Part I

Logic Regression

Chapter 1

INTRODUCTION TO LOGIC REGRESSION

1.1 Introduction: Motivation and Goals

Regression is arguably the most important tool in the field of Statistics to analyze data and make inference about associations between predictors and response. However, in most regression problems a model is developed that only relates the predictors as they are ("main effects") to the response. Although interactions between predictors are considered as well, those interactions are usually kept very simple (two- to three-way interactions at most). But often, especially when all predictors are binary (0-1, on-off, true-false, . . .), the interaction of many predictors is what causes the differences in response. For example, in a recent publication by Lucek and Ott [28], the authors are concerned about analyzing the relationship between disease loci and complex traits. In the introduction of the paper, Lucek and Ott recognize the importance of interactions between loci and potential shortcomings of methods that do not take those interactions appropriately into account:

"*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 state 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 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 in general), 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, 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 gene chip data, where one is interested in finding an association between gene expressions and diseases, for example certain types of cancer.

The first part of this thesis contains 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**. We think that Logic Regression may be a tool that fills a gap in the regression and classification methodology. Logic rules, especially the rules in Disjunctive Normal Form [13], play a key role in many fields covered by the engineering and machine learning literature. The similarity between these methods and our methodology is that they all partition the search space by investigating logic rules. However, the methods from the engineering and machine learning literature only cover classification problems in general, and do not include objective functions such as the deviance or likelihood in generalized regression models to our knowledge. Also, almost all of those methods aim for being computationally simplistic. These include forming logic rules growing one rule at a time [31], constructing decision trees and deriving the logic rules modifying its paths [36], deriving rules in a greedy fashion using swaps [52], among others. While computational simplicity is a worthwhile goal to aim for, all the above mentioned methods do not guarantee the minimality of the rules derived [22]. Most learning systems even make the "noise free" data assumption, which prevents them from being applied to real world learning problems [54]. Making it computationally feasible to search through the entire space of models without compromising the desire for minimality, 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 the before mentioned void in the regression and classification methodology. Analyzing data from heart attack patients admitted to the Medical Center at the University of California at San Diego, Breiman et al [6] indeed acknowledged the need for such an algorithm:

"*Given the implicit concern CART has with relationships among variables, it may have been preferable to use a variable selection scheme (possibly CART itself) that looked directly at 'interactions' and not merely 'main effects'.*"

In the remainder of Chapter 1 we introduce the basic terminology and rules of Boolean Algebra, and define and discuss "Logic Trees", a basic construct of Logic Regression. To find the good logic combinations of predictors among the huge number of possible Boolean expressions, we rely on search algorithms that we discuss in Chapter 2. In Chapter 3 we introduce and discuss the models for Boolean regression that we consider in this thesis. Chapter 4 explains some algorithmic details, technicalities, and tricks we use when searching for good models. We introduce some tools in Chapter 5 for statistical inference and model selection, designed to avoid over fitting. In Chapter 6 we use the Logic Regression methodology in two real data case studies.

1.2 *The Basics of Logic Regression*

1.2.1 Terminology in Boolean Logic

Our task is to find those combinations of binary variables that have the highest predictive power for the response variable. These combinations are Boolean Logic Expressions, and since the predictors are binary, any of those combinations of predictors will be binary. The data shown on page 6 is a simulated binary data set. Can we find a classification rule that correctly assigns a case to either class 0 ($Y = 0$) or class 1 ($Y = 1$)? In this case, the correct solution for this simulated data set is the Boolean Equation

$$L = [(X_1 \wedge X_2) \vee (X_3^c \wedge X_4)] \wedge (X_5 \vee X_6^c \vee X_7) \tag{1.1}$$

We assume familiarity with the basic concepts of Boolean Logic, but briefly clarify the terminology that will be used throughout this part of the thesis.

Values

The only two values that are used are 0 and 1 (False and True, Off and On, . . .)

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).

Response	Predictors									
Y	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
0	1	0	0	0	0	0	0	1	0	0
0	1	0	0	1	0	1	0	0	1	0
0	0	1	0	0	0	0	1	1	0	1
1	1	1	1	1	1	1	1	1	0	1
0	1	0	0	1	0	1	0	0	1	0
1	1	1	1	0	1	1	0	1	0	0
0	1	1	0	1	0	1	0	1	1	0
1	1	1	0	1	1	0	0	0	1	0
1	1	1	0	0	1	1	0	1	0	1
0	0	1	0	1	0	1	0	0	1	1
0	0	0	1	1	1	0	0	1	1	0
1	0	1	0	1	0	0	0	0	0	0
0	1	0	0	1	0	1	0	1	0	0
0	0	1	0	0	0	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0	1
0	0	1	1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1	1	1	0
0	0	0	1	0	1	1	1	0	1	0
0	0	0	0	0	1	0	1	0	1	1
1	0	1	0	1	1	1	1	0	1	1
0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	1	1	0	0	0	1	0
0	1	0	0	0	0	1	0	1	1	1
0	1	0	0	1	0	1	0	0	1	0
0	1	0	1	0	1	0	0	1	1	0
0	0	1	0	0	1	0	1	0	1	1
1	1	1	0	1	1	1	0	1	0	1
0	1	0	1	1	0	1	0	1	1	1
1	0	0	0	1	1	1	1	0	1	1
1	1	1	1	1	0	1	1	1	1	0

Expressions

The combination of values and variables with operators results in expressions. For example,

$$X_1 \wedge X_2^c \tag{1.2}$$

is a logic (Boolean) expression built from two variables and two operators. X_1 and X_2^c are called the operands of $\wedge$.

Equations

An equation assigns a name to an expression. For example, using

$$L = X_1 \wedge X_2^c \tag{1.3}$$

we can refer to the expression $X_1 \wedge X_2^c$ by simply stating L.

1.2.2 Rules and Laws in Boolean Algebra

Here we introduce the Boolean algebra precedence rules and give a summary of the most important laws in Boolean algebra, taken from Peter Wentworth's on-line tutorial "Boolean Logic and Circuits" [53]. Here, L, L_1, L_2, L_3 stand for any Boolean expression.

Boolean Algebra Precedence Rules

In Boolean algebra, as in any other algebra, brackets straightforwardly determine some of the precedence of the terms involved in an expression. In addition, the rules below determine the precedence of the operators in Boolean expressions. They allow to remove or insert brackets in Boolean expressions.

1. First apply the complement operators (^{c}),
2. Second apply the "AND" operators ($\wedge$),
3. Third apply the "OR" operators ($\vee$),
4. Apply the operators at the same precedence from left to right.

Hence, $L_1 \wedge L_2^c$ means $L_1 \wedge (L_2)^c$, but not $(L_1 \wedge L_2)^c$.

Boolean Algebraic Laws

The following laws are stated here without proofs. Their validity can easily be shown for example by considering all possibilities of 0/1 values that the Boolean variables can adopt.

- associative operators

$$(L_1 \wedge L_2) \wedge L_3 = L_1 \wedge (L_2 \wedge L_3) \tag{1.4}$$

$$(L_1 \vee L_2) \vee L_3 = L_1 \vee (L_2 \vee L_3) \tag{1.5}$$

- commutative operators

$$L_1 \wedge L_2 = L_2 \wedge L_1 \tag{1.6}$$

$$L_1 \vee L_2 = L_2 \vee L_1 \tag{1.7}$$

- double complement law

$$L = (L^c)^c \tag{1.8}$$

- identity laws

$$L \wedge 1 = L \tag{1.9}$$
$$L \vee 0 = L \tag{1.10}$$

- null laws

$$L \wedge 0 = 0 \tag{1.11}$$
$$L \vee 1 = 1 \tag{1.12}$$

- complement laws

$$L \wedge L^c = 0 \tag{1.13}$$
$$L \vee L^c = 1 \tag{1.14}$$

- idempotent laws

$$L \wedge L = L \tag{1.15}$$
$$L \vee L = L \tag{1.16}$$

- distributive laws

$$L_1 \vee (L_2 \wedge L_3) = (L_1 \vee L_2) \wedge (L_1 \vee L_3) \tag{1.17}$$
$$L_1 \wedge (L_2 \vee L_3) = (L_1 \wedge L_2) \vee (L_1 \wedge L_3) \tag{1.18}$$

- de Morgan's laws

$$(L_1 \vee L_2)^c = (L_1^c \wedge L_2^c) \tag{1.19}$$
$$(L_1 \wedge L_2)^c = (L_1^c \vee L_2^c) \tag{1.20}$$

- absorption laws

$$L_1 \wedge (L_1 \vee L_2) = L_1 \tag{1.21}$$
$$L_1 \vee (L_1 \wedge L_2) = L_1 \tag{1.22}$$

1.2.3 Representations of Logic Statements

The way to represent logic statements is not unique. Below, we briefly discuss the two common choices, and introduce two forms of what we will call “Logic Trees”.

Logic Terms (LTE)

As described in the previous section, the most straightforward way to denote logic statements (also called Boolean equations or Boolean expressions) such as *A and B* or such as *C or D but not E* is by using the operators $\vee$ (“or”), $\wedge$ (“and”), ^{c} (“not”), and using brackets for *every* operator $\wedge$ or $\vee$. An example for such a **Logic Term** (or expression) is

$$\{(A \wedge B^c) \wedge [(C \wedge D) \vee (E \wedge (C^c \vee F))]\}. \tag{1.23}$$

The outer (curly) brackets can be omitted. We will use this example throughout the remainder of this section.

Disjunctive Normal Forms (DNF)

A widespread notation of logic statements in the Engineering and Computer Science literature is the **Disjunctive Normal Form**, which is a special case of a Boolean expression (see for example [13]). A Disjunctive Normal Form is a Boolean Expression, expressed as $\vee$-combinations of $\wedge$-terms. For example, the Logic Term in equation (1.23) can be rewritten in Disjunctive Normal Form as

$$(A \wedge B^c \wedge C \wedge D) \vee (A \wedge B^c \wedge E \wedge C^c) \vee (A \wedge B^c \wedge E \wedge F) \tag{1.24}$$

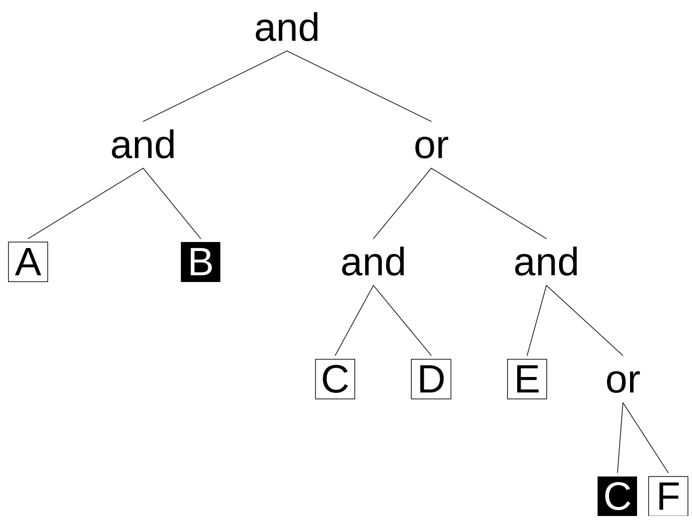


Figure 1.1: The Logic Tree representing the Logic Term in equation (1.25).

Logic Trees (LTR)

Using brackets, any Boolean expression can be generated by iteratively combining two leaves, a leaf and a Boolean expression, or two Boolean expressions. For example, the Logic Term in equation (1.23) can be generated as outlined below:

$$\underbrace{\underbrace{(A \wedge B^c)}_{1} \wedge [\underbrace{\underbrace{(C \wedge D)}_{2} \vee \underbrace{(E \wedge \underbrace{(C^c \vee F)}_{3})}_{4}]}_{5}}_{6} \tag{1.25}$$

The above expression can be understood as an "and" statement, generated from the Boolean expressions $A \wedge B^c$ and $(C \wedge D) \vee (E \wedge (C^c \vee F))$. The latter can be understood as an "or" statement, generated from the Boolean expressions $C \wedge D$ and $E \wedge (C^c \vee F)$, and so on. This enables us to represent any Logic Term in a binary tree format. The **Logic Tree** for the Logic Term in equation (1.23) is shown in Figure 1.1. White letters on black background denote the conjugate of the letter. The evaluation of the tree as a logic statement for a particular case occurs in a "bottom-up" fashion.

We use the following terminology and rules for Logic Trees (similar to the terminology used by Breiman et al [6] for classification trees):

- The location for each element (letter, conjugate letter or operators $\wedge$ and $\vee$) in the tree is a knot.
- Each knot has either zero or two sub-knots.
- The two sub-knots of a knot are called its children, the knot itself is called the parent of the sub-knots.
- 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).

Since the representation of a Boolean expression as a Logic Term is not unique, neither is the representation as a Logic Tree. For example, the Boolean expression in equation (1.23) or (1.25) can also be written as

$$((A \wedge B^c) \wedge (C \wedge D)) \ \vee \ ((A \wedge B^c) \wedge (E \wedge (C^c \vee F))) \tag{1.26}$$

The former leads to the tree as shown in Figure 1.1, the latter leads to the tree as shown in Figure 1.2. This is not simply a matter of the complexity of the Boolean expression, since it can also be written as

$$A \wedge \{B^c \wedge [(C \wedge D) \vee (E \wedge (C^c \vee F))]\}, \tag{1.27}$$

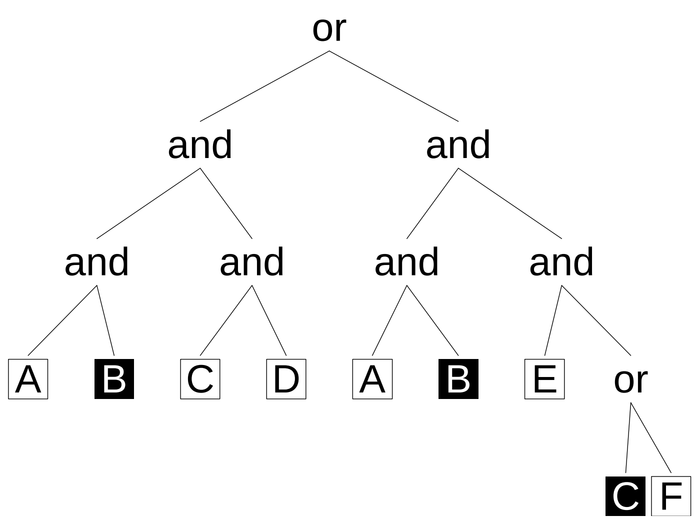


Figure 1.2: The Logic Tree representing the Logic Term in equation (1.26).

which would lead to yet another tree. A Boolean expression can be written in different ways as a Logic Term, but each Logic Term corresponds to exactly one Logic Tree.

More details on how to construct a Logic Tree from a Logic Term are provided in Section 1.2.4.

Generalized Logic Trees (GLT)

We can introduce a hierarchy of the leaves in a Logic Tree by assigning a number to each leaf (which we call the depth of the leaf), counting how often the types of operators (links) change on the path between its parent (registered as change number one) and the root. For example, leaf "D" in the tree in Figure 1.1 has an "and" as parent, which has an "or" as parent etc. The entire sequence of links between leaf "D" and the root is $\wedge \longrightarrow \vee \longrightarrow \wedge$, which means there are three link changes (counting the parent of the leaf as change number one), and hence leaf "D" has depth 3. The chart below shows the depth for each leaf in the Logic Tree of Figure 1.1.

leaf	A	B^c	C	D	E	C^c	F
depth	1	1	3	3	3	4	4

Using the depth information of its corresponding Logic Tree, it is possible to display every Logic Term as a **Generalized Logic Tree**. On each "level" of the tree there is only one type of link (either $\wedge$ or $\vee$), and leaves of the same depth. Note that those trees are not binary in general, and hence the Generalized Logic Trees themselves are neither sub- nor superset of the Logic Trees as introduced in Section 1.2.3. Loosely speaking, a Logic Tree is to a Logic Term what a Generalized Logic Tree is to a Boolean expression with as many parenthesis as possible removed. Figure 1.3 shows the Generalized Logic Tree for the Boolean expression in equation (1.23).

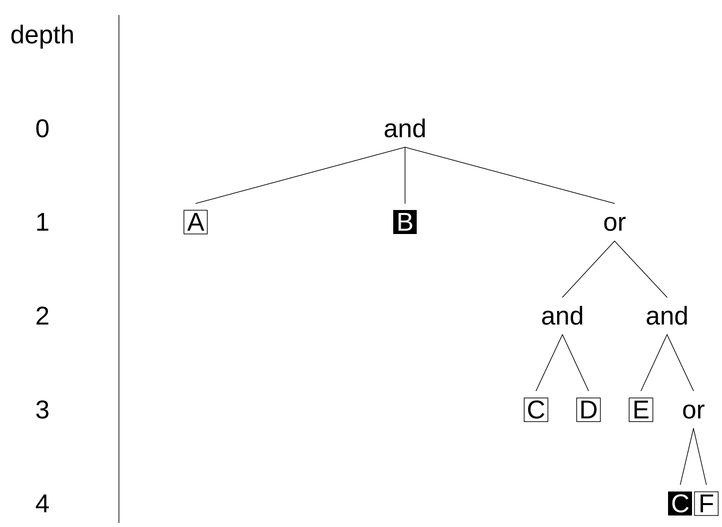


Figure 1.3: The Generalized Logic Tree representing the Logic Term in equation (1.23).

1.2.4 Equivalence of Logic Terms, Disjunctive Normal Forms, Logic Trees and Generalized Logic Trees

At first glance it is not clear whether or not the above stated representations of logic expressions are equivalent in the sense that the classes of logic expressions they represent are the same. We show that these classes are indeed the same, by establishing the following relations between classes:

$$\text{Class(LTE)} \subseteq \text{Class(LTR)} \subseteq \text{Class(DNF)} \subseteq \text{Class(GLT)} \subseteq \text{Class(LTE)} \qquad (1.28)$$

The above abbreviations are:

LTE	:	Logic Term
LTR	:	Logic Tree
DNF	:	Disjunctive Normal Form
GLT	:	Generalized Logic Tree

Class(LTE) ⊆ Class(LTR)

In the previous section we already outlined how to construct a Logic Tree from a given Logic Term. The steps are the following:

1. Combine all pairs of letters in brackets to initial subtrees.
2. Combine single letters with subtrees in brackets to new subtrees.
3. Combine subtrees.

