (parietal lobe)	6
	7	Posterior Cerebral Artery (temporal lobe)	12
	8	Posterior Cerebral Artery (occipital lobe)	31
B	9	Superior Cerebellar Artery	23
	10	Anterior Inferior Cerebellar Artery	7
	11	Posterior Inferior Cerebellar Artery	53
C	12	Cerebellar White Matter	58
D	13	Caudate	110
	14	Lentiform Nuclei	601
	15	Internal Capsule (anterior limb)	156
	16	Internal Capsule (posterior limb)	77
	17	Thalamus	236
E	18	Midbrain	15
	19	Pons	29
	20	Medulla Oblongata	0
F	21	Watershed (ACA to MCA)	10
	22	Watershed (MCA to PCA)	12
	23	Cerebral White Matter	217

of 65 were recruited in four communities in the United States. To increase the minority representation in the study, an additional 687 African Americans were recruited between 1992 and 1993. During 1992 and 1994, a subset of these patients agreed to undergo an MRI scan. Neuroradiologists, blinded to the identity and characteristics of those patients, read the images and attempted to identify the presence and locations of infarcts, defined as an area of brain tissue damaged by lack of oxygen due to compromised blood flow. For 3,647 CHS participants, MRI detected strokes (infarcts bigger than 3mm) were recorded as entries into a 23 region atlas of the brain. Up to five infarcts were identified per subject, and each of those infarcts was present in up to four of the 23 locations (i.e., a single infarct was detectable in up to four regions). For every patient the infarcts were recorded as binary variables (absent/present) in 23 regions. Table 3 lists the 23 regions of the CHS atlas and the number of patients with infarcts in those regions. For more details on the scanning procedure, see Bryan et al. (1994).

One of the goals of the MRI substudy of the CHS is to assess the associations between stroke locations and various response variables, such as the one resulting from the mini-

mental state examination, a screening test for dementia. Patients participate in a question-and-answer session, and score points for each correct answer. The final score, the sum of all points earned, is a number between 0 (no correct answer) and 100 (everything correct). For more details about this test see Teng and Chui (1987). In our analysis we focused on demonstrating the use of the logic regression methodology. In practice, maybe only a subset of the predictors should be included in the analysis since, for example, there are spatial correlations between predictors. Using a clustering approach, McClelland (2000) reported improvements after adjusting for gender, age, and so on. These variables were not available to us when we started analyzing the data, and we did not include those into our model. Ruczinski (2000) discussed several approaches to include continuous predictors in logic regression models. See also the example on SNP data in Section 5.3, in which we included continuous predictors as additional variables.

Most patients scored in the 90s in the mini-mental test, and we found that a logarithmic transformation was beneficial after investigating potential model violations such as nonnormally distributed (skewed) residuals. Since the mini-mental score is a number between 0 and 100, we defined the response variable as

$$Y := \log(101 - [\text{mini-mental score}]).$$

Usually such a transformation is avoided because it makes the parameter interpretation and their association to the original mini-mental score rather difficult, but since in our methodology we only deal with a very small number of classes of patients (see further analysis below) this is of no importance, as the few fitted values associated with the mini-mental score can simply be listed, without causing confusion.

The models we investigated were linear models of the form

$$Y = \beta_0 + \beta_1 \times I_{\{L_1 \text{ is true}\}} + \cdots + \beta_p \times I_{\{L_p \text{ is true}\}} + \epsilon,$$

with $\epsilon \sim N(0, \sigma^2)$ and various numbers p of Boolean expressions. These models define groups of patients with similar stroke patterns in the brain atlas within each group, and different average scores between groups. In the following analysis, we used the residual variance (residual sums of squares divided by the number of cases considered) as scoring function.

We first carried out a "null model" randomization test as described in Section 4.3.1, fitting only one tree, and allowing up to 16 leaves in the tree. The null model, fitting the intercept β_0 with $\beta_1 = 0$ had a score of 0.7202. Using simulated annealing, the best scoring model we found had a score of 0.7029. We permuted the response, refit the model, recorded the score, and repeated this procedure 1,000 times. In Figure 7 we compare the score for the best scoring model [a], the score for the null model [b], and a histogram of the scores obtained from the randomization procedure. Since all of those randomization scores are considerably worse than the score [a], we conclude that there is information in the predictors with discriminatory power for the (transformed) mini-mental score. The results we got using models with more than one tree were similar.

To get a better idea of the effect of varying model sizes, we show the training scores

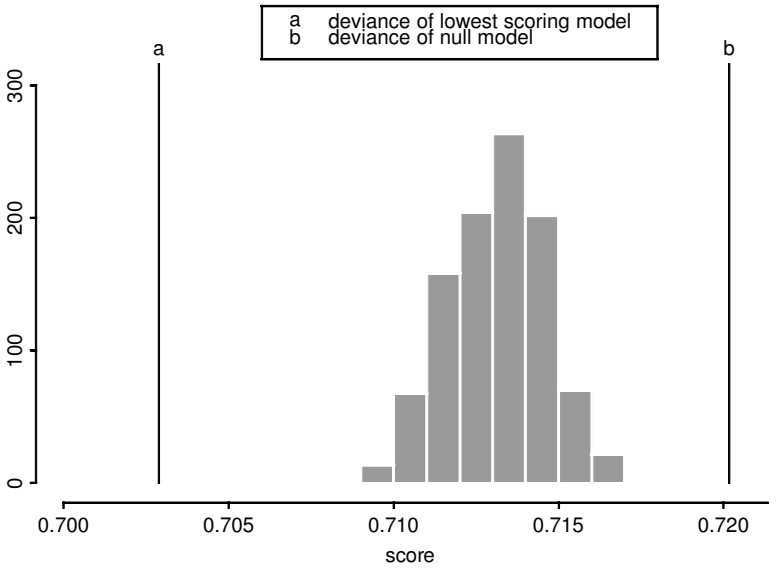


Figure 7. The residual variance scores of the null model randomization test. The scores in the histogram came from linear models with a single tree, allowing up to 16 leaves.

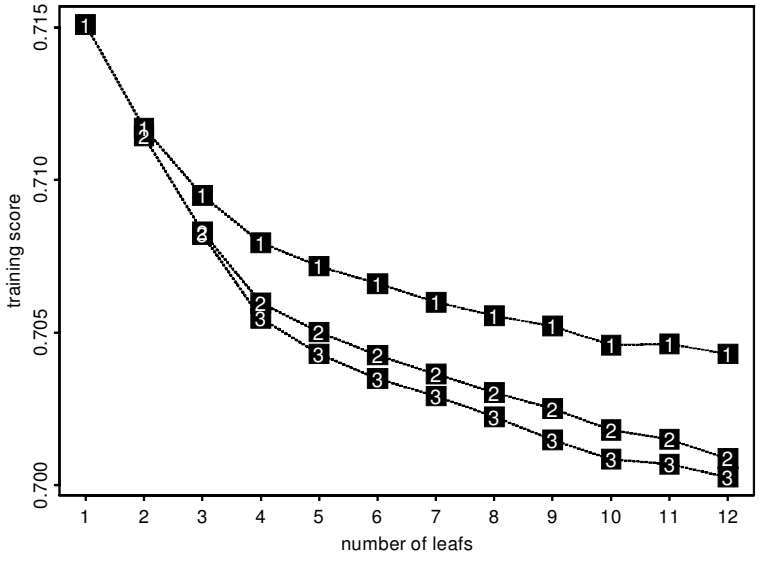


Figure 8. The training cross-validation scores of the best linear models with various numbers of leaves and trees. The number of trees allowed in the linear model is indicated by the white number super-imposed on the black squares.