The steps of the construction of the Logic Tree for the Logic Term in equation (1.23) are illustrated in Figure 1.4.

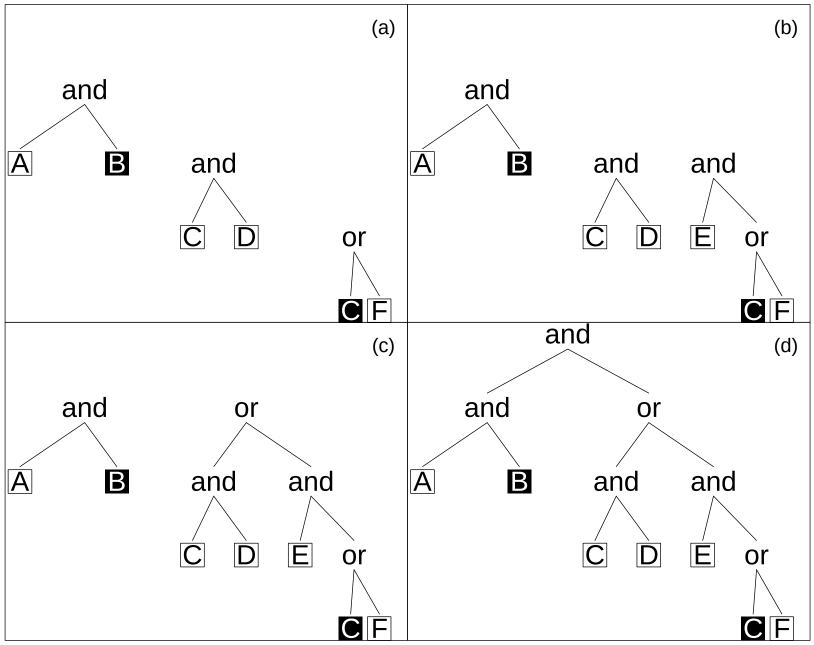


Figure 1.4: Illustration of the construction of the Logic Tree in Figure 1.1 from the Logic Term in equation (1.25).

Class(LTR) $\subseteq$ *Class(DNF)*

In this section we establish that the class of logic expressions representable by Logic Trees is a subset of the class of logic expressions representable by Disjunctive Normal Forms. We show this by constructing an algorithm, that generates the Disjunctive Normal Form of a boolean expression from its Logic Tree. The algorithm requires that the knots are numbered as shown in Figure 1.5, starting with 1 at the root, and with the left most knot on each level of the tree being numbered as a power of 2.

A Logic expression L in Disjunctive Normal Form is a $\vee$-combination of $\wedge$-terms. This means that L is true if (at least) one of the $\wedge$-terms is true, which is the case if within this $\wedge$-term all predictors are true. Consider a Logic Tree with leaves $\{X_1, \ldots, X_k\}$ (from "left to right" in the tree, independent of the level, not in the sequence as indicated in Figure 1.5). The idea of the algorithm, to obtain the Disjunctive Normal Form of the Boolean

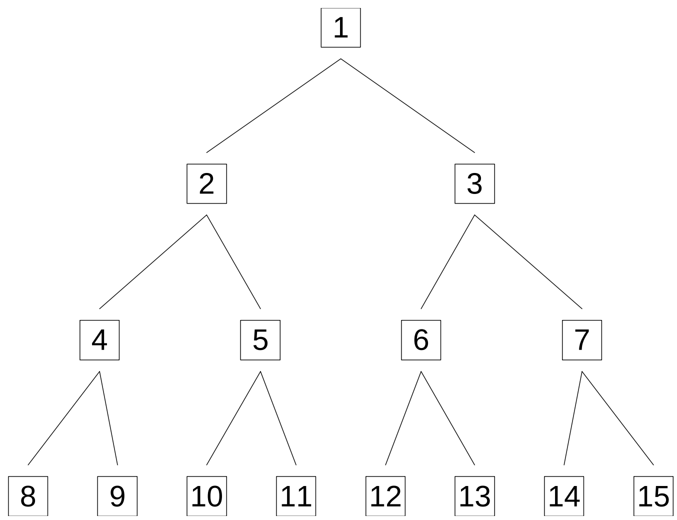


Figure 1.5: The labeling of knots in Logic Trees.

expression from this Logic Tree, is to split the set of leaves into subsets until we have sets of leaves that represent the $\wedge$-terms in the Disjunctive Normal Form. We sequentially check the Logic Tree for $\vee$-operators, starting with the root. There are two subtrees of the root, with the left and right child of the root as their respective roots. These subtrees have the leaves $X_1, \ldots, X_j$ and $X_{j+1}, \ldots, X_k$ respectively. If the root is an $\vee$-operator, it is sufficient and necessary for the entire tree to be true that at least one subtree is true. If the root is a $\wedge$-operator, both subtrees have to be true. In the former case we will consider the set of leaves $\{X_1, \ldots, X_j\}$ and $\{X_{j+1}, \ldots, X_k\}$ independently, and check what kind of operators knot 2 and knot 3 are. In the latter case, check knot 2: If it is an $\wedge$, than both subtrees of knot 2 and the subtree with knot 3 as root have to be true for the entire tree to be true. If knot 2 is a $\vee$, than it is necessary and sufficient if at least one subtree of knot 2 is true in addition to the subtree with knot 3 as root. Hence, if knot 2 is an $\vee$, we have to consider the sets $\{X_1, \ldots, X_l, X_{j+1}, \ldots, X_k\}$ and $\{X_{l+1}, \ldots, X_j, X_{j+1}, \ldots, X_k\}$. In other words, at every knot that is an $\vee$ operator we split the appropriate set of leaves, until we have considered the last $\vee$. Then, if for any of those subsets all leaves are true, the entire tree is true.

Below is a rather cryptic version of the algorithm in a pseudo language.

```
WRITE all leaves -> set

DO j=1,maxknot

   IF (SYMBOL(j)==OR)
      SPLIT appropriate set
            -> subsets (children OF j LEFT RIGHT)

ENDDO

WRITE subsets -> dnf
```

We clarify the idea of this algorithm in our example, using the tree in Figure 1.1. Using the outlined algorithm, the initial set is

```
A B C D E F G
```

The first $\vee$ is at node 3, and we split the set into two subsets, distinguished by the children of node 3.

```
A B C D
A B E F G
```

There is one more $\vee$ at node 15, hence we split the latter of the two above subsets, and get

A B C D

A B E F

A B E G

Hence the Disjunctive Normal Form for the tree in Figure 1.1 is

$$(A \wedge B \wedge C \wedge D) \vee (A \wedge B \wedge E \wedge F) \vee (A \wedge B \wedge E \wedge G) \tag{1.29}$$

Class(DNF) $\subseteq$ *Class(GLT)*

To construct a Generalized Logic Tree from a Boolean expression in Disjunctive Normal Form is straightforward. The root is a $\vee$ and its children are $\wedge$s. The number of children the root has is simply the number of $\wedge$-terms in the Disjunctive Normal Form. All children of the $\wedge$s in the tree are leaves, representing the predictors in the respective terms in the Disjunctive Normal Form. Figure 1.6 shows a Generalized Logic Tree for the Boolean expression in (1.29).

Class(GLT) $\subseteq$ *Class(LTE)*

Since any Generalized Logic Tree represents some Logic Term, there is nothing to show.

Outline of an Alternative Proof to show the Equivalences of all of the above Forms of Representation:

The proofs we showed above served two purposes: they established the equivalence of all forms of representation we considered, and also helped getting a better understanding of those representation forms. There is a potentially faster, but more technical way to show

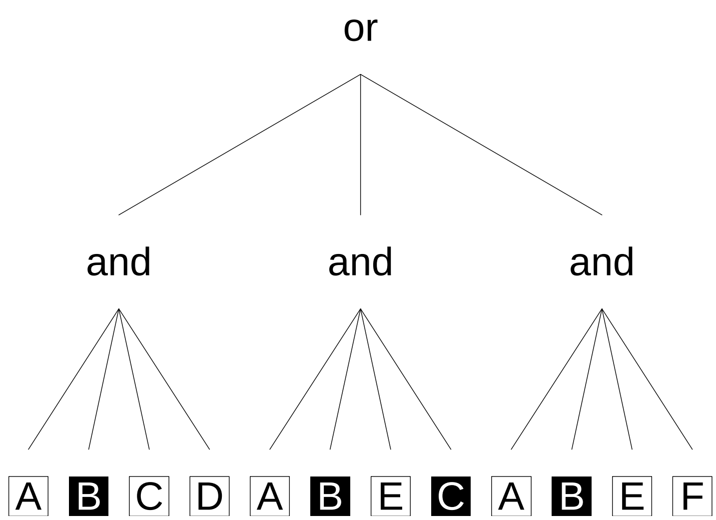


Figure 1.6: The Generalized Logic Tree for the Boolean expression in Disjunctive Normal Form from equation (1.29).

all of the above equivalences, and we outline the proof in this section. They idea is to show that there are only finitely many unique Boolean expressions, as we will explain below.

Assume we have k predictors, $X_1, \ldots, X_k$. Since the predictors are all binary, there are 2^k different possibilities how we can assign zeros and ones to the predictors (see Table 1.1). Each of the predictions $Y_1, \ldots, Y_{2^k}$ can be zero or one, hence we have 2^{2^k} possible so-called prediction scenarios.

If the predictors have been assigned values (which corresponds to a row in the table), then there is exactly one sequence $a = (a_1, \ldots, a_k) \in \{1, c\}^k$ such that

$$X^a := \bigwedge_{i=1}^{k} X_i^{a_i} \tag{1.30}$$

is true (if X_j is 0 (1), then a_j is 1 (c).) Name this sequence a^*. Now assume $Y_1, \ldots, Y_{2^k}$ have been assigned values, i. e. we pick one prediction scenario. How can we construct the right prediction, i. e. given X, how can we obtain Y? Let

$$M := \{j : Y_j = 1\} = (m_1, \dots , m_n). \tag{1.31}$$

Then

$$L = \bigvee_{j=1}^{n} X^{a^*_{m_j}} = \bigvee_{j=1}^{n} \left(\bigwedge_{i=1}^{k} X_i^{a^*_{i m_j}} \right) \tag{1.32}$$

correctly predicts Y. Hence, using this method, any prediction scenario can be obtained. It then only remains to be shown how we can re-write (1.32) as Logic Tree, Disjunctive Normal Form, etc. However, we decided to skip this part of the proof here.

Table 1.1: A table indicating all prediction scenarios.

X_1	$\cdots$	X_k	
0	$\cdots$	0	Y_1
$\vdots$		$\vdots$	$\vdots$
1	$\cdots$	1	Y_{2^k}

1.2.5 Relationship between Logic forms and Decision Trees (CART)

At first glance Logic Trees seem to be quite similar to classification trees as introduced by Breiman et al [6]. However, there are some key differences between those types of trees. We try to clarify these in this section.

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 for a case if the Boolean equation

$$L = P_1 \vee P_2 \vee \cdots \vee P_l \tag{1.33}$$

is true, where

$$P_i = C_1^i \wedge C_2^i \wedge \cdots \wedge C_{k^i}^i. \tag{1.34}$$

Hence every binary classification tree can be written as a Boolean equation in Disjunctive Normal Form. For example, the tree in Figure 1.7 predicts class 1 for a case if the Boolean equation

$$L = (C^c \wedge A \wedge B) \vee (C \wedge D^c \wedge A \wedge B) \vee (C \wedge D) \tag{1.35}$$

is true.

However, not every Boolean equation in Disjunctive Normal Form 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. For example, in the tree in Figure 1.7, the first question asked is always "is C true or not?". Therefore, it is not immediately clear if a classification tree can be constructed for the Boolean equation

$$L' = (A \wedge B) \vee (C \wedge D), \tag{1.36}$$

although L' is in DNF.

Using De Morgan's rules and standard Boolean operations, we convinced ourselves that a classification tree can be constructed from every Logic Term. However, these classification

Figure 1.7: An example of a CART tree.

trees often result in awkward looking constructions, much more complex than the simple Logic Trees constructed from the Logic Term. For example, for the Tree in Figure 1.7, we have

$$\begin{aligned} L &= (C^c \wedge A \wedge B) \vee (C \wedge D^c \wedge A \wedge B) \vee (C \wedge D) \\ &\equiv (A \wedge B \wedge [C^c \vee (C \wedge D^c)]) \vee (C \wedge D) \\ &\equiv (A \wedge B \wedge [(C^c \vee C) \wedge (C^c \vee D^c)]) \vee (C \wedge D) \\ &\equiv (A \wedge B \wedge [\,\mathtt{1} \wedge (C^c \vee D^c)]) \vee (C \wedge D) \end{aligned}$$

$$
\begin{aligned}
&\equiv (A \wedge B \wedge (C^c \vee D^c)) \vee (C \wedge D) \\
&\equiv [(A \wedge B) \vee (C \wedge D)] \wedge [(C^c \vee D^c) \vee (C \wedge D)] \\
&\equiv [(A \wedge B) \vee (C \wedge D)] \wedge [(C \wedge D)^c \vee (C \wedge D)] \\
&\equiv [(A \wedge B) \vee (C \wedge D)] \wedge \mathtt{1} \\
&\equiv (A \wedge B) \vee (C \wedge D) \\
&\equiv L' \qquad (1.37)
\end{aligned}
$$

This means that the classification tree in Figure 1.7 corresponds to the simple Logic Term $(A \wedge B) \vee (C \wedge D)$, which can also, very simply, be displayed as Logic Tree.

We established that every binary classification tree can be written in DNF, and we convinced ourselves that from every Logic Term a binary classification tree can be constructed. We previously showed that the classes of Disjunctive Norma Form and Logic Terms are equal, hence the class representing the binary classification trees is equal to all previously discussed classes. However, the simplicity of Logic Trees is one of their big attractions, which is one of the reasons we will not use classification trees in the development of our methodology for Logic Regression. This section was simply written to settle some of the questions about classification trees in comparison to Logic Trees that will almost invariably arise. Since we will not consider classification trees further, we also decided not to formally write down the proof that a binary classification tree can be constructed from any Logic Term.

Chapter 2

SEARCH ALGORITHMS

In the previous chapter we showed that, given a fixed number of predictors, there are only finitely many Boolean expressions that yield different predictions. If we have k predictors, we showed that there are 2^{2^k} different prediction scenarios. But given the values for the predictors, how many Logic Trees are there that yield different predictions? If we have l cases (and a sufficient number of predictors), there might be up to 2^l different Logic Trees. For example, if we have 1000 cases and 10 predictors, this could mean more than 10^{300} different Logic Trees! Besides the fact that we have to deal with huge numbers, it seems that there is no straightforward way how to enlist all those Logic Trees that yield different predictions. It seems impossible to us to carry out an exhaustive evaluation of all different Logic Trees. To find the good Logic Trees among the huge number of possibilities, we have to use some search algorithms that we discuss in this chapter in sections 2.2 (greedy search) and 2.3 (probabilistic search). Section 2.1 introduces what we call moves, a pre-requisite for the search algorithms.

2.1 Moving in the Search Space

In the search algorithms that we introduce in this chapter, we define the neighbors of a certain Logic Tree to be the Trees that can be reached from this Logic Tree by a single "move". 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 2.1(b) the leaf B from the initial tree has been replaced with the leaf D^c. To avoid tautologies, if the sibling of a leaf is a leaf as well, the leaf can not 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 $\wedge$s and $\vee$s:

  Any $\wedge$ can be replaced by a $\vee$, and vice versa (for example, the operator at knot 1 from the initial tree in Figure 2.1 has been changed in Figure 2.1(e)). These two moves complement each other as move and counter move.

- Branching (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, at knot 3 in the initial tree in Figure 2.1 we grew a branch (see Figure 2.1(f)). The counter move to branching 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 2.1(d).

- Splitting (deleting):

  Any leaf can be split by creating a sibling, and determining a parent for those two leaves. For example, in Figure 2.1(c) the leaf C from the initial tree in Figure 2.1 has been split, with leaf D^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 2.1(a).