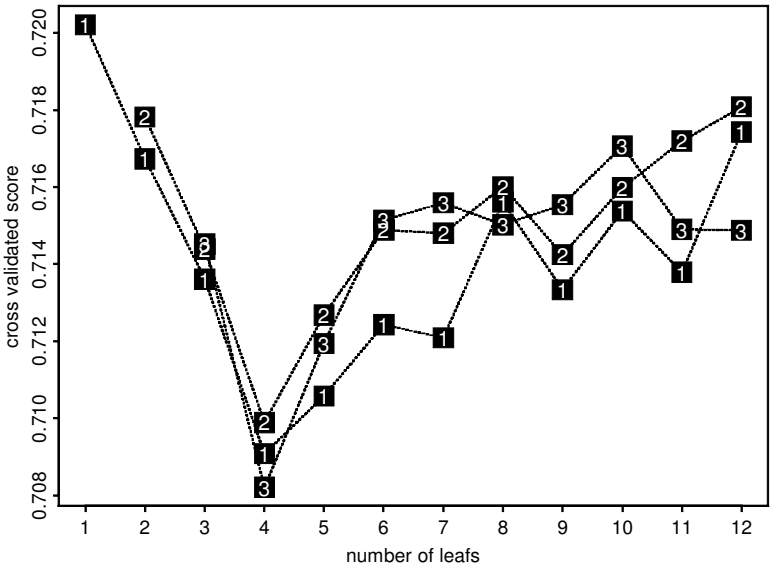


Figure 9. The cross-validation test scores of the linear models with various numbers of leaves and trees. The number of trees allowed in the linear model is indicated by the white number super-imposed on the black squares.

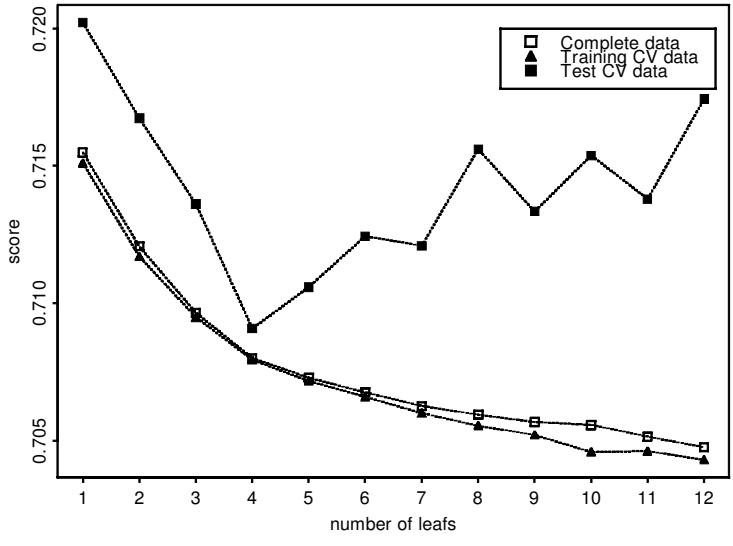


Figure 10. Comparison of the scores obtained by model fitting on the complete data (open squares), and the scores obtained from cross-validation (training sets: black triangles; test sets: black squares), allowing only one tree in the models.

(averages of all ten training scores from ten-fold cross-validation) of linear models with one, two, three trees and varying tree sizes in Figure 8 combined. Any model in the class of models with up to k trees and n leaves is also in the class of models with up to $k + 1$ trees and n leaves. Further, any model in the class of models with up to k trees and n leaves is also in the class of models with up to k trees and $n + 1$ leaves. Therefore the results are as expected: comparing two models with the same number of trees allowed, the model involving more leaves has a better (training) score. The more trees we allow in the model, the better the score for a fixed number of leaves. Allowing a second tree in the linear models seems to have a larger effect in the training scores (relative to the models with only one tree), than the effect of allowing a third tree in addition to the two trees.

However, larger models do not imply better cross-validation test scores, as shown in Figure 9. The cross-validated scores for models with identical number of leaves but different numbers of trees look remarkably similar. Regardless whether we allow one, two, or three trees, the models of size four seem to be favored. Since allowing for more than one tree obviously is not very beneficial with respect to the cross-validation error, we pick for simplicity the model with only one tree and four leaves. We should keep in mind that a model with k trees involves estimating $k + 1$ parameters. Thus, a model with three trees, with the same number of leaves as a model with one tree, is more complex.

We compare the cross-validated training and test scores to the scores using the complete data in Figure 10. For all model sizes, the scores obtained using the complete data are almost the same as the scores obtained from the cross-validation training data. This indicates that the same predictors were selected in most models in the cross-validation training data compared to the same sized models fitted using the complete data. As expected, the scores from the cross-validation test data are worse than the training scores. However, the test score for the model with four leaves is remarkably close to the training scores, indicating that the same predictors were selected in most models of size four using the cross-validation training data. The vast gap between training and test scores for the models of any size other than four strongly discourages the selection of those models.

Figure 11 shows histograms of the randomization scores after conditioning on various numbers of leaves, indicated by the number in the upper left corner of each panel. The averages of the randomization scores decrease when we increase the model size we condition on, until we condition on the model with four leaves. The overall best score for models with one tree (indicated by the left vertical bar in the figure) looks like a sample from the distribution estimated by the randomization scores after conditioning on four or more leaves, supporting the choice of four leaves in the selected model. The model with four leaves would also be selected by the automated rule with $p = 0.20$, described in Section 4.3.2.

Searching for the best linear model with a single tree of size four using the complete data yielded

$$Y = 1.96 + 0.36 \times I_{\{L \text{ is true}\}} \tag{5.1}$$

with the logic tree as shown in Figure 12.