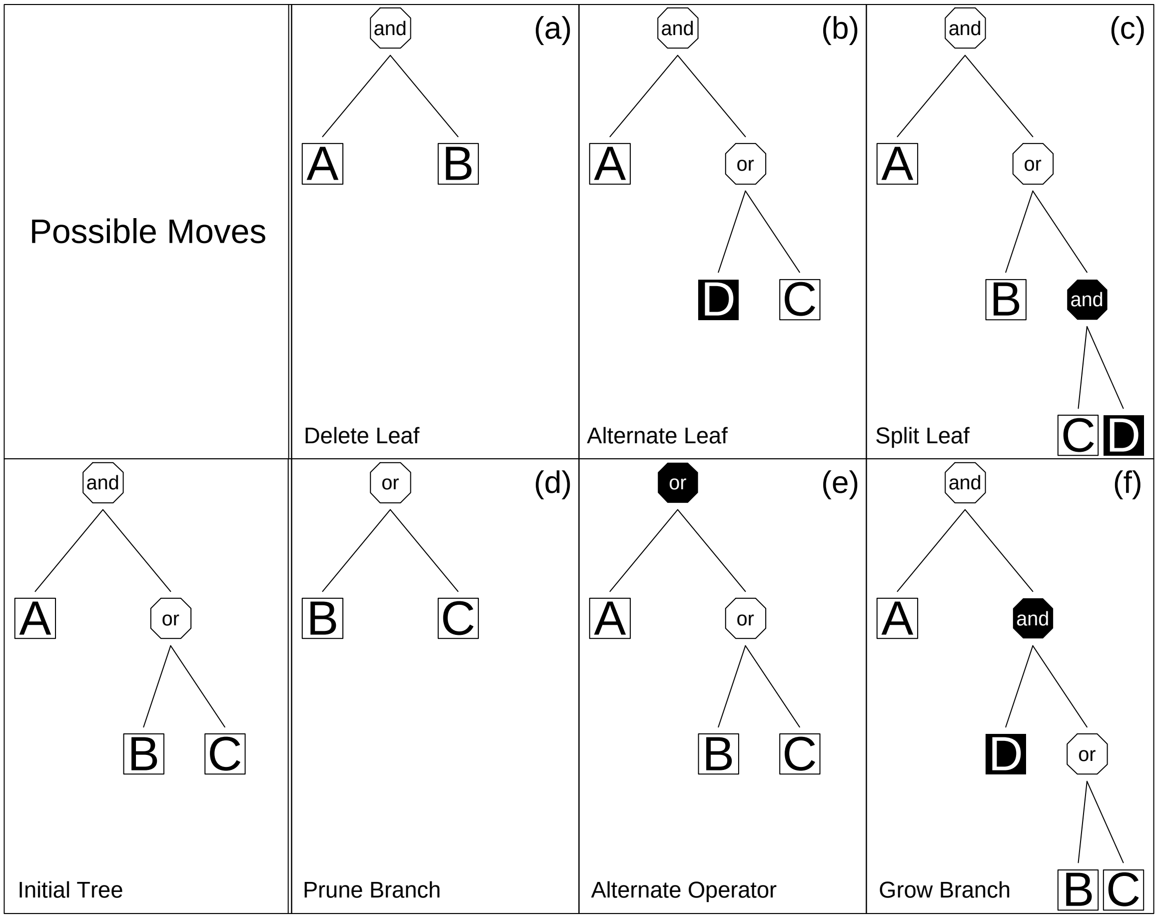


Figure 2.1: Permissible moves in the tree growing process.

Note: In principle, a Logic Tree can be reached from any other Logic Tree in a finite number of moves even if one omits pruning and branching. In this sense, pruning and branching are not necessary in the move set. However, their inclusion in the move set can enhance the performance of the algorithms that we will now introduce.

2.2 Greedy Search

Similar to the search algorithm in Classification and Regression Trees [6], a greedy algorithm can be used to search for “good” Logic Trees. In the context of Logic Regression, the first step is simply to find the variable that, used as a single predictor, minimizes the scoring function. 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,

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, etc.

Since the stop criterion is to be unable to find a move that improves the score, there is no guarantee to find the lowest scoring state possible. This can happen if the search gets "stuck", for example if a better tree can be reached in two moves, but not one. Another potential problem is that in the presence of noise in the data it can happen that even though the true tree has been reached in the search, there exist one or more additional moves that improve the score, and hence the final model involves some over fitting. It is also noteworthy that in contrast to the greedy search for Classification and Regression Trees, a greedy move for Logic Trees might actually result in a tree of lower or equal complexity (for example by deleting a leaf or changing an operator respectively).

Figure 2.2 shows parts of the outcome of a greedy search for Logic Trees on a simulated data set. The data was generated by simulating 20 binary predictors, with a total of 1000 cases each, and the value in each case of the predictors being a sample from a Bernoulli random variable with probability $\frac{1}{2}$. The underlying true Boolean equation was chosen to be

$$L = X_1 \wedge (X_2 \vee X_3) \wedge [X_4 \vee (X_5 \wedge (X_6 \vee X_7))] \tag{2.1}$$

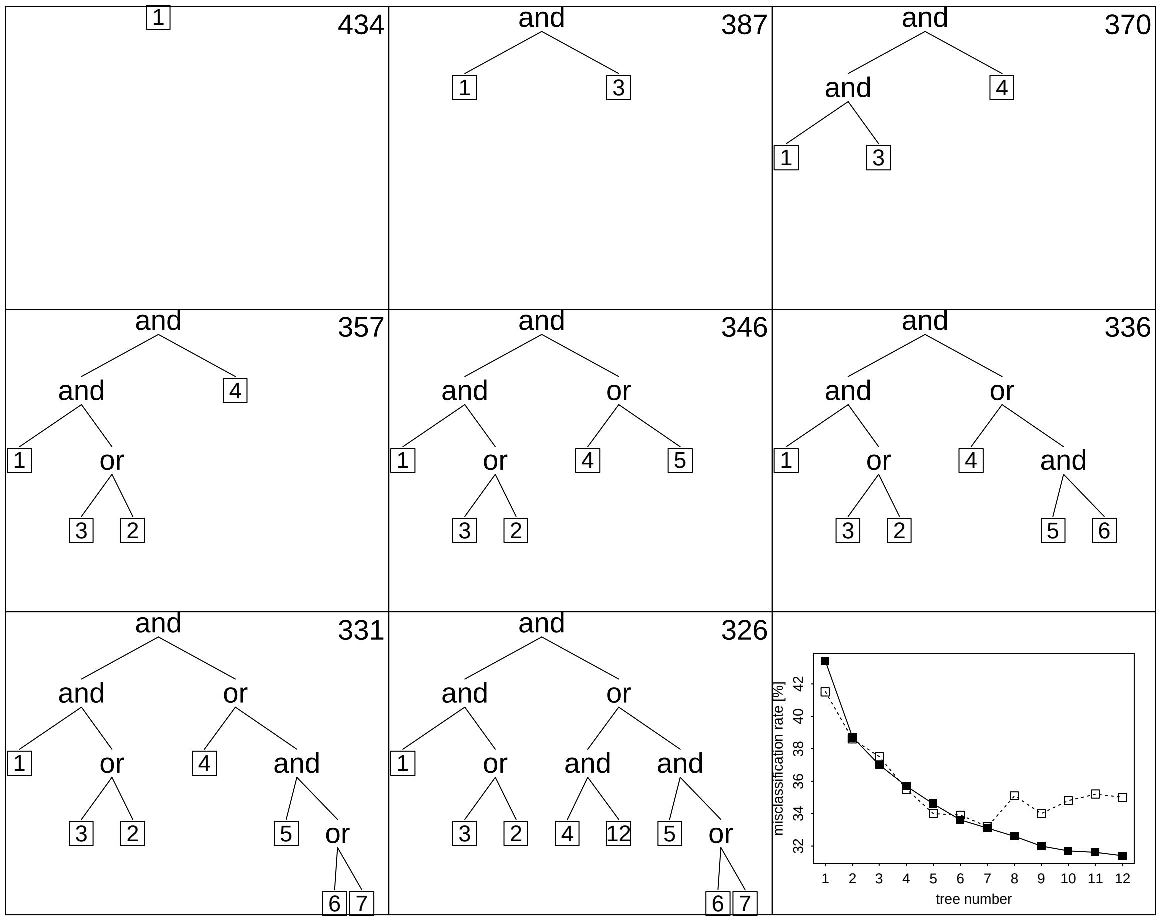


Figure 2.2: The sequence of trees visited in the greedy search.

If for a certain case L was true, the response was sampled from a Bernoulli random variable with probability $\frac{2}{3}$, otherwise it was sampled from a Bernoulli random variable with probability $\frac{1}{3}$. The score in the greedy search was simply chosen to be the number of misclassification (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 434 out of 1000 cases. The second step was splitting the first leaf into $X_1 \wedge X_3$, reducing the misclassification rate to 387 out of 1000 cases, etc. After seven steps, the correct tree was visited (lower left panel in Figure 2.2). The true misclassification rate in this example is 331 out of 1000. However, the algorithm didn't stop there. 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 5 fewer misclassification than the tree representing the true Boolean equation. After that, the greedy algorithm took four more steps (not displayed as trees in Figure 2.2) until it stopped, yielding a low misclassification rate of 314 out of 1000. The misclassification rate (in percent) for each tree is displayed in the lower right panel in Figure 2.2 as solid points. These misclassification rates were all calculated from the simulated data set described above. To get an estimate of the true misclassification rate, we generated a data set in the same manner as the training data set, except this time with 100,000 cases. The open points in the lower right panel in Figure 2.2 represent those estimates of the true misclassification rate. We see that through tree seven (which represents the true underlying Boolean equation) the true misclassification rate is decreasing with the tree number in the greedy search. After that, however, the following trees also predict some noise, and the true misclassification rate increases, as expected. This emphasizes the need for further statistical tools, since in real life problems the truth is unknown and the subject of the search.

2.3 Simulated Annealing

A greedy search algorithm is very fast compared to a probabilistic search algorithm, such as the one that we will introduce in this section. However, the greedy search can lead to wrong results in certain cases, as illustrated for example by Chipman et al [10]. The search can be trapped in a state that scores locally the best (i. e. all its neighbors have a worse score), but the state is not the best scoring globally. This can be avoided if the search algorithm is probabilistic, as we will explain below. There exists a profound analogy between this algorithm and some phenomena in the fields of statistical mechanics and condensed matter physics, which is the origin of the word "annealing" (see for example Kirkpatrick et al [24] and Cerný [9]).

In section 2.3.1 we introduce the terminology we use, and some basic definitions. The terminology used in this section follows the terminology used in the book by Otten and van Ginneken [35]. The definitions and theorems in this section were taken from this book and the book by van Laarhoven and Aarts [50]. In section 2.3.2 we list some properties of Markov chains and explain their connection to the simulated annealing algorithm that enables us to find good solutions in optimization problems. This is also illustrated in a simple example in section 2.3.4.

2.3.1 Terminology and Definitions

In this section we introduce most of the terminology we will use in context with simulated annealing, and the building blocks for the simulated annealing algorithm.

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. We require the state space to be finite. Th size of the state space is fixed, but can be arbitrarily large, therefore this assumption does not result in a loss of generality - for our purposes a computer must be able to distinguish all states anyways.

The following functions govern the search through the state space:

Definition 1

The **score function**

$$\epsilon : S \rightarrow I\!\!R_+ \tag{2.2}$$

assigns a positive real number (score) to each state.

The score is understood as a measure of the quality of the state. In the following we always assume that lower scores are associated with states that represent better quality configurations. Since the state space is finite, there exists at least one state with a minimal score. This score is denoted by ϵ_0.

Definition 2

The **selection probability** *is a function*

$$\beta : S \times S \rightarrow [0, 1] \tag{2.3}$$

such that

$$\begin{aligned} \forall_{(s,s') \notin M} \beta(s, s') &= 0, & (2.4) \\ \forall_{(s,s') \in M} \beta(s, s') &\neq 0, & (2.5) \\ \forall_{s \in S} \sum_{s' \in S} \beta(s, s') &= 1. & (2.6) \end{aligned}$$

The selection probability therefore is the probability that state s' is proposed as new state, given that the current state is s. Therefore the move set can be defined as

$$M := \{(s, s') \in S \times S : \beta(s, s') > 0\} \tag{2.7}$$

We call the move set M **symmetric** if

$$\forall_{s \in S} \forall_{s' \in S} \left[(s, s') \in M \Rightarrow (s', s) \in M\right] \tag{2.8}$$

Definition 3

The **acceptance function**

$$\alpha : I\!R_+^3 \rightarrow (0, 1] \tag{2.9}$$

assigns a positive probability to a pair of scores and a positive real number, called the **temperature**.

The acceptance function decides whether or not the proposed state will be accepted as the new state. Note that for any fixed temperature, this probability only depends on the scores of the current and proposed state, but not on those states themselves.

Definition 4

The **transition probability** *is a function*

$$\tau : \mathbb{R}^3_+ \to [0,1] \tag{2.10}$$

defined as

$$\tau(s,s',t) := \begin{cases} \alpha(\epsilon(s),\epsilon(s'),t) \times \beta(s,s') & s \neq s' \\ 1 - \sum_{s'' \in S} \alpha(\epsilon(s),\epsilon(s''),t) \times \beta(s,s'') & s = s'. \end{cases} \tag{2.11}$$

Therefore, the transition probability $\tau(s,s',t)$ can be understood as the probability that the next step is a move to state s', given that the current state is s and the temperature is t. The probability that the state after n moves is s', given the current state s and temperature the t, will be denoted $\tau_n(s,s',t)$.

A process that possesses the above property is called a **Markov process**. A sequence of events as a special case of such a Markov process is called a **Markov chain**. A Markov chain in which the transition probabilities between the pairs of states are constant throughout the process is called **homogeneous**. A Markov chain is called **irreducible** if any state in the chain is connected to any other state by only a finite number of moves, i. e. if

$$\bigcup_k M^k = S \times S. \tag{2.12}$$

A Markov chain is called **aperiodic** if for every state s the greatest common divisor of all integers $n \geq 1$ with $\tau_n(s,s,\cdot) > 0$ is equal to 1.

2.3.2 Properties of Markov Chains

Theorem 1 below is usually referred to as the Chain Limit Theorem. It states that an irreducible and aperiodic (homogeneous) Markov chain has a limiting distribution.

Theorem 1

For each irreducible and aperiodic chain there exists a density function

$$\pi : S \times I\!R_+ \rightarrow (0, 1], \tag{2.13}$$

in s for any given $t > 0$, with

$$\pi(s, t) = \lim_{n \to \infty} \tau_n(s', s, t), \tag{2.14}$$

(independent of s') and satisfying the following equations:

$$\begin{aligned} \sum_{s' \in S} \pi(s', t)\tau(s', s, t) &= \pi(s, t), && (2.15) \\ \sum_{s \in S} \pi(s, t) &= 1. && (2.16) \end{aligned}$$

Hence, if we constructed an irreducible and aperiodic (homogeneous) Markov chain for the annealing algorithm (i. e. run the chain at a fixed temperature), the distribution of states we sample from approaches a limit. However, the search through the state space should yield low scoring states. Some simple additional requirements will guarantee this.

Theorem 2

An irreducible and aperiodic chain with a symmetric move set has the property

$$\forall_{s \in S}[\epsilon(s) \neq \epsilon_0 \Rightarrow \lim_{t \downarrow 0} \pi(s, t) = 0] \tag{2.17}$$

if it has an acceptance α function satisfying

$$\begin{aligned} \epsilon \geq \epsilon' \quad &\Rightarrow \quad \alpha(\epsilon, \epsilon', t) = 1, && (2.18) \\ \epsilon > \epsilon' > \epsilon'' \quad &\Rightarrow \quad \alpha(\epsilon, \epsilon', t) \times \alpha(\epsilon', \epsilon'', t) = \alpha(\epsilon, \epsilon'', t), && (2.19) \\ \epsilon < \epsilon' \quad &\Rightarrow \quad \lim_{t \downarrow 0} \alpha(\epsilon, \epsilon', t) = 0. && (2.20) \end{aligned}$$

Hence, if the requirements (2.18), (2.19) and (2.20) are satisfied, the likelihood of a non-optimal scoring state in the limiting distributions goes to zero as the temperature goes to zero. Therefore, if the annealing is run as a sequence of homogeneous Markov chains with decreasing temperatures, the search is guided towards optimal scoring states. The above mentioned requirements only affect the acceptance function and do not pose any constraints on β or M. In general it is quite easy to construct a state space with a symmetric move set that guarantees irreducibility and aperiodicity for the chain in the search algorithm. The desirable properties of the chains as stated in Theorem 2 can be achieved by choosing the right acceptance function.

Otten and van Ginneken [35] make the point that "it seems reasonable that smaller score increases are accepted with higher probability than bigger ones, and that this probability varies smoothly with the score difference". Sufficient but not necessary for this would be the requirement that the acceptance only depends on the score difference.

Theorem 3

The only acceptance functions $\alpha(\epsilon, \epsilon', t)$

- *that are differentiable in* ϵ'*,*
- *whose values depend on* t *and the difference of* ϵ *and* ϵ'
- *that satisfy the conditions of Theorem 2*

have the form

$$\alpha(\epsilon, \epsilon', t) = \min\{1, e^{(\epsilon' - \epsilon)c(t)}\}, \tag{2.21}$$

where $c(t)$ *is a negative, monotonic and continuous function satisfying*

$$\lim_{t \downarrow 0} c(t) = -\infty. \tag{2.22}$$

In our works we always used $c(t) = -1/t$, yielding the acceptance function

$$\alpha(\epsilon, \epsilon', t) = \min\{1, e^{-(\epsilon'-\epsilon)/t}\}. \tag{2.23}$$

This acceptance function has been used by far the most in the literature. This is presumably the case because condensed matter physics is the origin of simulated annealing and the above acceptance function has a striking similarity to the Boltzmann distribution, which characterizes a system of particles in thermal equilibrium. However, it also has been established that this acceptance function has many desirable properties, as described above.

2.3.3 Practical Aspects of Simulated Annealing

When we implemented the simulated annealing algorithm for the Logic Regression methodology, some practical aspects had to be considered how to run the search. In this section we list what we believe are the most important issues.

- In theory, trees of any size can be grown, but considering that we want to be able to interpret these models, it makes sense to limit the tree sizes to a reasonable number of leaves. We always took this into consideration, and usually limited the number of leaves to a maximum of 16 per tree.

- 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 those points in time of the run is the “crunch time”, where we want to spend most of the run time in. 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 pre-determined 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 pre-specified number of iterations for a chain at 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 when no moves were accepted within 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 independent of the lowest temperature considered.

• Every Logic Tree has a finite number of neighbors. Especially towards the end of the run at very low temperatures, very few moves get accepted. Since 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, i. e. deriving the values of their underlying Boolean equation from the leaves for each case. Since the acceptance of a move, given the temperature and the score of the current model, only depends on the score of the proposed model, we implemented a subroutine that keeps track of all states visited, and their scores. Therefore, for the decision whether or not to accept a certain move, the trees of the proposed model have to be evaluated only once, which speeds up the search dramatically especially at lower temperatures.

- To implement a simulated annealing run, we have make some decisions, such as specifying 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 the cooling scheme etc depend on the data we are analyzing. The theory of simulated annealing in the previous section tells us under which circumstances we can guarantee to find an optimal scoring state. Since we cannot run chains of length infinity, the optimal state is not guaranteed 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 data set, 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 to subsequent powers of 10 depend on the data as well; usually, the number is between 25 and 100. The lengths of the individual chains for the data we looked at so far have been between 10,000 and 100,000. The number of iterations might increase substantially though for larger data sets. In Section 3.2.1 we will give a detailed example of such a simulated annealing run.