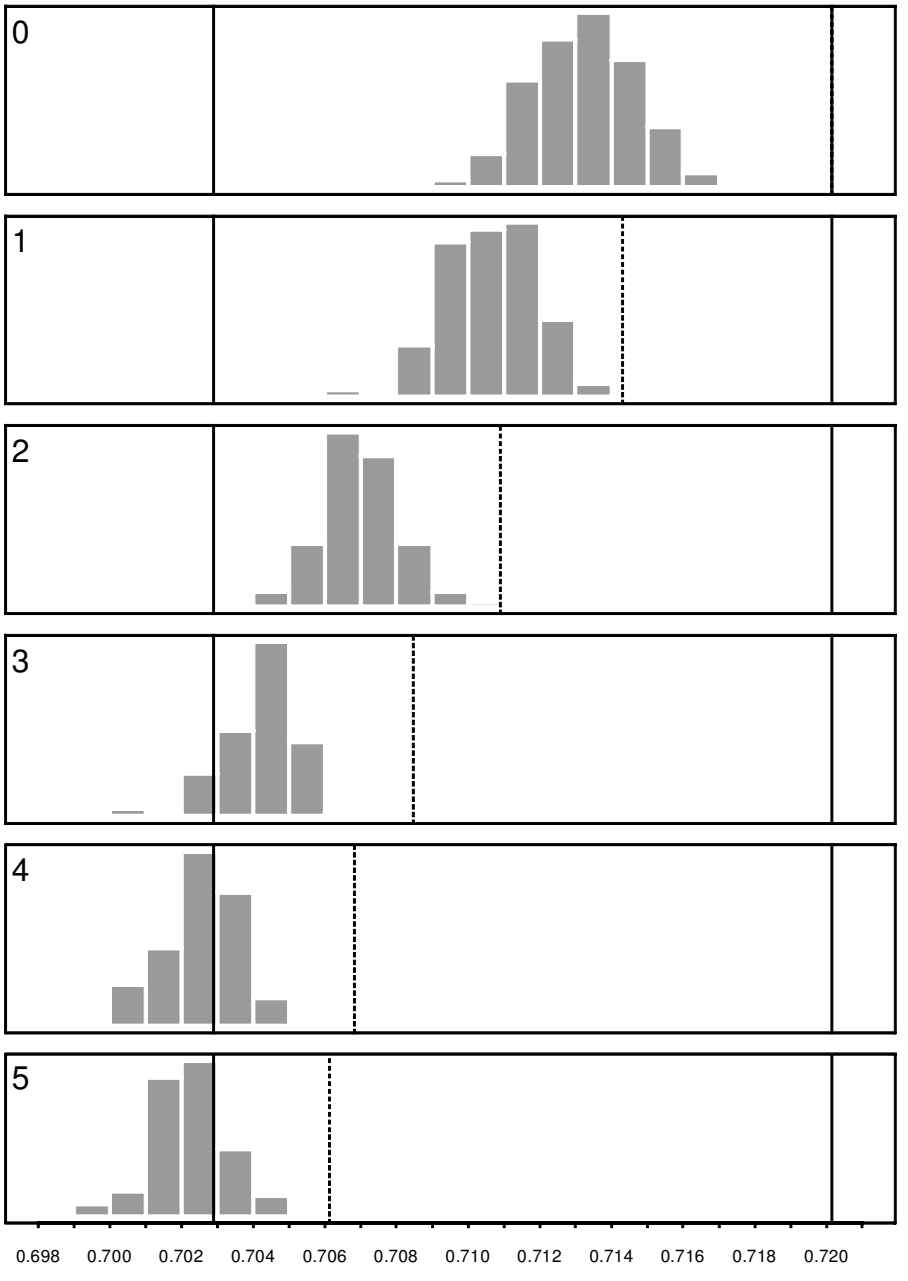


Figure 11. The residual variance scores of the randomization test for model selection. Randomization scores were obtained conditioning on 0, 1, 2, 3, 4, and 5 leaves, indicated by the number in the upper left corner of each panel. The solid bar on the right in the panels indicates the score of the null model, the solid bar more to the left indicates the best overall score found using the original (non-randomized) data. The dashed bar in each panel indicates the best score found for models with one tree and the respective size, using the original (nonrandomized) data.

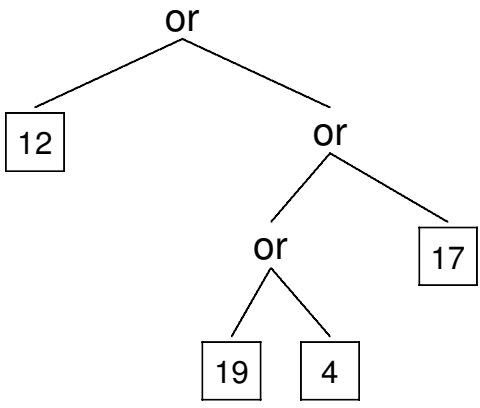


Figure 12. The Logic Tree of the selected model in Equation (5.1), representing the Boolean equation $L = X_{12} \vee X_{19} \vee X_4 \vee X_{17}$.

5.2.1 Comparison to Other Modeling Approaches

There are 58 patients with an infarct in region 12, 29 patients with an infarct in region 19, 43 patients with an infarct in region 4, and 236 patients with an infarct in region 17. These 366 infarcts occurred in 346 different patients, thus these four predictors are pairwise almost exclusive. Since the Boolean expression in (5.1) has only $\vee$ operators, the above model is very similar to a linear regression model with these four predictors as main effects, summarized in Table 4. (This linear regression model was also found using backwards stepwise regression. The initial model contained all predictors and no interactions.) Note that 0.36, the parameter estimate for β_1 in the logic model, is almost within the 95% confidence region for each region parameter in the linear regression model.

Even though the linear model looks very similar to the model in (5.1), the logic model clearly has a much simpler interpretation: using the Logic Regression methodology, we found two classes of patients that differ in health history and the performance in the mini-mental test. Using the fitted values from model (5.1), we can state that the estimated median transformed mini-mental score for patients with infarcts in either region 4, 12, 17, or 19 was 2.32, compared to 1.96 for patients who did not have an infarct in any of those regions. This translates to a median mini-mental score of 90.8 for patients with infarcts in either region

Table 4. Summary of the Regular Regression Model that Compares with the Logic Regression Model for the Mini Mental Score Data.

	$\hat{\beta}$	$\hat{\sigma}$	*t value*
Intercept	1.961	0.015	133.98
Region 4	0.524	0.129	4.06
Region 12	0.460	0.112	4.09
Region 17	0.236	0.057	4.17
Region 19	0.611	0.157	3.89

4, 12, 17, or 19, compared to a median mini-mental score of 93.9 for patients that did not have an infarct in any of those four regions. The residual error is almost exactly the same in these two models. However, the "standard" linear regression model, by using four binary predictors, defines $2^4 = 16$ possible classes for each patient. But there is no patient with three or four strokes in regions 4, 12, 17, and 19. Therefore, 5 of those 16 classes are empty. There is also a significant difference in the performance in the mini-mental test between patients with no strokes in regions 4, 12, 17, and 19, and those who do. But there is no significant "additive" effect, that is, patients with two strokes in regions 4, 12, 17, and 19 did not perform worse than people with one stroke in either of those regions. The logic model therefore, besides being simpler, also seems to be more appropriate.

The MARS model is very similar to the linear regression model. It was chosen to minimize the GCV score, with penalty equal 3. The maximum number of terms used in the model building was set to 15 and only interactions up to degree 4 were permitted. The model search yielded

$$Y = 1.96 + 0.53X_4 + 0.37X_{12} + 0.24X_{17} + 0.61X_{19} + 1.05(X_{12} * X_{15}). \tag{5.2}$$

The additional term compared to the linear regression model concerns patients who have a stroke in both regions 12 and 15; there are only five patients for which this applies. However, these five patients indeed performed very poorly on the mini-mental state examination. Although this term certainly is within the search space for the logic model, it did not show up in the logic model that was selected using cross-validation and randomization, because it applied only to 5 out of more than 3,500 patients. The score (residual error) of the MARS model is almost identical to the previously discussed models. CART returned a model that used the same variables as the linear and the logic model, shown in Figure 13. Only if all of the four variables are zero (i.e., the patient has no stroke in either of the four regions), a small value (less than 2) is predicted. In fact, a simple analysis of variance test does not show a significant difference between the four other group means (and if one bundles them, one obtains the logic model). So while the model displayed in Figure 13 is a fairly simple CART tree, we feel that the logic model is easier to interpret.

5.3 APPLICATION TO SNP DATA

This section briefly describes an application of the logic regression algorithm to genetic SNP data. This application was described in detail by Kooperberg, Ruczinski, LeBlanc, and Hsu (2001). Single base-pair differences, or single nucleotide polymorphisms (SNPs), are one form of natural sequence variation common to all genomes, estimated to occur about every 1,000 bases on average. SNPs in the coding region can lead to amino acid substitutions and therefore impact the function of the encoded protein. Understanding how these SNPs relate to a disease outcome helps us understand the genetic contribution to that disease. For many diseases the interactions between SNPs are thought to be particularly important (see the quote by Lucek and Ott in the introduction). Typically with any one particular SNP only two out of the four possible nucleotides occur, and each cell contains a pair of every

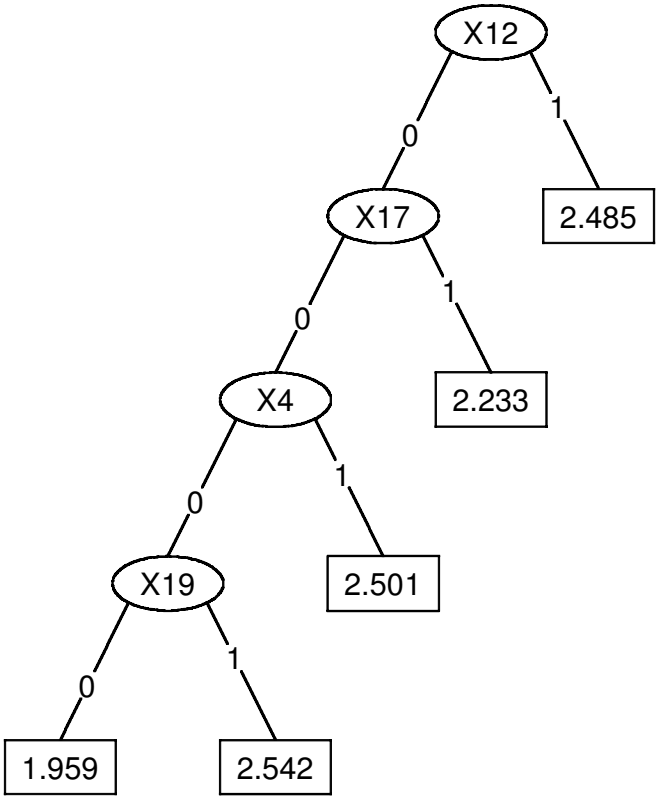


Figure 13. CART tree for the mini-mental score data.

autosome. Thus, we can think of an SNP as a random variable X taking values 0, 1, and 2 (e.g., corresponding to the nucleotide pairs AA, AT/TA, and TT, respectively). We can recode this variable corresponding to a dominant gene as $X_d = 1$ if $X \geq 1$ and $X_d = 0$ if $X = 0$ and as a recessive gene as $X_r = 1$ if $X = 2$ and $X_r = 0$ if $X \leq 1$. This way, we generate $2p$ binary predictors out of p SNPs. The logic regression algorithm is now well suited to relate these binary predictors with a disease outcome.

As part of the 12th Genetic Analysis Workshop (Wijsman et al. 2001), the workshop organizers provided a simulated genetic dataset. These data were simulated under the model of a common disease. A total of 50 independent datasets were generated, consisting of 23 pedigrees with about 1,500 individuals each, of which 1,000 were alive. The variables reported for each living subject included affection status, age at onset (if affected), gender, age at last exam, sequence data on six genes, five quantitative traits, and two environmental variables. We randomly picked one of the 50 datasets as our training set, and another randomly chosen dataset as our test set. In the sequence data were a total of 694 sites with at least 2% mutations distributed over all six genes. We coded those into $2 \times 694 = 1{,}388$ binary variables. In the remainder we identify sites and coding of variables as follows: $Gi.D.Sj$ refers to site j on gene i, using dominant coding, that is, $Gi.D.Sj = 1$ if at least one variant allele exist. Similarly, $Gi.R.Sj$ refers to site j on gene i, using recessive

coding, that is, $Gi.R.Sj = 1$ if two variant alleles exist. We identify complements by the superscript ^{c}, for example, $Gi.D.Sj^c$.

As our primary response variables we used the affected status. We fitted a logistic regression model of the form

$$\text{logit(affected)} = \beta_0 + \beta_1 \times \text{environ}_1 + \beta_2 \times \text{environ}_2 + \beta_3 \times \text{gender} + \sum_{i=1}^{K} \beta_{i+3} \times L_i. \quad (5.3)$$

Here gender was coded as 1 for female and 0 for male, environ_j, $j = 1, 2$, are the two environmental factors provided, and the L_i, $i = 1, \ldots, K$ are logic expressions based on the 1,388 predictors that were created from the sequence data.

Initially we fit models with $K = 1, 2, 3$, allowing logic expressions of at most size 8 on the training data. Figure 14 shows the deviance of the various fitted logic regression models. As very likely the larger models over-fit the data, we validated the models by computing the fitted deviance for the independent test set. These test set results are also shown in Figure 14. The difference between the solid lines and the dotted lines in this figure shows the amount of adaptivity of the logic regression algorithm. From this figure we see that the models with three logic trees with a combined total of three and six leaves have the lowest test set deviance. As the goal of the investigation was to identify sites that are *possibly* linked to the outcome, we preferred the larger of these two models. In addition, when we repeated the experiment on a training set of five replicates and a test set of 25 replicates, the model with six leaves actually had a slightly lower test set deviance than the model with three leaves (results not shown). We also carried out a randomization test which confirmed that the model with six leaves actually fitted the data better than a model with three leaves. The trees of the model with six leaves that was fitted on the single replicate are presented in Figure 15. The logistic regression model corresponding to this logic regression model is

$$\begin{aligned} \text{logit(affected)} &= 0.44 + 0.005 \times \text{environ}_1 - 0.27 \times \text{environ}_2 \\ &\quad + 1.98 \times \text{gender} - 2.09 \times L_1 + 1.00 \times L_2 - 2.82 \times L_3. \quad (5.4) \end{aligned}$$

All but the second environment variable in this model are statistically significant.