2.3.4 An Example

For simplicity, we decided not to show how simulated annealing works for growing Logic Trees before discussing the logic models that we consider (see Chapter 3). A much more straightforward problem is the traveling salesman problem, a "classic" in theoretical computer science. Similar to searching for Logic Trees, this is a very discrete problem and an enumeration of states and therefore a sequential search is not really possible.

Our traveling salesman has to visit n cities. He wants to find the path that minimizes his traveling expenses, assumed to be proportional to the length of his journey. In our example, the cities are on a regular 10×10 unit grid. A possible path for the sales man's journey is shown in Figure 2.3.

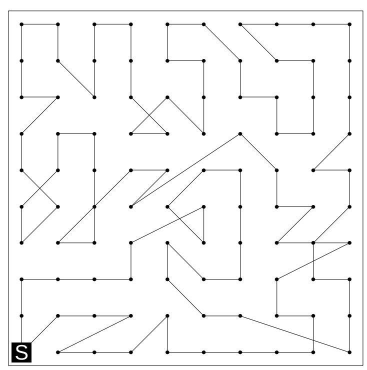


Figure 2.3: A possible path for the traveling salesman, visiting 100 cities.

Here, the state space is a sequence of the numbers 1 through 100, each number representing one city. A sequence of these 100 numbers is a path, and represents one configuration of the traveling salesman problem.

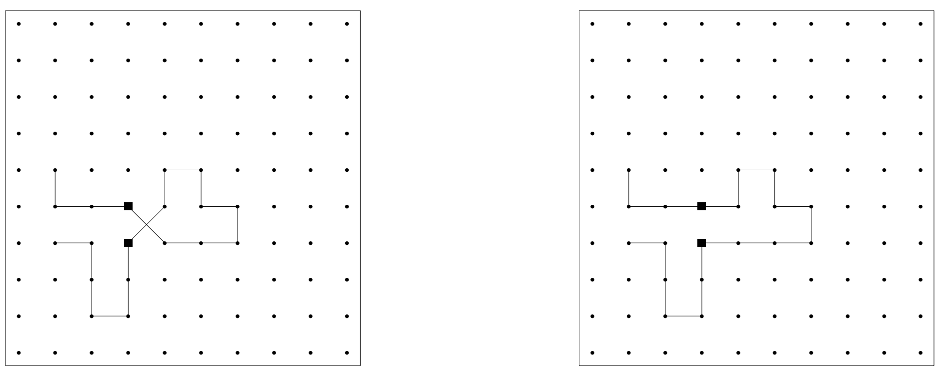
Clearly, there are plenty of ways how to define the neighborhood system, and hence the move set, for this problem. Černý [9] for example suggested the permutation of the cities in a sub path. For any pair of cities, reverse the sub path between them. For example, if

$$\cdots\ 52\ 42\ 43\ 44\ 35\ 36\ 37\ 47\ 46\ 56\ 55\ 45\ 34\ 24\ 14\ 13\ 23\ 33\ 32\ \cdots$$

is a path, then reversing the sub path between cities 44 and 34 is a permissible move:

$$\cdots\ 52\ 42\ 43\ 44\ \underbrace{45\ 55\ 56\ 46\ 47\ 37\ 36\ 35}_{\text{reversed subpath}}\ 34\ 24\ 14\ 13\ 23\ 33\ 32\ \cdots$$

We also refer to these type of moves as swaps. Figure 2.4 shows such a move. It is clear now that two states (i. e. paths through the cities) are neighbors if one can be generated by the other using one swap.



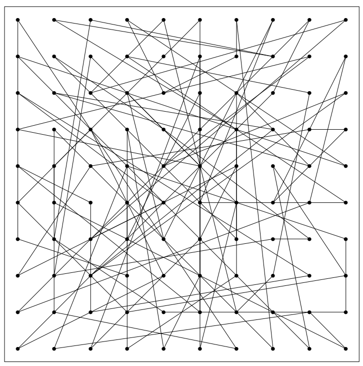
(a) The path before the swap.

(b) The path after the swap.

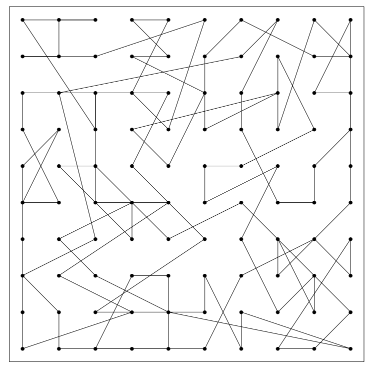
Figure 2.4: A move in the traveling salesman problem (only a part of the path is shown).

The score of each state in this problem is simply the length of the path. For example, the score for the path shown in Figure 2.4 (a) is $2\times\sqrt{2}-2\times 2 \approx 0.83$ units higher than the score for the path shown in Figure 2.4 (b), and hence less favorable. The selection probability is simply determined by randomly selecting two cities on the path, reversing the sub path between them, and propose the resulting path as new state. Clearly, this makes the move set symmetric. Also, every sequence of the numbers 1 through 100 can be generated from any other sequence by successively exchanging two neighboring numbers in the sequence. This is of course not the fastest way of doing so, but easily demonstrates that our Markov chain is irreducible. It is also not too hard to see that the Markov chain is aperiodic. If a state has a lower scoring neighboring state, there is a positive probability that this state will be the new state in the chain. This state now has the original state as neighbor, and hence there is a positive probability that the chain will not leave this state in the next step in the chain (since the original state has a higher score). But there is also a positive probability of returning to the original state, of course depending on the temperature. However, since the chain can remain in the new state, the return to the original state could happen after any number of steps. Hence the chain is aperiodic (if a state does not have a lower scoring neighboring state, there is nothing to show, since there is a positive probability that the chain will remain in this state). Therefore, all the necessary requirements in the above theorems are fulfilled. Figure 2.5 shows the result of an annealing run. Each Markov chain in the annealing algorithm is referred to as a step in this algorithm.

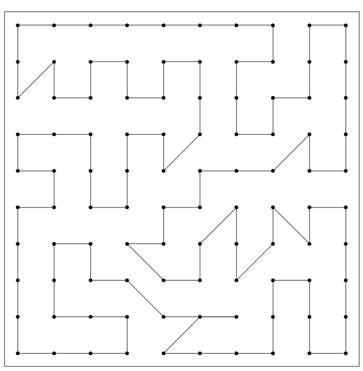
How simulated annealing works in the context of Logic Regression will be clarified after we introduce Logic models in Chapter 3.



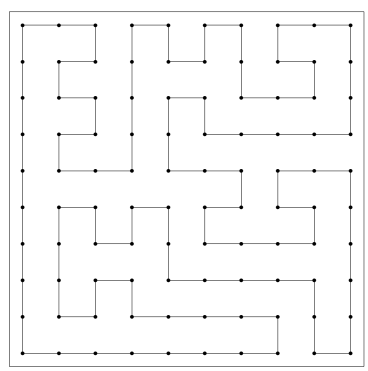
(a) The initial path.



(b) The tour after 60 steps.



(c) The tour after 90 steps.



(d) The tour after 125 steps.

Figure 2.5: Configurations obtained during a simulated annealing run of the 100 cities traveling salesman problem.

Chapter 3

LOGIC MODELS

After describing the search algorithms that enable us to find good Logic Trees, we now introduce the Logic models that we think are most useful in the Logic regression framework. In principle, we distinguish Logic models with a single tree and Logic models with multiple trees. The models we consider involving one Logic Tree are for classification and regression problems. In the former, we try to find a Boolean expression L that classifies each case into one of two possible classes. In other words, we are searching for a classification rule

$$C = I_{\{L \text{ is true}\}}. \tag{3.1}$$

In the latter problem, we try to model a characteristic in each of two subpopulations, such as the population mean of a certain variable in the subpopulations. Here, we search for a logic rule and parameters that, for example, optimize the model

$$\mu = a + b \times I_{\{L \text{ is true}\}}. \tag{3.2}$$

In both cases, a meaningful measure of goodness (i. e. scoring function) has to be defined. Which measure to use depends of course on the type of problem. In Section 3.1.1 we discuss these models and scoring functions in detail.

After we discussed models with one tree, we will generalize (3.2) to

$$\mu = a + \sum_{i}^{k} b_i \times I_{\{L_i \text{ is true}\}}. \tag{3.3}$$

These models with multiple trees are discussed in Section 3.1.2. In Section 3.2 we describe how all of the above mentioned models can be fit.

3.1 Introduction of Logic Models

3.1.1 Models with One Tree

3.1.1.1 Classification Problems

A common statistical problem is to classify members of a population in one of two possible categories, solely based upon a set of "predictive" variables, measured for each member of the population. For example, it is believed that certain types of cancer are partly caused be genetic abnormalities. Is it possible to predict which class (cancer / no cancer) a person belongs to, given a list of genes expressed? The type of model we could fit in this example might be something like "if gene A and gene B are expressed, or gene C but not gene D, then we predict that the person whose genes we analyzed belongs to the group with cancer, otherwise he or she belongs to the no cancer group. below, we discuss which scoring functions can be used for this type of problem.

Misclassification Rate: In the classification problems we consider, we search for rules that enable us to assign each case into one of two possible classes (say class 0 and class 1). A Boolean expression L predicts a class C for each case via $C_{\text{predicted}} = I_{\{L \text{ is true}\}}$. The scoring function for this case could simply be the total number of misclassifications resulting from this prediction n_{mc}, or the misclassification rate, the total number of misclassifications divided by the total number of cases ($\epsilon = n_{\text{mc}}/n$). In many applications, the types of misclassification are not equally severe. For example in medical cases, predicting that a person is healthy although he or she is sick and needs treatment immediately often is a much more serious mistake than predicting the person is sick although he or she is healthy. In those cases it is common to use (for example) a weighted version of the number of misclassifications, such as $\epsilon = w_0 n_{\text{mc}_0} + w_1 n_{\text{mc}_1}$ as scoring function, where $n_{\text{mc}_0}(n_{\text{mc}_1})$ is the number of cases falsely predicted as class 0 (1).

The above described scoring functions are pretty much the simplest scoring functions possible, and are the most commonly used in classification problems. Weights are also commonly used in situations where one class is much more common than the other class. An alternative is to use the

Inter-rater Disagreement: An index called inter-rater agreement κ (see for example Fleiss [14]) was developed to assess how two people agree in classifying (rating) the same objects into different classes. In our case we can understand this as "how well does the prediction agree with the response?". The difference to the simple misclassification rate is that we now take into account what one expects in terms of agreement between prediction and response. Let the total number of cases considered be n, assume there are r_0 zeros and r_1 ones in the response ($r_0 + r_1 = n$), and assume that we predict a_0 zeros and a_1 ones ($a_0 + a_1 = n$). Let $m := 2k + n - a_0 - r_0$ be the number of agreements between prediction and response, as indicated in Figure 3.1.

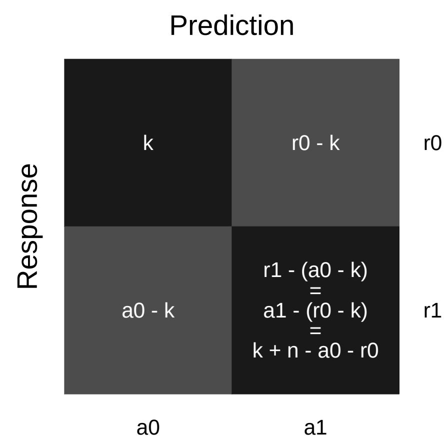


Figure 3.1: The agreement between prediction and response.

The inter-rater agreement κ as in Fleiss [14] is defined as the ratio of the number of cases by which m exceeds its expectation, and the number by which m could theoretically exceed this expectation. Using our terminology, we can show that

$$\kappa \;=\; 1 - \frac{n(n-m)}{n(a_0 + r0) - 2a_0 r_0}. \tag{3.4}$$

Since in our scoring function lower scores should be associated with better states, we define the inter-rater disagreement as

$$\gamma := 1 - \kappa = \frac{n(n-m)}{n(a_0 + r0) - 2a_0 r_0}. \tag{3.5}$$

3.1.1.2 Regression Problems

In the regression problems that only involve a single tree we search for rules that characterize parameters associated with measurements in two subpopulations. If the response is binomial, the parameter of interest can be the odds of belonging to class 0 versus class 1. If the response is continuous, the parameter can be the average response in each of the two classes. For example, imagine there is a car dealer who sells new luxury cars all over the country wants to target potential customers with advertisement. To find out which people are willing to spend a lot of money for a new car, he carries out a survey. He questions people who recently bought a new car how much they paid for it and gathers additional information about these people, such as their gender, age, profession, residency, education, etc. Quite possibly he might find out that for example professionals with a graduate degree (lawyers, doctors, ...) spend on average more money on new cars than the average population. The same could be true for retired folks who made enough money to enjoy the rest of their life in Florida or California. A good model could be the following: A person who works professionally and has a graduate degree, or is retired and lives in Florida or California, belongs to a population which spends about $40,000 on average on a new car, otherwise this person belongs to a population which spends only about $25,000 on average on a new car.

Below, we outline what we consider the two most important types of regression models:

linear regression and logistic regression, with residual sum of squares and binomial log-likelihood (respectively) as scoring functions.

Residual Sum of Squares: In case of a continuous response, we want to model the means of the responses in two different subpopulations, for example the average amount of money spent on a new car in two subgroups, defined by variables recorded for the people in the population of buyers of new cars. We assume the true underlying model for the measurements taken to be

$$Y = \beta_0 + \beta_1 I_{\{L \text{ is true}\}} + \epsilon, \quad \text{with} \quad \epsilon \sim N(0, \sigma^2) \tag{3.6}$$

where L is a Boolean expression that determines the means of the response variable Y in the two subpopulations: the mean $\mathbb{E}[Y]$ of the response variable in class 0 (when $L = 0$) is β_0, which is β_1 smaller than the mean of the response variable in class 1 (when $L = 1$). For a given L, the model is fit using the method of least squares, which means finding the estimates $\hat{\beta}_0, \hat{\beta}_1$ that minimize the expression $\sum_i (\beta_0 + \beta_1 I_{\{L_i=1\}} - Y_i)^2$. As scoring function we select the residual sum of squares (RSS):

$$\text{RSS} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2, \tag{3.7}$$

where $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 I_{\{L \text{ is true}\}}$ are the fitted values of the model under consideration. In this case, the fitted values are simply $\hat{\beta}_0 = \bar{Y}_{\{L \text{ is false}\}}$ and $\hat{\beta}_1 = \bar{Y}_{\{L \text{ is true}\}} - \hat{\beta}_0$.

Logistic Log-Likelihood: The case with binary response can also be considered as a regression problem. Instead of predicting the outcome for each case as in the classification case, we can model the probability (actually, the log odds) of belonging to class 1 instead of class 0. We assume the true underlying model to be

$$\log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 I_{\{L \text{ is true}\}}, \tag{3.8}$$

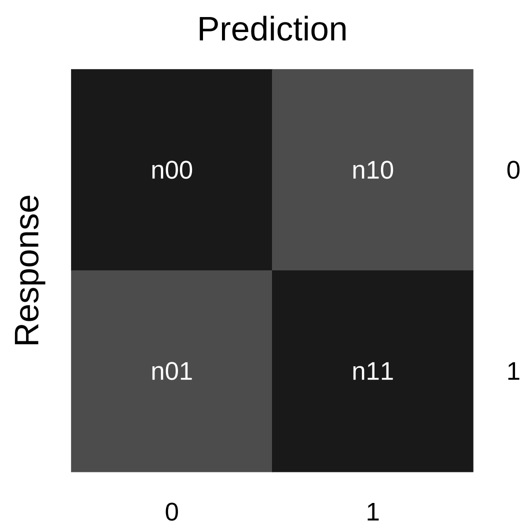


Figure 3.2: Counts of cases in each bin of the table indicating the agreement between prediction and response.

where π is the probability for a certain case to be in class 1 and L is a Boolean expression that determines these probabilities: the log odds of being in class 1 when $L = 0$ is β_0, which is β_1 smaller than the log odds of being in class 1 when $L = 1$. As scoring function we select the deviance (DEV), defined to be twice the difference between the maximum achievable log likelihood and that attained under the fitted model (see for example McCullagh & Nelder [30], p. 118ff).

In the case with only one Logic term (see Figure 3.2), the fitted probabilities for being in class 1 given $L = 0$ and $L = 1$ respectively are

$$\hat{\pi}_0 = \frac{n_{01}}{n_{0\cdot}}, \tag{3.9}$$

$$\hat{\pi}_1 = \frac{n_{11}}{n_{1\cdot}}, \tag{3.10}$$

where n_{ij} is the number of cases for which we predict i and the response is j ($i, j \in \{0, 1\}$).

This deviance as given in [30] then simplifies to

$$\begin{aligned}\text{DEV} = 2 \times \Bigg[& n_{00} \log\left(\frac{n_{0\cdot}}{n_{00}}\right) + n_{01} \log\left(\frac{n_{0\cdot}}{n_{01}}\right) + \\ & n_{10} \log\left(\frac{n_{1\cdot}}{n_{10}}\right) + n_{11} \log\left(\frac{n_{0\cdot}}{n_{11}}\right) \Bigg]\end{aligned} \tag{3.11}$$

Although we only discussed linear and logistic regression, any regression model can be used as long as a meaningful measure of goodness can be defined; in particular, this includes the generalized linear regression models, and the Cox model.

3.1.2 Models with Multiple Trees

Models involving a single Logic Tree always characterize two subpopulations. But certainly there are cases in which it is desirable to consider more than two subpopulations. For example, when patients are admitted to hospital for a certain reason, a variety of variables are measured upon arrival. Breiman et al [6] for example analyzed data from heart attack patients admitted to the Medical Center at the University of California at San Diego. The goal of a medical study was to identify high risk patients (patients who will not survive at least 30 days) based on the data recorded within 24 hours upon arrival at the hospital. It would be desirable to have a model that permits statements such as: the basic odds of surviving at least 30 days is α_0. If the minimum systolic blood pressure over the initial 24 hour period is higher than c_{msbp} and the patient is older than c_{age} years and sinus tachycardia is present, than the odds are α_1 times higher. If the patient is female and there is no history of angina, the odds are α_2 times lower, etc. Note that these statements are not exclusive and that more than one of the conditions can apply. We will get back to those data analyzed by Breiman et al [6] in Section 6.2.

The only models with multiple trees that we considered and implemented are in the context of linear and logistic regression. Again, in principle any model can be considered, as long as a meaningful scoring function can be determined. Classification problems in which there are more than two possible classes for the outcome could also be targeted using a polychotomous regression approach to Logic Regression (see for example Kooperberg et al [27]).

3.1.2.1 Linear Regression

In an extension of the single tree case, we assume the true underlying model to be

$$Y = \beta_0 + \beta_1 I_{\{L_1 \text{ is true}\}} + \cdots + \beta_p I_{\{L_p \text{ is true}\}} + \epsilon, \quad \text{with} \quad \epsilon \sim N(0, \sigma^2) \tag{3.12}$$

where $L_1, \ldots, L_p$ are a Boolean expressions that determine the means of the response variable Y in multiple subpopulations. For a given set of Boolean expressions $L_1, \ldots, L_p$, the model is fit using the method of least squares, and as scoring function we use the residual sum of squares (RSS):

$$\text{RSS} = \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2, \tag{3.13}$$

where

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 I_{\{L_1 \text{ is true}\}} + \cdots + \hat{\beta}_p I_{\{L_p \text{ is true}\}} \tag{3.14}$$

are the fitted values of the model under consideration. Note that this, in principle, defines 2^p subgroups of cases. However, some of these groups might be empty, for example if 'L_1 is true' requires the gender "male" and 'L_2 is true' requires the gender "female".

3.1.2.2 Logistic Regression

In another extension of the single tree case, we assume that the true underlying model is

$$\log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 \times I_{\{L_1 \text{ is true}\}} + \cdots + \beta_p \times I_{\{L_p \text{ is true}\}} \tag{3.15}$$

where π is the probability for a certain case to be in class 1 and $L_1, \ldots, L_p$ are the Boolean expressions that determine these probabilities. As scoring function we select the deviance (DEV), defined to be twice the difference between the maximum achievable log likelihood and that attained under the fitted model (see for example McCullagh & Nelder [30], p. 118ff).

3.2 Model Fitting

3.2.1 Models with One Tree

As introduced in Chapter 2, we basically have two search algorithms: the greedy search and the simulated annealing. Both of them involve a step-wise search through the space of possible models. In every step, we alter the Boolean expression currently under investigation, according to the set of permissible moves (also introduced in Chapter 2). Given the new tree, we then calculate the new score, and decide whether or not to accept the new model based on old score and new score, and we adjust the temperature if we carry out a simulated annealing algorithm. In some of the models this might involve re-fitting parameters. For example if we consider a linear regression type of model, a change in the Boolean expression L also alters the parameters β_0 and β_1. These are then used to calculate the goodness of fit, e. g. the residual sum of squares. Below we show as an example the outcome of a simulated annealing run for a classification problem, which involved a Logic model with a single tree. We simulated a dataset with 20 binary predictors, having a total of 1000 cases each. The value in each case of the predictors was a sample from a Bernoulli random variable with $p = 0.5$. The underlying Boolean equation was chosen to be

$$L = [X_1 \vee X_2] \wedge [(X_3^c \vee X_4) \wedge (X_5^c \vee X_6)]. \tag{3.16}$$

If for a certain case L was true, the response was sampled from a Bernoulli random variable with $p = \frac{2}{3}$ (class 1), otherwise the response was sampled from a Bernoulli random variable with $p = \frac{1}{3}$ (class 0). The score in the simulated annealing run was chosen to be the number of misclassifications (i. e. how often a tree under consideration predicted the wrong class).

Figure 3.3 displays some information about the scores throughout the simulated annealing run. We started the first Markov chain at a temperature of 10^4, and ended the run with a Markov chain at temperature of 10^{-2}. The temperature was decreased after each run of the Markov chain in increments of $\frac{1}{25}$ on a $\log_{10}$ scale. Each Markov chain had a fixed length, 50000 iterations in our case. For each Markov chain at a fixed temperature, we recorded the median and the upper and lower 2.5 percentile of the scores of the Logic Trees accepted in the chain. For simplicity, only trees with a maximum of 16 leaves (depth 4) were considered. The median scores for each temperature are connected by a solid line in Figure 3.3, and the upper and lower 2.5 percentile define the shaded area around the median. We see how the median scores drop with the temperature, together with the range

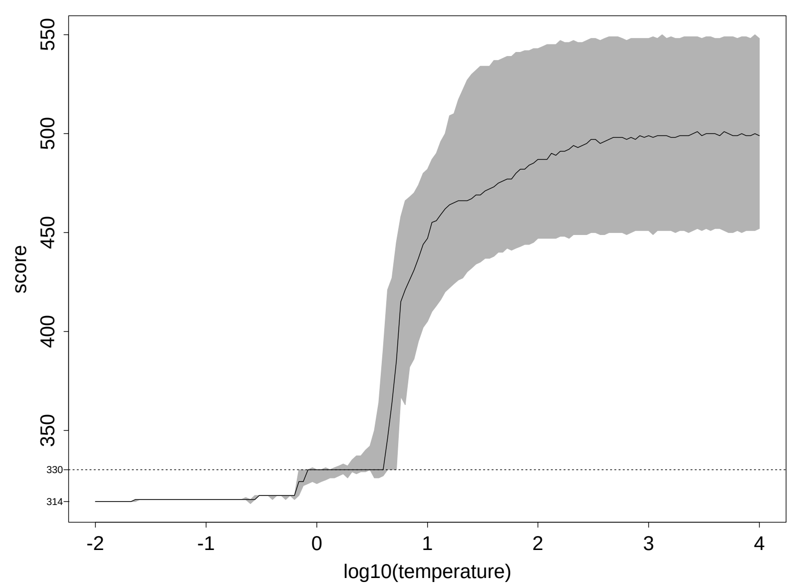
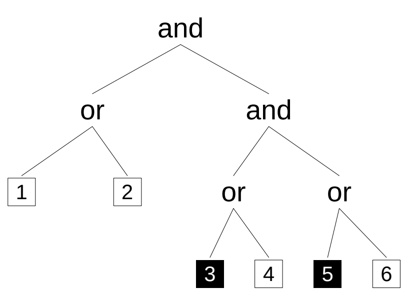


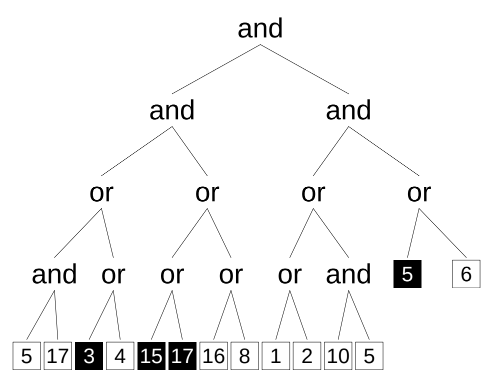
Figure 3.3: Temperature versus score in a simulated annealing run, trying to solve a classification problem.

and the variability of the scores of the accepted tree. The true Logic Tree has a score of 330 (330 out of 1000 cases are misclassified under L). There is noise in the data, and we see that eventually we start predicting some of that noise by visiting trees that have a score less than 330. In the end, we found a Logic Tree with a score of 314.

Figure 3.4 compares the correct Logic Tree (Figure 3.4(a)) with the Logic Tree that had the lowest score in the simulated annealing run (Figure 3.4(b)). The Logic Tree with score 314 captures the information of the underlying Boolean equation (3.16) (which means that the predictors $X_1, X_2, X_3^c, X_4, X_5^c, X_6$ are all in the model), but has, as already stated, some additional predictors not part of the true model. We discuss in Chapter 5 how to deal with noisy data, or data for which the Logic model is only an approximation.



(a) The true Logic Tree.



(b) The Logic Tree with the lowest score.

Figure 3.4: Trees associated with the simulated annealing run.

3.2.2 Models with Multiple Trees

Considering models with one tree, we only had the choice of picking a greedy or a stochastic search algorithm. In a model with multiple trees, we have more options.

Fitting all trees simultaneously

Again, the search can be carried out either using a greedy algorithm or simulated annealing. In the former case, we start out with an empty model (equivalent to no trees), and find the best single move according to our scoring function. This is the start of the first tree in the model. Then we search for the next move, finding the best model we can reach from the given model. The move can either be altering the first tree, or starting a second tree. The next move could be altering any of the trees in the model, or starting a new tree. The greedy search stops when we can not improve the score of the model anymore.

For computational reasons, fitting all trees simultaneously requires that we pre-select the number p of trees when we use simulated annealing. The number of trees can be chosen arbitrarily large (although computer memory permitting), so this is not a major restriction. Unless we have an idea of how many trees we maximally want to fit, it may not be clear a priori what this number should be. We generally pick p conservatively, which means we pick p larger than necessary, and trim the model down if needed. In Chapter 5 this will be discussed in detail. For now, we assume that p is known. In theory, we could run the program with an undetermined number of trees.

When we use simulated annealing, we have two possibilities. We can either change one tree at a time, or change all trees simultaneously. The latter requires that we slightly change the move set, allowing "no move" as a possibility at an individual tree. We select a move for each tree from the move set described in section 2.1 or we do not alter this tree.

Fitting one tree at a time

In certain cases, we could grow one tree at a time. The main advantage compared to fitting all trees simultaneously is that it is computationally less expensive. However, it is a somewhat "semi-greedy" approach, which bears a risk of a non-optimal result of the search. We can grow the first tree as described in Section 3.2.1. Then we add a second tree, keeping the first tree fixed, etc. We stop when we reach a certain criterion, for example in linear regression when the significance (p-value) of the slope parameter exceeds a certain value. This method yields a sequence of Boolean expressions that we can use to find the final parameters in the linear regression model, using all trees in one fit.

Below we show the outcome of a simulated annealing run, trying to solve a regression problem which involved a Logic model with multiple trees. Exactly as in the example in the previous section, we simulated a dataset with 20 binary predictors, having a total of 1000 cases each. The value in each case of the predictors was a sample from a Bernoulli random variable with $p = 0.5$. The response was generated according to the model

$$Y = 3 + 2 \times I_{\{L_1 \text{ is true}\}} - 1 \times I_{\{L_2 \text{ is true}\}} + \epsilon, \tag{3.17}$$

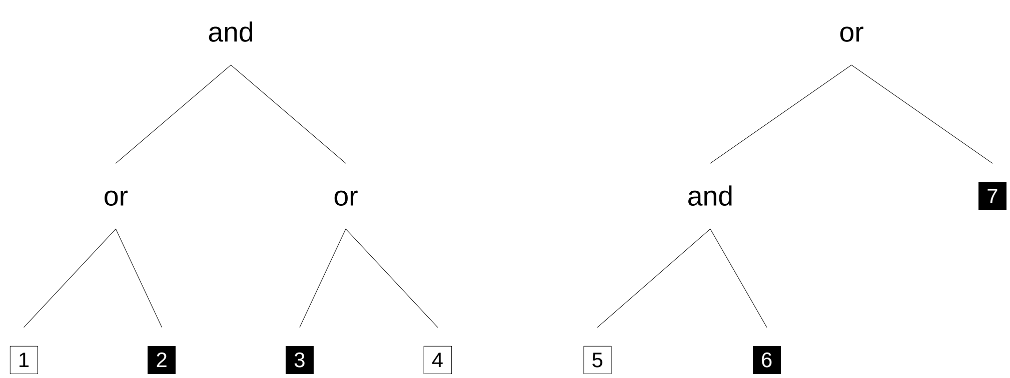


Figure 3.5: The Logic Trees of the correct regression model.

where $\epsilon \sim N(0,1)$, and

$$L_1 = (X_1 \vee X_2^c) \wedge (X_3^c \vee X_4) \quad (3.18)$$

$$L_2 = (X_5 \vee X_6^c) \wedge X_7^c \quad (3.19)$$

(see Figure 3.5).The score in the simulated annealing run was the residual sum of squares. For simplicity, we limited the number of Logic Trees in the model to 3, and only trees with a maximum of 16 leaves (depth 4) were considered.

We started the first Markov chain at a temperature of 10^3, and ended the run with a Markov chain at temperature of 10^{-2}. The temperature was decreased after each run of the Markov chain in increments of $\frac{1}{25}$ on a $\log_{10}$ scale. Each Markov chain had a fixed length, 50000 iterations in our case. The final model had a score of 896.9, compared to 1005.6 for the true model. Figure 3.6 displays the change in the parameter estimates throughout the simulated annealing run.

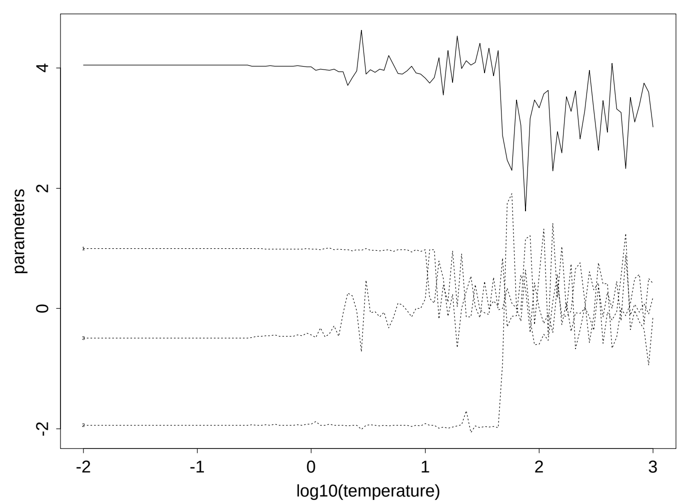


Figure 3.6: The temperature versus the estimates of regression parameters at the end of the Markov chain at the respective temperature. The solid line represents the estimates for the intercept $\hat{\beta}_0$, the dashed lines represents the estimates for the parameters $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$.

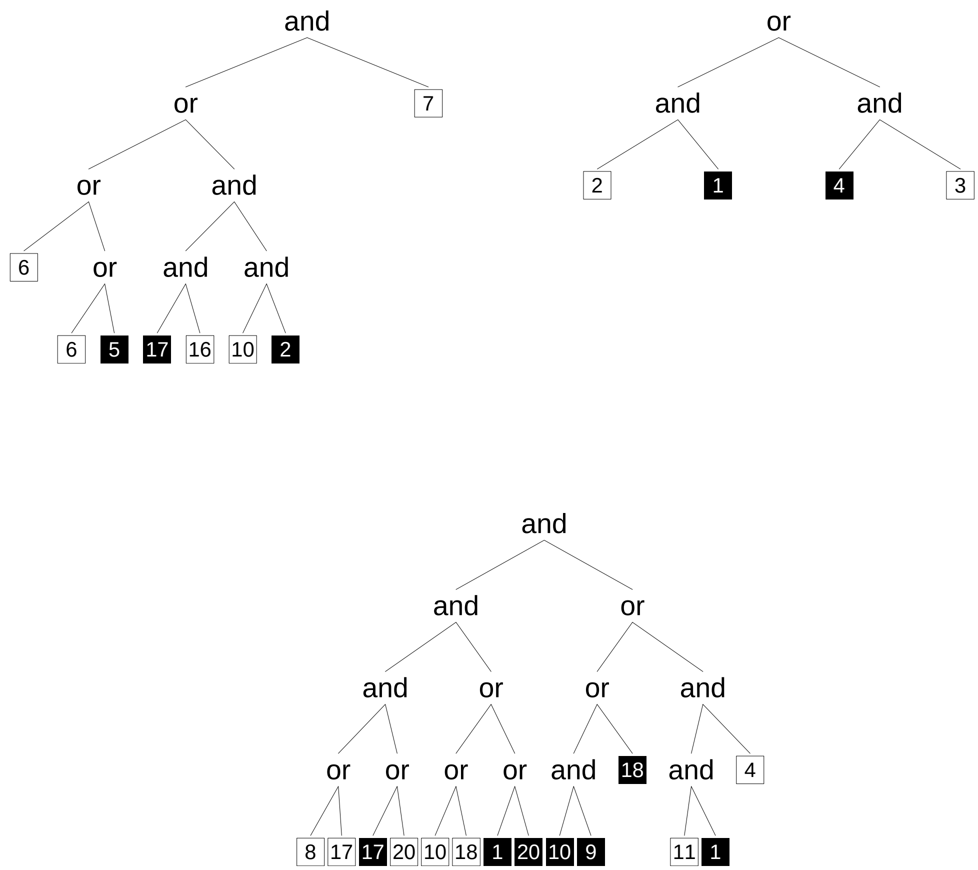


Figure 3.7: The Logic Trees of the regression model with the lowest score.

Figure 3.7 shows the Logic Trees of the resulting model. This model can be written as

$$\hat{Y} = 4.05 + 1.00 \times I_{\{T_1 \text{ predicts } 1\}} - 1.94 \times I_{\{T_2 \text{ predicts } 1\}} - 0.49 \times I_{\{T_3 \text{ predicts } 1\}} \qquad (3.20)$$

The first tree (T_1) resembles the Logic Tree for L_2. The subtree $X_6 \vee (X_6 \vee X_5^c)$ contains redundancy, it is equivalent to $X_6 \vee X_5^c$. The subtree representing the Boolean expression $(X_{17}^c \wedge X_{16}) \wedge (X_{10} \wedge X_2^c)$ rarely predicts one, since all four leaves involved in the subtree have to have prediction one, so the Boolean equation underlying T_1 is essentially the complement of L_2 (using DeMorgan's rules). Hence $\hat{\beta}_1$ is very close to $+1$ instead of -1. T_2

is exactly the complement of L_1, hence $\hat{\beta}_2$ is very close to -2. Knowing the underlying model, we realize that T_3 is a result of over fitting. In the following chapters we deal with model simplification (removing redundancies, statistical inference, model selection, etc), necessary to distinguish noise from signal.