As the GAW data is simulated data, there is a "correct" solution (although this is not a logic regression model). The solution, which we did not look at until we selected model (5.4), showed that there was no effect from genes 3, 4, and 5, that all the effect of gene 1 was from site 557, that all the effect of gene 6 was from site 5782, and that the effect of gene 2 was from various sites of the gene. In addition the model showed an interaction between genes 1 and 2, while the effect of gene 6 was additive (but on a different scale than the logit scale which we used). For all 1,000 subjects with sequence data in replicate 25 (which we used), site 76 on gene 1 is exactly the opposite of site 557, which was indicated as the correct site on the solutions (e.g., a person with v variant alleles on site 76 always has $2 - v$ variant alleles on site 557 in the copy of the data which we used). Similarly, the logic regression algorithm identified site 5,007 on gene 6, which is identical for all 1,000 persons to site 5,782, the site which was indicated on the solutions. We note that the "correct" site on gene 1 appears twice in the logic tree model. Once, as a recessive coding ($G1.R.S557$)

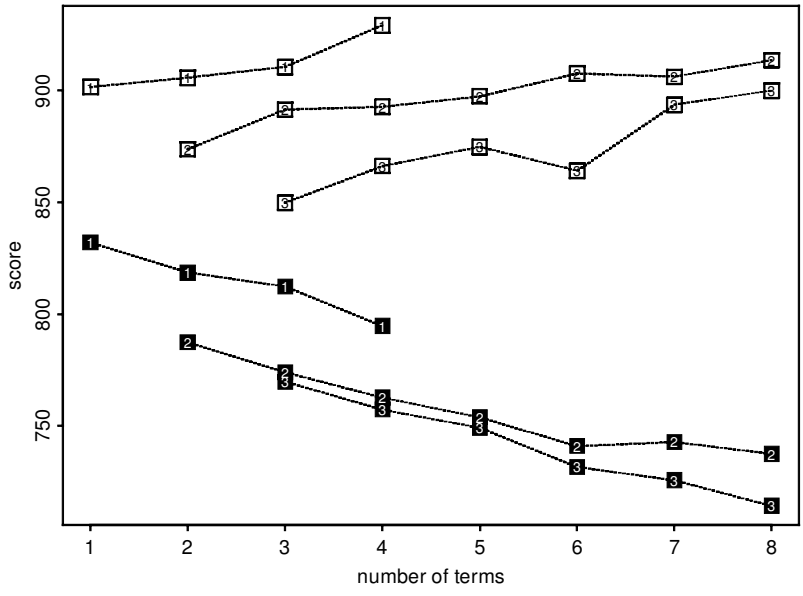


Figure 14. Training (solid) and test (open) set deviances for Logic Regression models for the affected state. The number in the boxes indicate the number of logic trees.

and one effectively as a dominant coding ($G1.R.S76^c \equiv G1.D.S557$ on this replicate) for site 557, suggesting that the true model may have been additive. The three remaining leaves in the model are all part of gene 2: two site close to the ends of the gene and one site in the center. All three sites on gene two are dominant.

In summary, the logic regression model identified all sites that were related to the affected status, the correct interaction between genes 1 and 2, and identified no false positives. As described by Witte and Fijal (2001), the logic regression approach was out of ten different approaches the only one that identified all correct sites and had no false positive SNPs.

6. DISCUSSION

Logic regression is a tool to detect interactions between binary predictors that are associated with a response variable. The strength of logic regression is that it can find even complicated interactions between predictors that may play important roles in application areas such as genetics or identifying prognostic factors in medical data. Logic regression considers a novel class of models, different from more established methodologies, such as CART and MARS. Clearly, depending on the application area, there will be situations where any method outperforms other method. We see logic regression as an additional tool in the statistical modelers' toolbox.

The logic regression models are not restricted to classification or linear regression, as many of the existing methodologies are. Any other (regression) model can be considered as

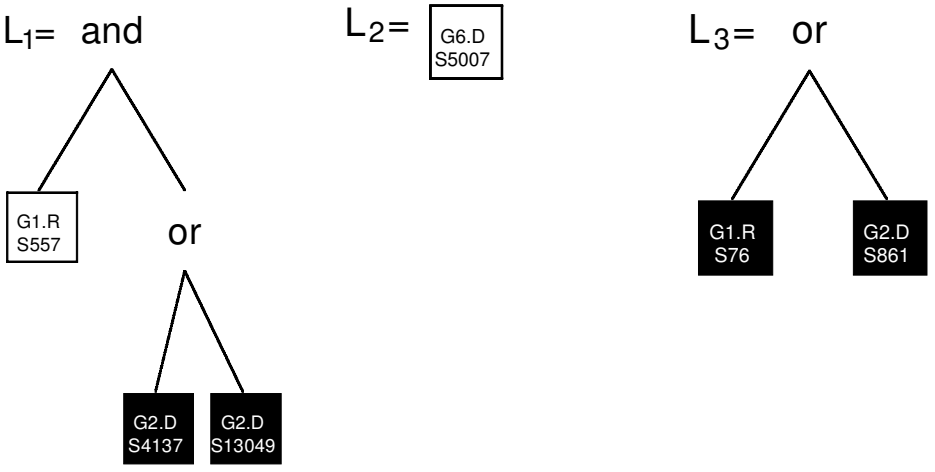


Figure 15. Fitted Logic Regression model for the affected state data with three trees and six leaves. Variables that are printed white on a black background are the complement of those indicated.

long as a scoring function can be determined. In our software for logic regression, we also implemented the Cox proportional hazards model, using partial likelihood as score. Simple classification problems are trivial to implement, using for example the (weighted) number of misclassifications as score. More complicated models also need to be considered when analyzing data with family structure, which is often the case in genetic data. Since family members are genetically related and often share the same environment, observations are no longer independent and one has to take those dependencies into account in modeling the covariance structure. As we envision that logic regression will be useful analyzing genetic data, we plan to investigate scoring functions that take familial dependence into account.

As the class of models that we consider in logic regression can be very large, it is critical to have a good search algorithm. We believe that the simulated annealing algorithm that we use is much more efficient in finding a good model than standard greedy algorithms. In future work we plan to investigate the use of Markov chain Monte Carlo algorithms, to assess uncertainty about the selected models. Because of the similarity between simulated annealing and McMC, computationally (but not conceptually) this should be straightforward. As is the case for many adaptive regression methodologies, straightforward application of the logic regression algorithm would likely over-fit the data. To overcome this, we have discussed a variety of model selection techniques that can be used to select the "best" logic regression model. At this point, these techniques are computationally fairly expensive (the running time for finding the best model or the best model of a particular size for the mini mental data took about a minute on a current generation Linux machine; therefore 10-fold cross-validation, where we investigated models of 30 different sizes, and the randomization tests took several hours). Other techniques, some of them much less computationally intense, seem to be feasible. This includes, for example, using a likelihood penalized by the size of the model as score, which would require only one run of the search algorithm. This is similar to using quantities such as AIC, BIC, or GCV in other adaptive regression

methodologies. We did not discuss this approach as we wanted to restrict the length of this article.

The logic regression software is available from bear.fhcrc.org/~ingor/logic. The algorithm also includes an option to search for models using only conjunctive or only disjunctive Boolean expressions. This was achieved in a straightforward manner by altering the move set to only allow one type of operator. Future versions will include several approaches to include continuous predictors in logic regression models.