Chapter 4

TRICKS AND TECHNICALITIES

In general, if we fit models as described in Chapter 3, there is the possibility of over-fitting. Therefore it is crucial to have tools for statistical inference and model selection that help us remove the noise from the signal. These tools will be introduced in Chapter 5. In this chapter we describe some tricks and technical aspects important in the search for the best model, and for the development of the above mentioned inference tools.

4.1 Growing Trees of Fixed Size

In certain situations it is of interest to know what the best scoring tree or model of a certain size is. In this thesis, we use the total number of leaves in the Logic Trees involved in a model as the model size (see also introduction to Chapter 5 on page 70). For example, this is essential when using cross-validation to determine the best overall model size, as we will see later. To find the best scoring tree or model of a fixed size, we have to use simulated annealing (a greedy search could end before the desired model size is reached, and if it is reached, the model is not guaranteed to have the best score among models of that size). However, if the simulated annealing run is carried out with the move set as described in Section 2.1, 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 considered the following possibilities:

- Changing the move set:

  When altering a leaf or operator in a Logic Tree, the tree size does not change. When splitting a leaf or growing a branch, the tree size grows by one. When deleting a leaf or pruning a branch, the tree size decreases by one. Hence we can alter the move set in the following way: if we split a leaf, we also have to delete a leaf, and vice versa. If we grow a branch, we also have to prune a branch, and vice versa. These operations do not necessarily have to occur on the same tree if we have a model with multiple trees. We can choose our new move set using those moves, and including alternating a leaf or an operator as before. Regardless whether we have a model with one or multiple trees, the model size stays always constant. To find the best scoring model of a certain size, we can simply start a simulated annealing run with a randomly chosen tree or model of the desired size, and use the above described move set.

- Prohibiting moves past the limit:

  A simpler solution, without altering the move set, is to 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 us only to find the best of **up to** the desired size. However, smaller tree sizes are desirable in general, so this would not be a problem. In reality, the maximum (desired) tree size always seems to be reached anyways, which is hardly surprising. Besides being the simpler solution compared to the above method (since we don't have to alter the move set), it is also the more efficient way to grow models of a pre-specified size. The way the code is implemented, the first method mentioned requires a different form of evaluation, which is way more expensive computationally, since certain moves now have two "steps" (such as splitting and deleting leaves simultaneously).

4.2 Including Continuous Predictors and Factors

The methodology described so far requires that all predictors are binary. But often some of the predictors are continuous variables or factors with more than two possible outcomes. Below, we discuss possible solutions how to include those predictors in the Logic Regression framework.

4.2.1 Continuous Predictors

Fitting Continuous Predictors in a Model

In certain cases, we can incorporate a continuous variable just ”as is” into the model. For example, assume the binary predictors are $X_{b,1}, \ldots, X_{b,k}$, and the continuous predictors are $X_{c,1}, \ldots, X_{c,k}$. If we consider a logistic regression model, we could fit a model such as

$$\log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + ... + \beta_1 I_{\{L_1 \text{ is true}\}} + \beta_p I_{\{L_p \text{ is true}\}} + \beta_{c,1} X_{c,1} + \cdots + \beta_{c,k} X_{c,k} \quad (4.1)$$

where $L_1, \ldots, L_p$ are Logic Trees generated from the binary predictors. The model is fit exactly as described in Section 3.1.2, simply including the continuous predictors when calculating the score (deviance).

However, one of the main advantages of the Logic regression models with exclusively binary predictors is their simplicity, having easily interpretable rules that ”slice” the predictor space into few classes. Therefore, even though the inclusion of the continuous predictors might results in a model with the lowest deviance possible, this might not necessarily be a desirable thing to do.

Dichotomizing the Continuous Predictors before Fitting the Model

Using a splitting rule, we can dichotomize the continuous predictors before fitting the model. For example, so-called regression stumps are often used for dichotomizing variables. Regression stumps are the result of a single step fit of a decision tree. We split a continuous predictor (assigning zero to one class, one to the other) at the value that minimizes a certain criterion (for example the binomial log-likelihood or the misclassification rate). After dichotomizing all continuous predictors, we can include those with the other binary predictors into the model search.

Dichotomizing the Continuous Predictors while Fitting the Model

Dichotomizing a continuous predictor as described above is fast and simple. However, the split point for the variables found that way might not be optimal: if a dichotomized predictor is used in a model, a slightly different split might result in a lower score. Below we describe a method how we can optimize the split point, which however is computationally more expensive than the method described above. We usually use this technique when the search is via simulated annealing, but theoretically this can also be done during a greedy search. We dichotomize the continuous predictors at an arbitrarily chosen point. We enlarge the move set by allowing the split point to shift during the search. During a simulated annealing run, this shift is chosen randomly, during a greedy search all possible splits are considered. Below we show the results of this technique for a simulated annealing run in a simple example.

The data were generated having seven binary and three continuous variables. The predictors $1-7$ were binary with a total of 1000 cases each, and the value in each case of the predictors being a sample from a Bernoulli random variable with probability $\frac{1}{2}$. The predictors $8-10$ were samples from uniform distributions between 0 and 50, 50 and 100, and 100 and

150 respectively. The distributions were chosen to have different supports for clarity in the figure we show. The underlying true Boolean equation from which the response was smapled chosen to be

$$L = (X_1^c \vee X_2) \wedge [(X_8 > 25) \vee (X_9 > 50) \vee (X_1^c \wedge (X_{10} > 75)] \tag{4.2}$$

If for a certain case L was true, the response was sampled from a Bernoulli random variable with probability $\frac{2}{3}$, otherwise it was sampled from a Bernoulli random variable with probability $\frac{1}{3}$. Using simulated annealing, we found a model that involved all those predictors, and Figure 4.1 shows the splitpoints for each of the continuous predictors at the end of each Markov chain during the simulated annealing. A missing mark for a variable at a certain temperature indicates that the variable was not part of the current model at that state during the simulated annealing run. At higher temperatures, the splitpoints "jump", and frequently the variables are not part of the model at all. As the temperature decreases, the splitpoints converge to a limit that we use to dichotomize the continuous predictors in the final model.

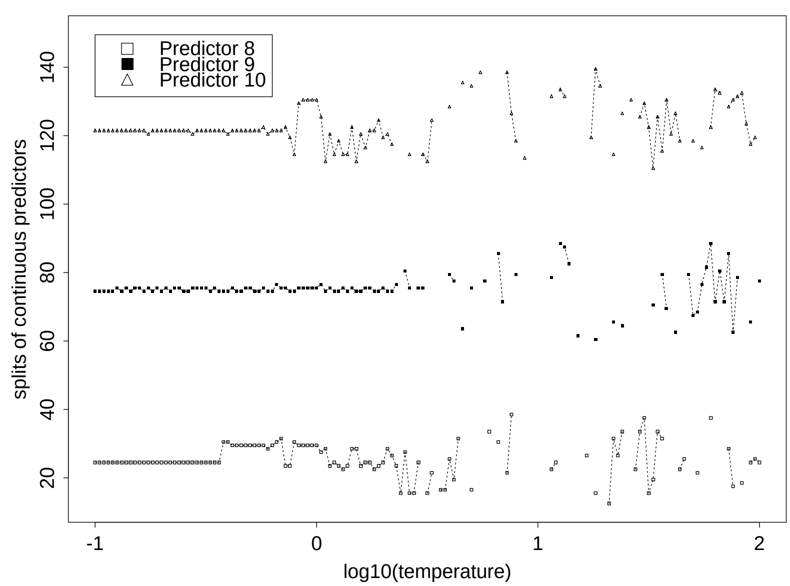


Figure 4.1: Split points in the simulated annealing run.

4.2.2 Categorical Variables with more than two Classes

Dummy Variables

A factor variable X with k possible outcomes can be re-written as $k-1$ so-called dummy variables, say $Z_1, \ldots, Z_{k-1}$. Without loss of generality, let $X \in \{1, \ldots, k\}$. If $X = 1$, then $Z_1 = \cdots = Z_{k-1} = 0$, and if $X = j > 1$, then $Z_{j-1} = 1$ and all other dummy variables are zero. Hence, instead of one variable with k factors we can include $k-1$ binary predictors into the search. This however might not always be a desirable solution, since introducing $k-1$ new binary variables might increase the complexity of the Logic Trees in the good scoring models substantially. Below we describe two possibilities how to generate one binary variable from the factor variable.

Ordered Factors

Often there is a natural ordering to factors, for example excellent, good, fair and poor as grades or rankings. Then the factors could be split into two groups, respecting the ordering. For example the above grades could be grouped as acceptable (excellent, good, fair) and non-acceptable (poor). The split point that defines those groups can either be pre-determined or updated during the simulated annealing, exactly as described in the above section on continuous predictors. If there is no natural ordering to the factors (for example if the factors are colors), an ordering could be induced by the association of each factor with the response (“the CART trick”, see Breiman et al [6]). This however might not be very desirable, and a potentially better way to handle those variables is described below.

Subgroups

The k factors can be classified into two groups, which will be determined throughout the simulated annealing. Initially, each factor gets randomly assigned to one group, assuring that there is at least one member per group. Then, as additional move in the simulated annealing, a factor can be transferred from one group to another, again assuring that there is at least one member per group. At the end of the simulated annealing we have two groups that represent an optimal assignment of the k factors into two groups.

4.3 Removing Redundancy from Logic Trees

As we have seen in for example in Section 3.1.2, the resulting Logic Trees from a simulated annealing run can contain redundancy. The two trees below in Figure 4.2 are equivalent in the sense that they yield exactly the same predictions for any possible case. How can we check if two trees predict the same (i. e. the underlying Boolean expressions are equivalent), or if a certain tree can be simplified (reduce the number of leaves, the depth of the tree, etc)?

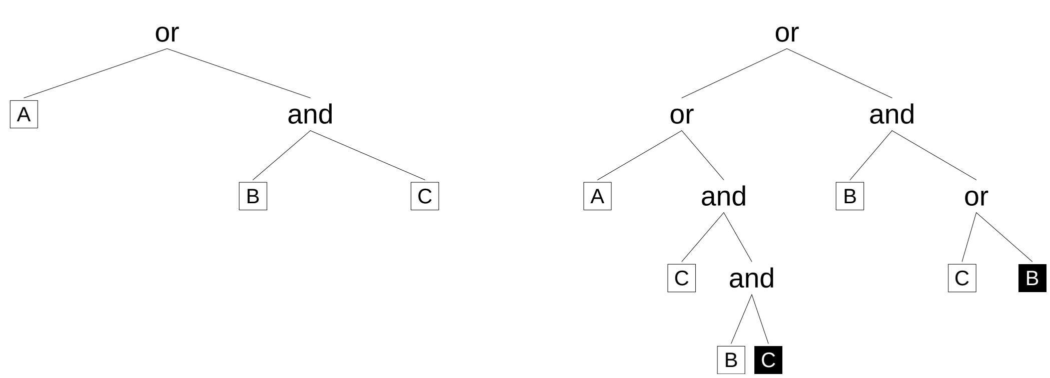


Figure 4.2: Two equivalent Logic Trees of different complexity.

One possibility is computational, using so-called truth tables, which will be discussed in Section 4.3.1. In Section 4.3.2 we investigate an alternative method, applying algebraic laws to Boolean expressions.

4.3.1 Comparing Trees using Truth Tables

Assume there are k predictors involved in a Logic Tree. Since every predictor is binary, there are at most 2^k different cases to consider. The prediction of the Logic Tree can be checked using a truth table, that presents the prediction of the tree for each of the 2^k different cases.

Table 4.1 and Table 4.2 show the truth tables for the Logic Trees in Figure 4.2(a) and Figure 4.2(b) respectively. In a truth table, we assign for a given prediction ($A, B, C \in \{0, 1\}^3$) a value to each occupied knots in the tree. We start with the leaves, and evaluate the tree bottom to top. The values in knot 1, the root, are the predictions for the respective cases.

Table 4.1: The truth table for the Logic Tree in Figure 4.2(a).

leaves			knots				
A	B	C	7	6	3	2	1
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	1
1	0	1	1	0	0	1	1
1	1	0	0	1	0	1	1
1	1	1	1	1	1	1	1

Table 4.2: The truth table for the Logic Tree in Figure 4.2(b).

leaves			knots												
A	B	C	22	21	15	14	11	10	7	6	5	4	3	2	1
0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0
0	1	0	1	1	0	0	1	0	0	1	0	0	0	0	0
0	1	1	0	1	0	1	0	1	1	1	0	0	1	0	1
1	0	0	1	0	1	0	0	0	1	0	0	1	0	1	1
1	0	1	0	0	1	1	0	1	1	0	0	1	0	1	1
1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	1
1	1	1	0	1	0	1	0	1	1	1	0	1	1	1	1

Comparing Tables 4.1 and 4.2, we see that no matter what value we assign to A, B, C, the prediction of the two trees is always the same, hence they are equivalent.

This method enables us to determine whether or not two trees are equivalent. It does not enable us to simplify a given tree if used as described above, unless the tree always predicts one or always predicts zero. A computationally and logistically very burdensome way could be to compare all subtrees of the tree under investigation to simple leaves or trees of smaller size, and replace a subtree with a smaller tree if a match is found.

4.3.2 Simplification of a Boolean Expression using Algebraic Laws

A tree can also be simplified if we write down the Logic Term it represents, and apply the algebraic laws introduced in Section 1.2.2. This strikes us as paper and pencil work, since the Logic Trees we consider are usually of reasonable size and the implementation of those algebraic laws on a computer seems very tricky.

As an example, the Logic Term for the tree in Figure 4.2(b) is

$$L = [A \vee (C \wedge (B \wedge C^c))] \vee [B \wedge (C \vee B^c)]. \tag{4.3}$$

Using the algebraic laws, we get

$$\begin{aligned}
A \vee (C \wedge (B \wedge C^c)) &\equiv A \vee ((B \wedge C^c) \wedge C)) && \text{commutative operators} \\
&\equiv A \vee (B \wedge (C^c \wedge C)) && \text{associative operators} \\
&\equiv A \vee (B \wedge (C^c \wedge (C^c)^c)) && \text{double complement} \\
&\equiv A \vee (B \wedge 0) && \text{complement law} \\
&\equiv A \vee 0 && \text{null law} \\
&\equiv A && \text{identity law}
\end{aligned}$$

$$\begin{aligned}
B \wedge (C \wedge B^c) &\equiv (B \wedge C) \vee (B \wedge B^c) && \text{distributive law} \\
&\equiv (B \wedge C) \vee 0 && \text{complement law} \\
&\equiv B \wedge C && \text{identity law}
\end{aligned}$$

Hence $L = A \vee (B \wedge C)$, which is the underlying Boolean Term of the tree in Figure 4.2(a).

4.3.3 Removing Redundancy by Numerical Means

Each model (or state) S has a score $\epsilon(S)$, and we denote the lowest global score by ϵ_0. If a tree in a model contains redundancy it means that this tree could be simplified without changing the score of the model. To find the simplest model among the best scoring models, we can distinguish all best scoring models by giving each of those models a bonus score, relative to its complexity:

- $\forall_S : [\epsilon(S) > \epsilon_0] \Longrightarrow \epsilon_{\text{new}}(S) := \epsilon(S)$,
- $\forall_S : [\epsilon(S) = \epsilon_0] \Longrightarrow \epsilon_{\text{new}}(S) := \epsilon_0 - \text{bonus}(S)$,

where bonus(S) is a positive function of the state S, that rewards states of low complexity. In our algorithms, we used for example the inverse of the number of leaves involved in the Logic Trees as bonus. This method requires that we know the optimal score. This can for example be achieved by running a simulated annealing twice. First without a bonus to determine ϵ_0, and then adding the bonus to the models that achieve this score.

Chapter 5

STATISTICAL INFERENCE AND MODEL SELECTION

Using simulated annealing gives us the good chance to find a model that has the lowest possible score. However, in the presence of noise in the data, we know that this model over-fits the data in general. This can also happen when using a greedy algorithm (see Section 2.2), although there is no guarantee of finding a lowest scoring model. In this section we describe some methods that allow to get a better grip on “what is signal, and what is noise” for models obtained through simulated annealing and greedy search.

To compare models, we also need a measure of model complexity. In this thesis, 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 also possible, such as the maximum number of leaves per Logic Tree, or the maximum of the depths of the Logic Trees involved in the model.

5.1 Cross-Validation

5.1.1 Standard Cross-Validation

We use cross-validation when the search is via simulated annealing. In principle, we can also use cross-validation when the search is greedy. However, the greedy search does not guarantee that we find the globally best scoring model, and hence cross-validation can lead to incorrect results and conclusions.

We chose the model size as the number of leaves in the Logic Trees involved in the model. Finding the globally best scoring model on the entire data, we know that we possibly have too many leaves in the Logic Tree, i. e. the optimal model possibly has a smaller size than the model found. We want to compare the performance of the best models for different sizes. Now assume that 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 data set 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 4.1), 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 score for model size k is $\epsilon_k = \frac{1}{m}\sum_i \epsilon_{ki}$.

Figure 5.1 shows the results of such a cross-validation. As in some previous examples, the data was generated by simulating 20 binary predictors, with a total of 1000 cases each, and

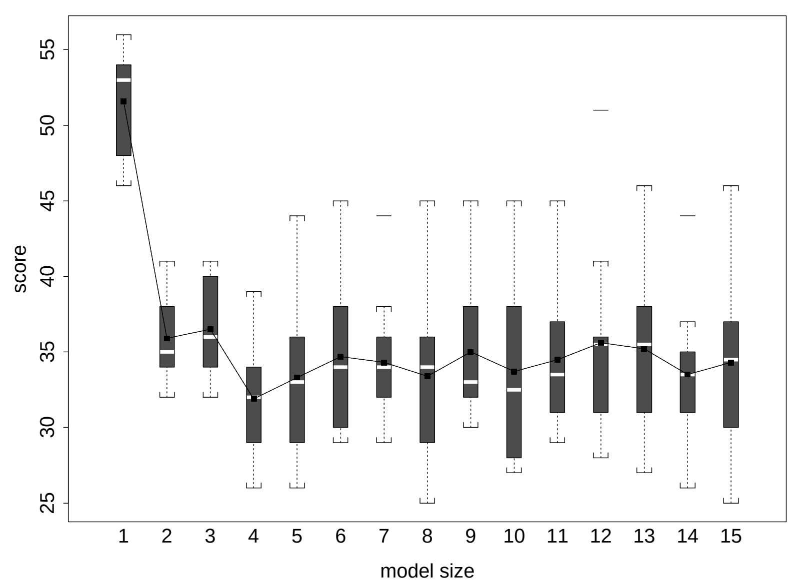


Figure 5.1: Cross-validated scores for a simple classification problem.

the value in each case of the predictors being a sample from a Bernoulli random variable with probability $\frac{1}{2}$. For simplicity, the underlying true Boolean equation was chosen to be

$$L = (X_1 \wedge X_2^c) \vee (X_3^c \wedge X_4) \tag{5.1}$$

If for a certain case L was true, the response was sampled from a Bernoulli random variable with probability $\frac{2}{3}$, otherwise it was sampled from a Bernoulli random variable with probability $\frac{1}{3}$. The score in the search was chosen to be the number of misclassifications (i. e. how often a proposed tree predicted the wrong response). We split the data in $m = 10$ subgroups, and the cross-validated scores for a fixed tree size is summarized in a box-plot in Figure 5.1. The means of the 10 cross-validated scores for each tree size are super-imposed onto the box-plots and connected by a solid line. The results are very typical for cross-validation: The average cross-validated score in general drops rapidly when the model size is increased, until the correct size is considered. Then, the scores very slowly increase with model size. There is usually a fair amount of variability in those cross-validated scores, and the average cross-validated score for the correct model size is not always the smallest among the averaged cross-validated scores. But cross-validation definitely provides a way to sensibly pick a reasonable model size.

An alternative, fairly similar method to the above described cross-validation is the training set/test set approach. If we have sufficient data, we can randomly assign the cases to two groups with pre-determined sizes, using one part of the data as the training set, and the other part as test set. That means, 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.

5.1.2 Simplified Cross Validation

The cross-validation described in Section 5.1.1 is a fairly unbiased method to select the optimal model size. However, computationally it is extremely expensive. For each model size under consideration (say sizes $1, \ldots, k$), and for each of the predetermined subsets (say $1, \ldots, m$) of the data for a specific model size, simulated annealing has to be used to find the best model. Hence a total of $m \times k$ simulated annealing runs have to be carried out to obtain the cross-validating scores. It would be desirable to have a computationally less expensive method that approximates those scores.

Assume we find the best scoring model of size k. We can prune each of the k leaves at a time and compare the scores of the resulting models. The best scoring model among those might not be the best overall scoring model of size $k-1$, but it usually is very similar. We pick the first of the m predetermined subsets as described in Section 5.1.1. We find the best scoring model of size k using the $m-1$ remaining subsets of cases as training set, and evaluate the first subset of cases, obtaining the score $\epsilon_{k,1}$. We then find the best model among those we can obtain by pruning one leaf. We evaluate the first subset of cases, and obtain the score $\epsilon_{k-1,1}$. We repeat this until we have score $\epsilon_{1,1}$. We then pick the second subset of cases as test set, and repeat the above described procedure, obtaining scores $\epsilon_{k,2}, \ldots, \epsilon_{1,2}$. We repeat this for the remaining subsets of cases, and obtain scores $\epsilon_{k,j}, \ldots, \epsilon_{1,j}$ for $j = 3, \ldots, m$. We then calculate $\epsilon_j = \sum_i \epsilon_{j,i}$, $j = 1, \ldots, k$, and use those as approximations of cross-validation scores. This procedure requires only m simulated annealing runs. The computing expense for the $m \times k$ pruning steps is negligible relative to the expense for the simulated annealing runs.

5.2 Randomization

5.2.1 Randomization for Greedy Searches

In a greedy search, we consider all possible moves from a given state, and select the move that improves the score the most. There is no guarantee that we will find the overall best scoring state this way, but as we have seen in Section 2.2, it is also possible that we over-fit the model. To distinguish what is signal and what is noise, we use randomization tests.

For each step in the greedy search, we calculate a p-value that reflects the evidence against the assumption (null hypothesis) that the improvement in score from the given model to the new model is due to noise. Figure 5.2 illustrates how the randomization test works for a model with a single tree. For one subset of the cases, the Logic Tree predicts zero,

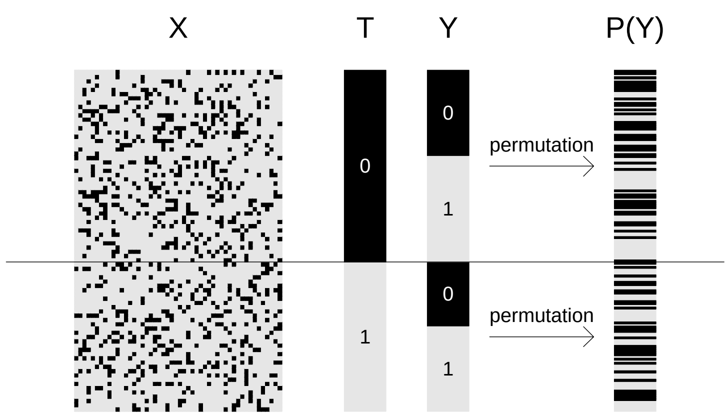


Figure 5.2: The setup for the randomization test for models with a single tree.

for the remaining cases it predicts one. If we randomly permutate the observations in the response within each of those subgroups, the model still scores the same, and the correlation structure between the predictors does not change. If we now again consider all moves from the given state, the move that improves the score the most might be different from the move selected given the original data, and the scores of the models resulting from those moves might be different, say ϵ_0 for the move using the original data, and $\epsilon_{\pi(1)}$ for the move using the data after the permutation. We repeat this procedure a predetermined number of times, and obtain the scores $\epsilon_{\pi(1)}, \ldots, \epsilon_{\pi(n)}$. We compare those scores to ϵ_0 and calculate the proportion of scores among $\epsilon_{\pi(1)}, \ldots, \epsilon_{\pi(n)}$ that are lower than ϵ_0. In case of a tie (this can happen for example when the score is the misclassification rate), we consider each tie as a ”half” count when calculating the above described proportion. If the null hypothesis is correct, we expect about half the scores among $\epsilon_{\pi(1)}, \ldots, \epsilon_{\pi(n)}$ to be lower than ϵ_0. If the proportion is very low, this is evidence that the improvement might actually be due to ”signal” in the data. The above described proportion is an exact p-value, reflecting the evidence against the assumption (null hypothesis) that the improvement in score from the given model to the new model (with score ϵ_0) is due to noise.

Figure 5.3 shows the outcome of the randomization test for the data used in the example in Section 2.2. We plotted the tree number as indicated by the sequence of trees in Figure 2.2 versus the p-values obtained in the randomization test. For the first six trees, we obtain low p-values (less or around 0.05). When testing tree 6 versus tree 7, we obtain a p-value of about 0.25, and hence pick tree 6 as the tree in our model. The correct tree is tree 7 however - it seems that we do not have enough cases in the data, and the randomization test does not have sufficient power.

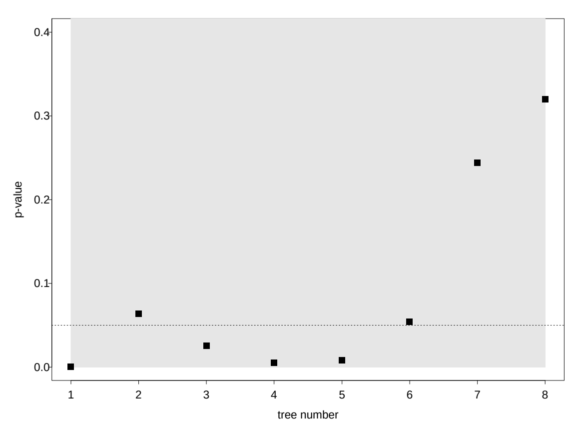


Figure 5.3: The p-values for a one-tree classification model obtained by a randomization test.

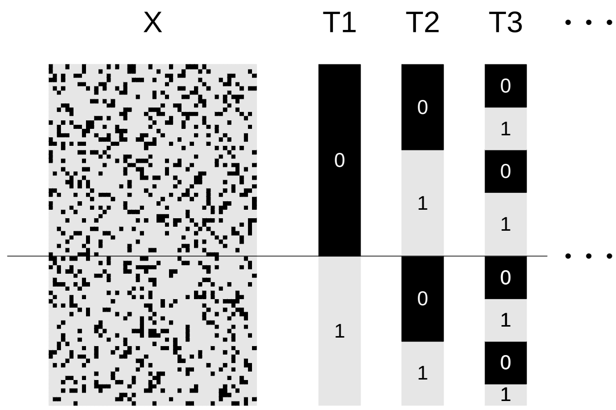


Figure 5.4: The setup for the randomization test for models with multiple trees.

If we have a model with multiple trees, the randomization is slightly different, which we try to illustrate in Figure 5.4. Assume we have k trees. For one subset of the cases, the first Logic Tree predicts zero, for the remaining cases it predicts one. For the first of those subsets, the second tree also predicts zero, but there might be cases for which it predicts one. For the second of those subsets, the second tree also predicts one, but there might be cases for which it predicts zero. This defines four subclasses. The third tree then determines two subclasses for each of those four classes, yielding a total of eight classes of cases, and so on. Hence there are 2^k groups of cases to consider. If we randomly permutate the observations in the response within each of those subgroups, the model still scores the same, and the correlation structure between the predictors does not change, and we can obtain a p-value in the same way as described above. Of course there should not be too many trees (and hence subgroups) relative to the number of cases.

5.2.2 Randomization for Simulated Annealing

In the previous sections we showed how to find the best scoring model of a certain class via greedy search or simulated annealing. The search will always find a best model, but the question whether or not there is any signal in the data at all still needs to be answered. Here, we introduce a randomization test that targets that problem and show a simple example.

Null Model Test: A Test for Signal in the Data

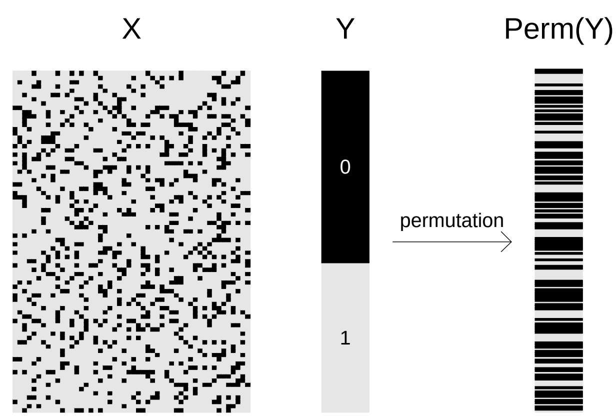


Figure 5.5: The setup for the null model randomization test.

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 signal in the data”. Now if that hypothesis was true, then the best model fit on the data with the response randomly permutated (as indicated in Figure 5.5) should yield about the same score as the best model fit on the original data. 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 a p-value, indicating evidence against the null hypothesis.

A Test to Detect the Optimal Model Size

For any model class we consider in our methodology (linear regression, logistic regression, etc) we first find the best scoring model, given the data, with score ϵ^*. We also find the best scoring models for a range of sizes, say 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 lower score obtained by models of larger sizes is due to noise." Here, j can be each model size we consider ($j \in \{0, \ldots, k\}$). 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 classes that the Logic Trees predict. For a model with p

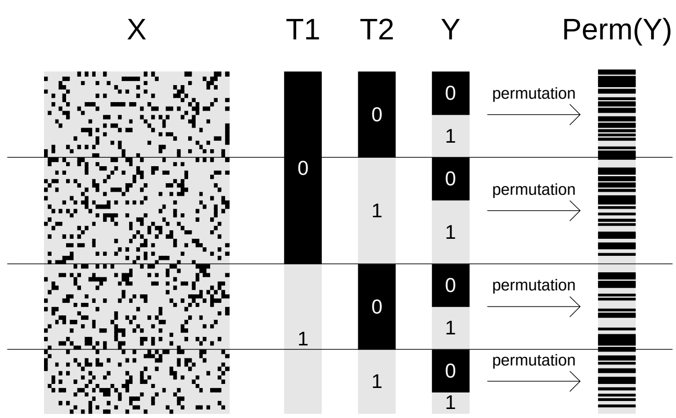


Figure 5.6: The setup for the sequential randomization test.

trees, those can be up to 2^p classes. Figure 5.6 shows the setup for models with 2 trees. We now randomly permute the response within each of those classes. The model of size j considered best still scores exactly the same (ϵ_j). If we now grow the overall best model (of any size), it will have a score (ϵ^{**}) as least as good, but usually better than ϵ_j. However, we know that this is due to noise! If the null hypothesis was true, and the model of size j was indeed the best, then ϵ^* would be a sample from the same distribution as ϵ^{**}. We can estimate this distribution as closely as desired by repeating this procedure multiple times. On the other hand, if the best model had a size larger than j, then the randomization would yield on average lower scores than the average of scores obtained by conditioning on the model of optimal size (larger than j).

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, etc. Comparing the distributions of the randomization scores, we can make a decision which model size to pick. See the applications in Chapter 6 for some examples.

Chapter 6

APPLICATIONS

6.1 The Cardiovascular Health Study

In this section we analyze some data that were gathered as part of the Cardiovascular Health Study. We briefly introduce the basics of the study and the data we use, and refer the reader to Fried et al [15] for a more detailed description of the study protocol and the goals of the study.

The Cardiovascular Health Study (CHS) is a study of coronary heart disease and stroke in elderly people. Between 1989 and 1990, 5201 subjects over the age 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 3647 CHS participants, MRI detected strokes (infarcts bigger than 3mm that led to deficits in functioning) 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 (that is, 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. For more details on the scanning procedure, see Bryan et al [7]. In Table 6.1 we list the 23 regions of the CHS atlas and the number of

Table 6.1: 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.

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 (AICA)	7
	11	Posterior Inferior Cerebellar Artery (PICA)	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

Table 6.2: The standard anatomical clusters of the 23 locations in the CHS atlas of the brain.

A	Regions	1-8	Cerebral Cortex
B	Regions	9-11	Cerebellar Cortex
C	Region	12	Cerebellar White Matter
D	Regions	13-17	Basal Ganglia
E	Regions	18-20	Brain Stem
F	Regions	21-23	Cerebral White Matter

patients with infarcts in those regions. The letters in the first column of Table 6.1 represent the standard anatomical clusters of the above 23 locations, as labelled in the thesis of Robyn McClelland [29], and shown in Table 6.2.

One of the goals of the Cardiovascular Health Study is to assess the association between stroke locations and various response variables. One of those response variables is a score derived 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 [49].

The objective of our analysis was mainly to demonstrate the use of the Logic Regression methodology. To seriously analyze those data, it would be crucial to interact with people more familiar with the study, the anatomy of the brain, etc. Maybe only a subset of the predictors should be included in the analysis since for example there are obviously spatial correlations between predictors. McClelland [29] also reported an improved performance of her algorithm after adjusting for gender, age, etc. These variables were not available to us when we started analyzing the data, so we did not include those into our model, although the methodology to do so was introduced in Section 4.2.

For technical reasons we used a modified version of the mini-mental score as response. Since most patients scored in the upper 90s in the mini-mental test, a logarithmic transformation seemed beneficial after investigating potential model violations such as non-normally distributed (skewed) residuals. Since the mini-mental score is a number between 0 and 100, we defined as response variable

$$Y := \log(101 - \text{mini-mental score}). \tag{6.1}$$

Usually such a transformation is avoided because it makes the parameter interpretation and their association to the original mini-mental score almost impossible, but since in our methodology we only deal with a very small number of classes of patients (see 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

$$Y = \beta_0 + \beta_1 \times I_{\{L_1 \text{ is true}\}} + \cdots + \beta_p \times I_{\{L_p \text{ is true}\}} + \epsilon, \tag{6.2}$$

with $\epsilon \sim N(0, \sigma^2)$ and various numbers p of Logic Terms. We used the residual sums of squares as scoring function. These models define groups of patients with similar stroke patterns in the brain atlas within each group, and different average scores between groups.

We first carried out a ”null model” randomization test as described in Section 5.2.2. Figure 6.1 shows the results from the randomization test fitting only one tree, allowing up to 16 leaves in tree. The null model, simply fitting the intercept β_0 with $\beta_1 = 0$ had a score of 2618.5. Using simulated annealing, the best scoring model had a score of 2555.7. We permuted the response as decribed in Section 5.2.2, refit the model, recorded the score, and repeated this procedure 250 times. In Figure 6.1 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 scores are considerably higher than the score [a], we can safely conclude that there is information in the predictors with disrciminatory

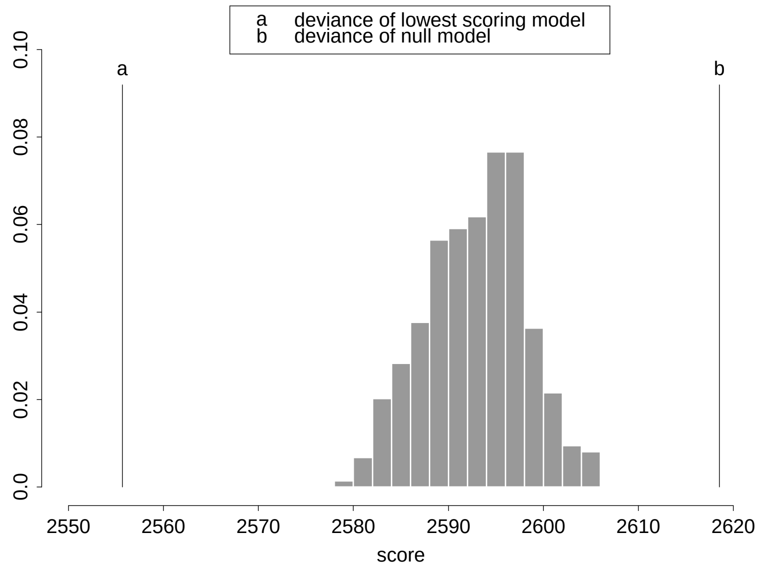


Figure 6.1: The results of the null model randomization test. The scores in the histogram came from linear models with a single tree, allowing up to 16 leaves.

power for the (transformed) mini-mental score. The results we got using models with more than one tree were very similar.