APPENDIXES

A. RELATIONSHIP BETWEEN LOGIC TREES, DECISION TREES (CART), AND BOOLEAN EXPRESSIONS IN DISJUNCTIVE NORMAL FORM

The disjunctive normal form (DNF) is a widespread type of notation of logic statements in the engineering and computer science literature. A DNF is a Boolean expression, written as $\vee$-combinations of $\wedge$-terms. For example, the Boolean expression

$$(A \wedge B^c) \wedge [(C \wedge D) \vee (E \wedge (C^c \vee F))]$$

can be written in disjunctive normal form as

$$(A \wedge B^c \wedge C \wedge D) \vee (A \wedge B^c \wedge E \wedge C^c) \vee (A \wedge B^c \wedge E \wedge F).$$

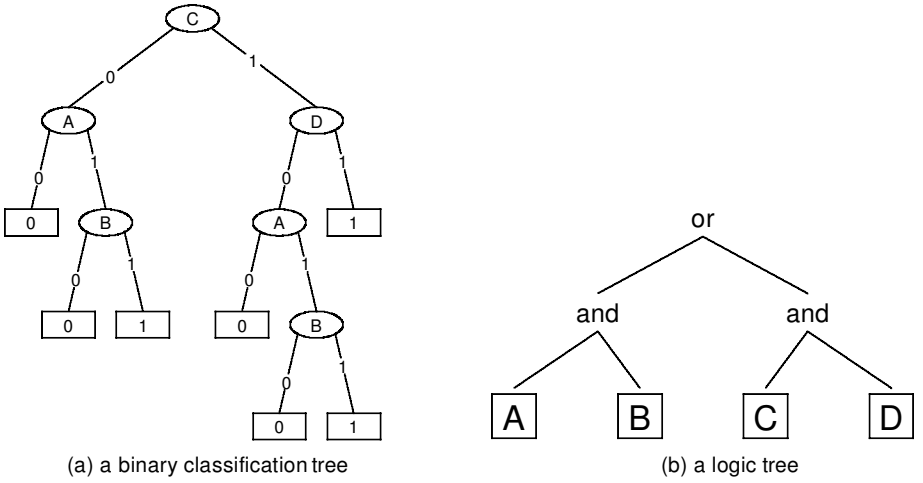
It can be shown that Boolean expressions and logic trees, as well as other logic constructs such as Boolean expressions in DNF are equivalent in the sense that the classes of logic expressions they represent are the same. That means, for example, that every Boolean expression can be represented in DNF, and as a logic tree (Ruczinski 2000).

The logic trees introduced in this article may appear to resemble classification trees (e.g., Breiman et al. 1984). In every classification tree, a leaf can be reached by a path through the tree, making decisions at every knot. If the tree is binary, these decisions reduce to checking whether or not the condition investigated at a particular knot is true or false. To reach a certain leaf, all conditions $C_1, C_2, \ldots, C_k$ along the path have to be satisfied (i.e., $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ has to be true). In general, there are multiple paths that reach a leaf that predicts class 1. Since there are only two outcome classes (0 and 1), the collection of all paths $P_1, P_2, \ldots, P_l$ that reach a leaf predicting 1 is a complete description of the binary classification tree. Therefore, the tree predicts class 1 if the Boolean equation

$$L = P_1 \vee P_2 \vee \cdots \vee P_l$$

is true, where

$$P_i = C_1^i \wedge C_2^i \wedge \cdots \wedge C_{k^i}^i.$$



(a) a binary classification tree (b) a logic tree

Figure A.1. Equivalence of trees: the binary classification tree and the Logic Tree predict exactly the same outcome for any set of values of A, B, C, D.

Hence every binary classification tree can immediately be written as a Boolean equation in DNF. For example, the tree in Figure A.1(a) predicts class 1 if the Boolean equation

$$L = (C^c \wedge A \wedge B) \vee (C \wedge D^c \wedge A \wedge B) \vee (C \wedge D)$$

is true.

However, it is not obvious whether every Boolean expression in DNF can directly be expressed as a classification tree. The reason for this is that in a classification tree the first knot is part of every path. Using De Morgan's rules and standard Boolean operations, it can be shown that a classification tree can be constructed from every Boolean expression. However, these classification trees often result in awkward looking constructs, much more complex than the simple logic trees constructed from the Boolean expression. For example, for the tree in Figure A.1(a), we have

$$L = (C^c \wedge A \wedge B) \vee (C \wedge D^c \wedge A \wedge B) \vee (C \wedge D) \equiv (A \wedge B) \vee (C \wedge D) = L'.$$

This means that the classification tree in Figure A.1(a) corresponds to the simple Boolean expression $(A \wedge B) \vee (C \wedge D)$, which can be displayed as very simple logic tree, as done in Figure A.1(b) (which, incidentally, is also in DNF). We feel that the simplicity of logic trees is one of their big attractions.

B. PRACTICAL ASPECTS OF SIMULATED ANNEALING

For simulated annealing, in addition to the previously described scoring functions and the move set, we have to specify a selection probability for the moves, an acceptance function, and a cooling scheme. Further, when we implemented the simulated annealing

algorithm for the logic regression methodology, some practical aspects had to be considered how to run the search. This section lists what we believe are the most important issues.

- At every stage of the algorithm there is a temperature T (see Section 3.3 and below). Suppose that the current state of the algorithm has score ϵ_{old}, and the new proposed state has score ϵ_{new}, where smaller scores are better. The new state is accepted with probability $p = \min\left\{1, \exp\left(\left[\epsilon_{\text{old}} - \epsilon_{\text{new}}\right]/T\right)\right\}$ (note that Markov chain Monte Carlo Metropolis–Hastings can be seen as simulated annealing algorithm where T remains constant). See Otten and van Ginneken (1989) and van Laarhoven and Aarts (1987) for a discussion of this acceptance probability. In general, the selection probability for the moves affects the performance of the annealing algorithm. For logic regression, we found that it is beneficial to give the highest preference to the moves that alternate the leafs. Alternating operators can change the prediction of the logic tree under consideration considerably, and we usually assign those moves a somewhat lower probability.
- There are two fundamentally different ways how to implement a simulated annealing algorithm. One way is to slightly decrease the temperature at each step in the algorithm. The other way is to keep the temperature constant for a certain number of iterations, and then decrease the temperature by a somewhat larger amount. The performances of both algorithms have been studied (Otten and van Ginneken 1989; van Laarhoven and Aarts 1987), and in general one cannot claim that one way of running the chain is superior to the other. However, we find that monitoring the convergence process is more straightforward using a sequence of homogeneous Markov chains (runs at constant temperature).
- In the beginning of a simulated annealing run with high temperatures, virtually all proposed moves are accepted. Towards the end, almost every proposed move is rejected. Somewhere in between is the "crunch time," where we want to spend most of the computing time. To speed up the simulated annealing and avoid spending too much time either at the beginning or the end of the run, we implemented the following features:

  – Running a Markov chain at fixed temperature, we keep track of how many moves have been accepted. If this number reaches a predetermined threshold, we exit the Markov chain (even though the number of iterations specified has not been reached) and lower the temperature to its next value. This avoids spending too much time at the beginning of the run in "random models." Later in the annealing algorithm, we will not be able to reach this threshold and the Markov chains run for their full lengths. The threshold typically is between 1% and 10% of the prespecified number of iterations for a chain at a fixed temperature.
  – We have to specify a lowest temperature before starting the simulated annealing run. Several criteria can be considered to exit the simulated annealing run early when the search virtually has been finished, for example the case when no moves

were accepted for a substantial number of consecutive chains. This avoids running the algorithm all the way to its end although no improvement can be achieved anymore. It also allows setting the lowest temperature arbitrarily low, since the exit criteria can be chosen independently of the lowest temperature considered.

- To implement a simulated annealing run, we have to specify a temperature scheme. That means we have to choose the starting (highest) temperature, the finishing (lowest) temperature and the cooling scheme, which also determines the total number of chains we run at constant temperatures. In making this decision, there is usually some trial and error involved, since in practice the cooling scheme depends on the data we are analyzing. The theory of simulated annealing is thoroughly discussed, for example, in the books by van Laarhoven and Aarts (1987) and Otten and van Ginneken (1989), stating conditions for the simulated annealing run under which we can guarantee to find an optimal scoring state. One obvious problem is that we cannot run chains of length infinity as required, the optimal state may not be reached in practice. We need to make sure that the individual chains we run come close to their limiting distributions, and cool sufficiently slowly such that this can be achieved with a reasonable number of iterations in the chain. The choices for the parameters we pick therefore influence each other. We already explained above that picking the highest and lowest temperature is not a problem. We want the starting temperature high enough such that the first chains are essentially random walks This can be controlled by monitoring the acceptance rate. The lowest temperature is chosen such that the above described exit criterion terminates the chain. Depending on the size of the dataset, we usually choose for the exit criterion the number of chains without acceptance between 10 and 20. The temperatures are usually lowered in equal increments on a $\log_{10}$ scale. The number of chains between two subsequent powers of 10 depend on the data as well; usually, the number is between 25 and 100. This translates to decreasing the temperature between homogeneous chains by a factor between 0.91 and 0.98. The lengths of the individual chains for the data we looked at so far have been between 10,000 and 100,000. The search for a single best model usually takes only a few minutes on a Pentium processor, the randomization tests (depending on the number of permutations carried out) usually require several hours of CPU time.
- Every logic tree has a finite number of neighbors. Especially toward the end of the run at low temperatures, very few moves get accepted. Because simulated annealing is a probabilistic search, a move might get rejected several times before it is accepted. The worst "bottle neck" in terms of computing time is the evaluation of the logic trees, that is, deriving the values of their underlying Boolean equation from the leaves for each case. Because the acceptance of a move, given the temperature and the score of the current model, depends only on the score of the proposed model, we implemented a subroutine that keeps track of all states visited and their scores in a table. Therefore, for the decision whether or not to accept a certain move from a given state, the trees of the proposed model have to be evaluated only once, which

speeds up the search dramatically at lower temperatures. After a move is accepted, the old table is flushed and a new table is generated.

- In theory, trees of any size can be grown, but considering that in our applications we want to be able to interpret these models, it makes sense to limit the tree sizes to a reasonable number of leaves. We usually limit the number of leaves to a maximum of 8 or 16 per tree. As previously noted, we also limit the number of trees we allow. In the case studies we carried out, optimal logic regression models typically had between one and three logic trees. However, we usually allow up to five trees at the beginning of our search for good models.

ACKNOWLEDGMENTS

We thank Richard Kronmal and Robyn McClelland for providing us with the data used in Section 5.2 and the permission to use those data in this article. We thank the GAW organizers for permission to use their data in this article. Ingo Ruczinski and Charles Kooperberg were supported in part by NIH grant CA74841. Michael LeBlanc was supported by NIH grant CA90998. All authors were supported in part by NIH grant CA53996. The GAW workshop is supported by NIH grant GM31575.

[Received June 2001. Revised September 2002.]

REFERENCES

Amit Y., and Geman, D. (1997), "Shape Quantization and Recognition with Randomized Trees," *Neural Computation*, 9, 1545–1588.

Anthony, M. (2001), "Discrete Mathematics of Neural Networks: Selected Topics," *SIAM Monographs on Discrete Mathematics and Applications*, 8.

Apte, C., Damerau, F., and Weiss, S. M. (1994), "Towards Language Independent Automated Learning of Text Categorisation Models," *Research and Development in Information Retrieval*, 23–30.

Apte, C., and Weiss, S. M. (1997), "Data Mining with Decision Trees and Decision Rules," *Future Generation Computer Systems*, 13, 197–210.

Bala, J., DeJong, K., and Pachowicz, P. (1991), "Learning Noise Tolerant Classification Procedures by Integrating Inductive Learning and Genetic Algorithms," in *Proceedings of the First International Workshop on Multistrategy Learning MSL-91*, pp. 316–323.

Bayardo, R. J. Jr., and Agrawal, R. (1999), "Mining the Most Interesting Rules," *Knowledge Discovery and Data Mining*, 145–154.

Breiman, L. (1996), "Bagging Predictors," *Machine Learning*, 24, 123–140.

——— (1999), "Random Forests—Random Features," Technical report, Department of Statistics, University of California Berkeley.

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

Bryan, R. N., Manolio, T. A., Schertz, L. D., Jungreis, C., Poirier, V. C., Elster, A. D., and Kronmal, R. A. (1994), "A Method for Using MR to Evaluate the Effects of Cardiovascular Disease on the Brain: the Cardiovascular Health Study", *American Journal of Neuroradiology*, 15, 1625–1633.

Buntine, W. (1992), "Learning Classification Trees," *Statistics and Computing*, 2, 63–73.

Chipman, H., George, E., and McCulloch, R. (1998), "Bayesian CART Model Search" (with discussion), *Journal of the American Statistical Association*, 93, 935–960.

——— (2002), "Bayesian Treed Models," *Machine Learning*, 48, 299–320.

Clark, P., and Niblett, T. (1989), "The CN2 Induction Algorithm," *Machine Learning*, 3, 261–283.

Cohen, W. (1995), "Fast Effective Rule Induction," *International Conference on Machine Learning*, 115–123.

Cohen, W., and Singer, Y. (1999), "A Simple, Fast, and Effective Rule Learner," in *Proceedings of Annual Conference of American Association for Artificial Intelligence*, 335–342.

Daly, M. J., Rioux, J. D., Schaffner, S. F., Hudson, T. J., and Lander, E. S. (2001), "High Resolution Haplotype Structure in the Human Genome," *Nature Genetics*, 29, 229–232.

Denison, D., Mallick, B., and Smith, A. F. M. (1998), "A Bayesian CART Algorithm," *Biometrika*, 85, 363–377.

Deshpande, A. S., and Triantaphyllou, E. (1998), "A Greedy Randomized Adaptive Search Procedure," *Mathematical Computer Modelling*, 27, 75–99.

Dietterich, T. G. (1999), "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," *Machine Learning*, 40, 139–158.

Domingos, P. (1996), "Linear-Time Rule Induction," *Knowledge Discovery and Data Mining*, 96–101.

Fleisher, H., Tavel, M., and Yeager, J. (1983), "Exclusive-OR Representations of Boolean Functions," *IBM Journal of Research and Development*, 27, 412–416.

Fleisher, H., Giraldi, J., Martin, D. B., Phoenix, R. L., and Tavel, M. A. (1985), "Simulated Annealing as a Tool for Logic Optimization in a CAD Environment," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 203–205.

Frean, M. (1990), "Small Nets and Short Paths: Optimising Neural Computation," PhD thesis, University of Edinburgh, Center for Cognitive Science.

Freund, Y., and Schapire, R. (1996), "Experiments With a New Boosting Algorithm," in *Proceedings of the 13th International Conference on Machine Learning*, pp. 148–156.

Fried, L. P., Borhani, N. O., Enright, P., Furberg, C. D., Gardin, J. M., Kronmal, R. A., Kuller, L. H., Manolio, T. A., Mittelmark, M. B., Newman, A. B., O'Leary, D. H., Psaty, B., Rautaharju, P., Tracy, R. P., and Weiler, P. G. (1991), "The Cardiovascular Health Study: Design and Rationale," *Annals of Epidemiology*, 1, 263–276.

Friedman, J. H. (1991), "Multivariate Adaptive Regression Splines" (with discussion), *The Annals of Statistics*, 19, 1–141.

Friedman, J. H., and Fisher, N. I. (1999), "Bump Hunting in High-Dimensional Data," *Statistics and Computing*, 9, 123–162.

Giordana, A., and Saitta, L. (1993), "REGAL: An Integrated System for Learning Relations Using Genetic Algorithms," in *Proceedings of the Second International Workshop on Multistrategy Learning (MSL-93)*, pp. 234–249.

Gray, D. L., and Michel, A. N. (1992), "A Training Algorithm for Binary Feedforward Neural Networks," *IEEE Transactions on Neural Networks*, 3, 176–194.

Hong, S. (1997), "R-MINI: An Iterative Approach for Generating Minimal Rules from Examples," *IEEE Transactions on Knowledge and Data Engineering*, 9, 709–717.

Hong, S., Cain, R. G., and Ostapko, D. L. (1974), "MINI: A Heuristic Approach for Logic Minimization," *IBM Journal of Research and Development*, 18, 443–458.

Jordan, M. I., and Jacobs, R. A. (1994), "Hierarchical Mixtures of Experts and the EM Algorithm," *Neural Computation*, 6, 181–214.

Kooperberg, C., Ruczinski, I., LeBlanc, M. L., and Hsu, L. (2001), "Sequence Analysis using Logic Regression," *Genetic Epidemiology*, 21, Supplement 1, 626–631.

Liu, B., Hsu, W., and Ma, Y. (1998), "Integrating Classification and Association Rule Mining," *Proceedings of International Conf on Knowledge Discovery and Data Mining*, pp. 80–86.

Lucek, P. R., and Ott, J. (1997), "Neural Network Analysis of Complex Traits," *Genetic Epidemiology*, 14, 1101–1106.

Lutsko, J. F., and Kuijpers, B. (1994), "Simulated Annealing in the Construction of Near-Optimal Decision Trees," in *Selecting Models from Data: AI&Statistics IV*, New York: Springer, pp. 453–462.

McClelland, R. (2000), "Regression Based Variable Clustering for Data Reduction," PhD thesis, University of Washington, Department of Biostatistics.

Mehta, M., Rissanen, J., and Agrawal, R. (1995), "MDL-Based Decision Tree Pruning," in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining KDD95*, pp. 216–221.

Mehta, M., Agrawal, R., and Rissanen, J. (1996), "SLIQ: A Fast Scalable Classifier for Data Mining," *Extending Database Technology*, 18–32.

Meretakis, D., and Wuthrich, B. (1999), "Extending Naive Bayes Classifiers Using Long Itemsets," *Knowledge Discovery and Data Mining*, 165–174.

Michalski, R., Mozetic, I., Hong, J., and Lavrac, N. (1986), "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains," *Proceedings of the AAAI*, pp. 1041–1045.

Murthy, S. K., and Salzberg, S. (1995), "Decision Tree Induction: How Effective is the Greedy Heuristic?," *Knowledge Discovery and Data Mining*, 222–227.

Otten, R. H., and Ginneken, L. P. (1989), *The Annealing Algorithm*, Boston: Kluwer Academic Publishers.

Quinlan, J. R. (1986), "Induction of Decision Trees," *Machine Learning*, 1, 81–106.

——— (1992), "Learning with Continuous Classes," *5th Australian Joint Conference on Artificial Intelligence*, 343–348.

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

——— (1996), "Bagging, Boosting and C4.5", in *Proceedings of AAAI'96 National Conference on Artificial Intelligence*, pp. 725–730.

Ruczinski, I. (2000), "Logic Regression and Statistical Issues Related to the Protein Folding Problem," PhD thesis, University of Washington, Department of Statistics.

Sutton, C. (1991), "Improving Classification Trees with Simulated Annealing," in *Proceedings of the 23rd Symposium on the Interface*, ed. E. Kazimadas, Fairfax, VA: Interface Foundation of North America.

Teng, E. L., and Chui, H. C. (1987), "The Modified Mini-Mental State (3MS) Examination," *Journal of Clinical Psychiatry*, 48, 314–318.

Thimm, G., and Fiesler, E. (1996), "A Boolean Approach to Construct Neural Networks for Non-Boolean Problems," *Proceedings of the Eighth International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 458–459.

Tibshirani, R., and Knight, K. (1999), "Model Search and Inference by Bootstrap 'Bumping'," *Journal of Computational and Graphical Statistics'*, 8, 671–686.

Torgo, L. (1995), "Data Fitting with Rule-based Regression," in *Proceedings of the 2nd International Workshop on Artificial Intelligence Techniques (AIT'95)*, eds. J. Zizka, and P. Brazdil, Brno, Czech Republic.

Torgo, L., and Gama, J. (1996), "Regression by Classification," *Brazilian Symposium on Artificial Intelligence*, 51–60.

Vafaie, H., and DeJong, K. (1991), "Improving the Performance of a Rule Induction System Using Genetic Algorithms," in *Proceedings of the First International Workshop on Multistrategy Learning MSL-91*, pp. 305–315.

van Laarhoven, P. J., and Aarts, E. H. (1987), *Simulated Annealing: Theory and Applications*, Boston: Kluwer Academic Publishers.

Webb, G. (2000), "Efficient Search for Association Rules," in *Proceedings of the sixth International Conference on Knowledge Discovery and Data Mining KDD 2000*, 99–107.

——— (2001), "Discovering Associations with Numeric Variables," in *Proceedings of the seventh International Conference on Knowledge Discovery and Data Mining KDD 2001*, pp. 383–388.

Weiss, S. M., and Indurkhya, N. (1993a), "Optimized Rule Induction," *IEEE Expert*, December, 61–69.

——— (1993b), "Rule-Based Regression," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1072–1078.

——— (1995), "Rule-Based Machine Learning Methods for Functional Prediction," *Journal of Artificial Intelligence Research*, 3, 383–403.

Wijsman, E. M., Almasy, L., Amos, C. I., Borecki, I., Falk, C. T., King, T. M., Martinez, M. M., Meyers, D., Neuman, R., Olson, J. M., Rich, S., Spence, M. A., Thomas, D. C., Vieland, V. J., Witte, J. S., MacCluer, J. W. (2001), "Analysis of Complex Genetic Traits: Applications to Asthma and Simulated Data," *Genetic Epidemiology*, 21, Supplement 1.

Witte, J. S., and Fijal, B. A. (2001), "Introduction: Analysis of Sequence Data and Population Structure," *Genetic Epidemiology*, 21, Supplement 1, pp. 600–601.

Wnek, J., and Michalski, R. (1994), "Comparing Symbolic and Subsymbolic Learning: Three Studies," *Machine Learning: A Multistrategy Approach*, eds. R. Michalski and G. Tecuci, 4, San Francisco: Morgan-Kaufmann.

Zhang, J., and Michalski, R. S. (1989), "Rule Optimization via SG–Trunc Method," *European Working Session on Learning*, 251–262.