To get a better idea of the effect of varying model sizes, we show the training scores of linear models with one, two and three trees, and varying tree sizes in Figure 6.2. 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 lower (training) score. The more trees we allow in the model, the lower the score for a fixed number of leaves. Allowing a second tree in the linear models seems to has 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

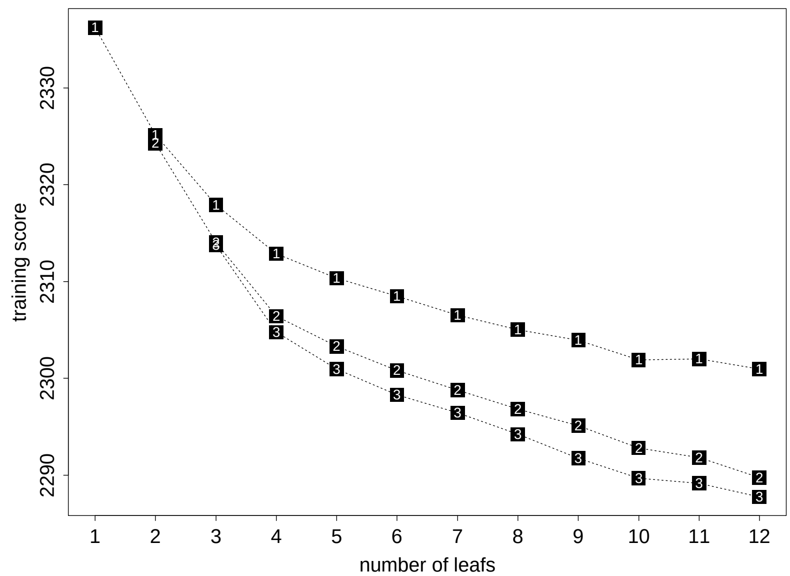


Figure 6.2: Training scores of the best linear models for a fixed number of leaves and trees. The number of trees allowed in the linear model is the white number super-imposed on the black squares.

trees.

However, the comparison of effects allowing additional trees in the models does not hold up for the cross-validated scores, as shown in Figure 6.3. The cross-validated scores for models with identical number of leaves but different tree sizes look remarkably similar. The scores of the models with only one tree are connected by a solid line, since obviously allowing for more trees does not seem very benefitial, and a model with only one tree and four leaves is the best choice in terms of simplicity among the models with low cross-validated scores. We also added cross-validated scores obtained from ”standard” linear models to the plot (open circles): a linear model with main effects only is fit for the training set, and using the *step* function in the Splus software package [1] we trim the model of one

[1]Splus uses the AIC criterion to select which predictor to drop, which might not be the one that results in the lowest residual sum of squares in the training set (say model M). In this case, the cross-validated score

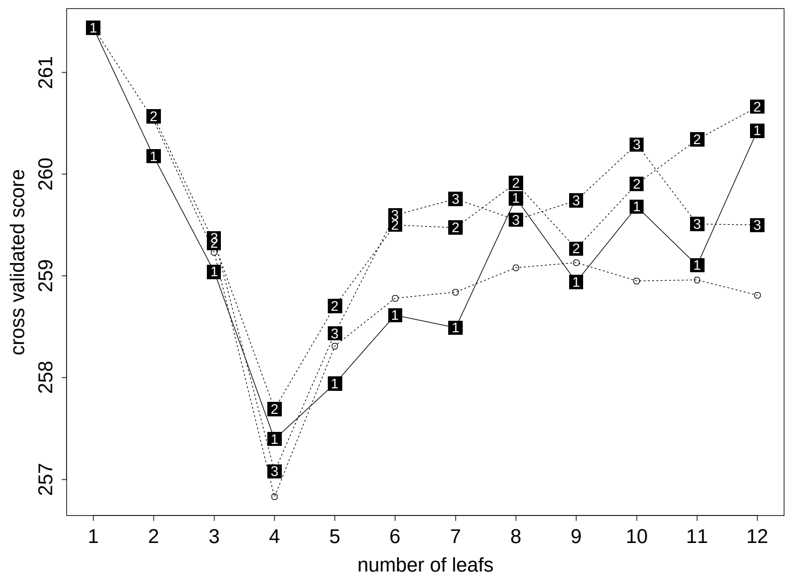


Figure 6.3: Test scores of the linear models obtained from the trainig set for a fixed number of leaves and trees. Again, the number of trees allowed in the linear model is the white number super-imposed on the black squares. The open circles indicate cross-validated scores obtained from ”standard” linear models.

predictor at a time, and calculate the score for the test data under the given model. Although the scores from the models using Logic Trees and the scores from linear models with main effects only are not quite comparable, we aligned the cross-validated scores in Figure 6.3 such that the number of predictors involved (but not the number of parameters fitted!) in the models are the same. It seems surprising at first glance that those scores and the scores from the Logic Models are very much alike. We will see why this happens when we further investigate those models.

We used ten-fold cross-validation in this example, so we multiplied the training scores by a factor of $\frac{10}{9}$ and the cross-validated scores by a factor of 10, which enabled us to compare those scores with the scores of models fit on the entire data set, plotted in Figure 6.4. While

can actually be higher or lower than the cross-validated score under M.

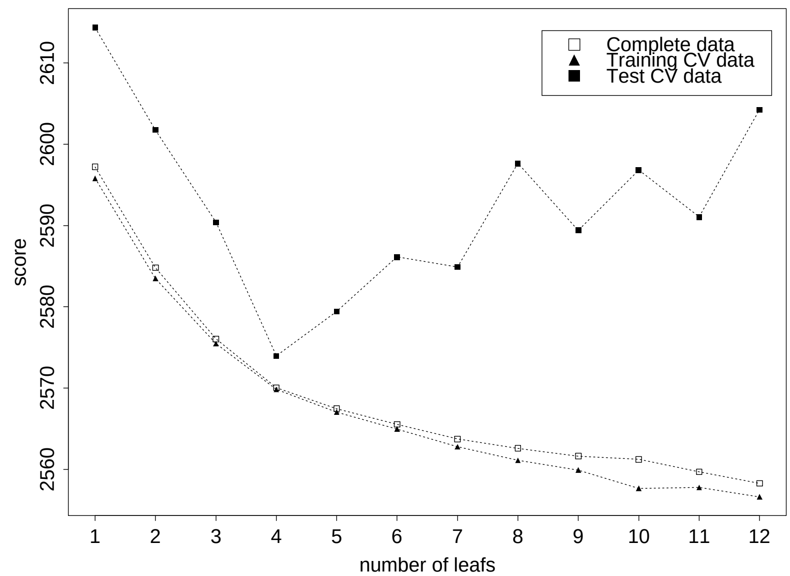


Figure 6.4: Comparison of (adjusted) scores from cross-validation and model fitting on the complete data.

the adjusted training scores from the cross-validation and the scores from the models fit on the entire data set are very similar, the vast gap between those scores and the test scores from the cross-validation, in particular for the models with one or two leaves only, is quite remarkable and strongly discourages the selection of the smallest models.

The best results using cross-validation were obtained for the models with with four leaves, and we prefer the simplest model with only one tree. This selection is encouraged by the results of the randomization test for model selection, as described in Section 5.2.2. Figure 6.5 shows histograms of the randomization scores after conditioning on 0 (null model), $1, 23, 4$ and 5 trees, indicated by the number in the upper left corner of each panel. The right bar in each panel indicates the score of the null model, and the left bar indicates the lowest overall score found for the original data. This score looks like another sample from the distribution of scores of the randomized data after conditioning on four trees.

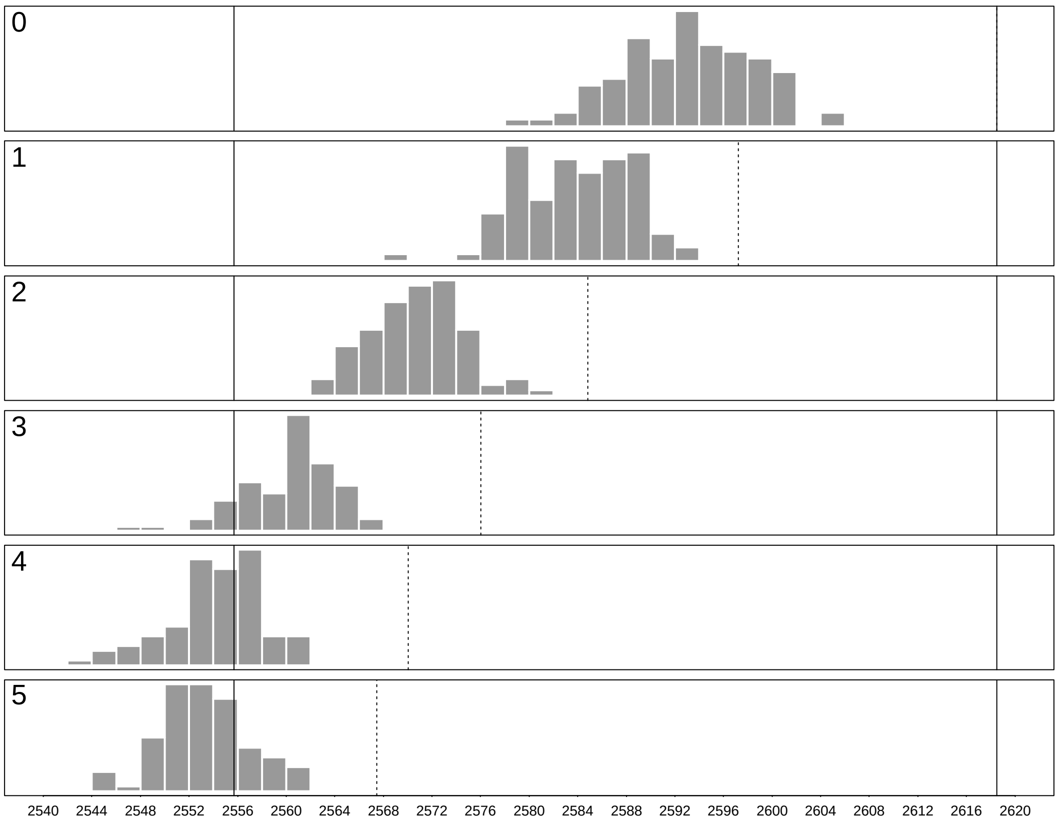


Figure 6.5: The results of the randomization test for model selection, supporting the choice of four leaves in the selected model.

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

$$Y = 2.32 - 0.36 \times I_{\{L \text{ is true}\}} \tag{6.3}$$

with the Logic Term

$$L = X_{12}^c \wedge [(X_{19}^c \wedge X_4^c) \wedge X_{17}^c] = X_{12}^c \wedge X_{19}^c \wedge X_4^c \wedge X_{17}^c, \tag{6.4}$$

shown as Logic Tree in Figure 6.6.

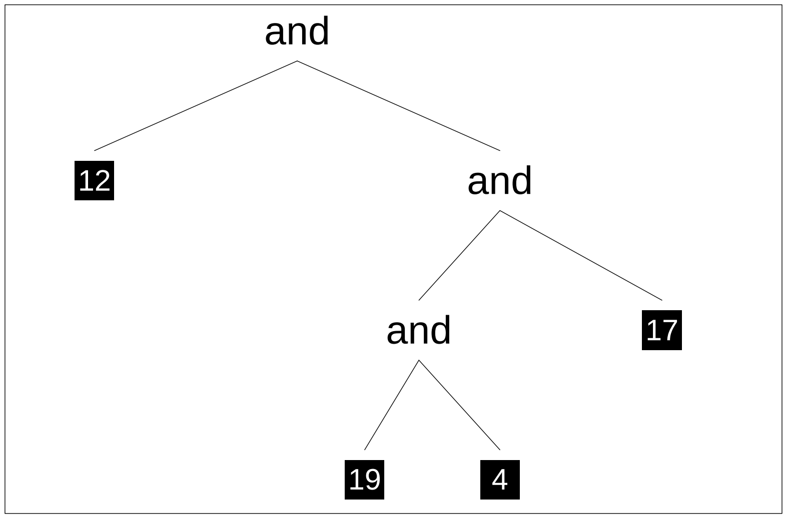


Figure 6.6: The Logic Tree in the selected model in equation (6.3)

Using the complement rule, the above model can also be written as

$$Y = 1.96 + 0.36 \times I_{\{L^c \text{ is true}\}} \tag{6.5}$$

with

$$L^c = X_{12} \vee X_{19} \vee X_4 \vee X_{17}. \tag{6.6}$$

It now becomes clear why the linear model using main effects only looked so similar to the Logic Models in the comparison of the cross-validated scores. 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. However, it can be seen in Figure 6.7 that for example among those 236 patients with an infarct in region 17, only 2 have also an infarct in region 4.

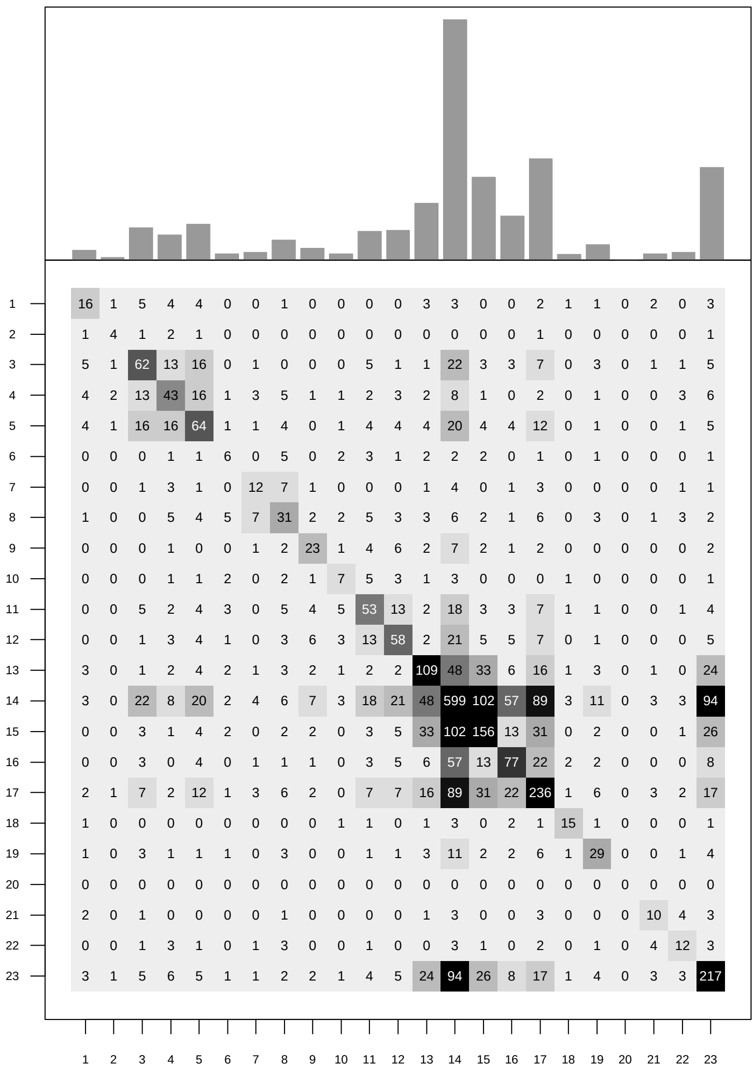


Figure 6.7: The counts of patients having strokes in at least two locations. Along the diagonal are the counts of patients having a stroke in one particular region of the brain, with the marginal distribution added to the plot.

Considering the presence of infarcts, these four predictors are pairwise almost exclusive. Since the Logic Term in (6.6) has only $\vee$ operators, the above model is very similar to a linear model with these four predictors as main effects, summarized below.

	$\hat{\beta}$	$\hat{\sigma}$	t-value	p-value
Intercept	1.961	0.015	133.98	< 0.001
Region 4	0.524	0.129	4.06	< 0.001
Region 12	0.460	0.112	4.09	< 0.001
Region 17	0.236	0.057	4.17	< 0.001
Region 19	0.611	0.157	3.89	< 0.001

If we fit those four predictors with a single paramter, we get which is almost exactly the same model as (6.5).

	$\hat{\beta}$	$\hat{\sigma}$	t-value	p-value
Intercept	1.960	0.015	133.84	< 0.001
Region 4	0.344	0.044	7.84	< 0.001
Region 12	0.344			
Region 17	0.344			
Region 19	0.344			

However, even though the linear models look very similar to the model in (6.5), 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 (6.5), we can state that the estimated median mini-mental score for patients with infarcts in either region $4, 12, 17$ or 19 was 90.8, compared to 93.9 for patients that did not have an infarct in any of those regions.

6.2 Prognosis after Acute Myocardial Infarction

We briefly introduce the data we use in this section and describe the basics of the study they arouse from. We refer the reader to Henning et al [19] and Gilpin et al [17] for a more detailed description of the study and data. A subset of the data were also analyzed by Breiman et al [6].

The data investigated in this section of the thesis were gathered in the 1970s at the University of California Medical Center, San Diego and the Vancouver General Hospital, British Columbia. The subjects in the study are patients who were admitted to the hospital within

Table 6.3: Thirteen variables with predictive power for early mortality after acute myocardial infarction.

Type	Predictor	Name
Binary	1	Gender
	2	History of previous myocardial infarctions
	3	History of previous congestive heart failure
	4	History of angina
	5	Persistent pain
	6	Atrial fillrilation
	7	Sinus brachiacardia
	8	Sinus tachycardia
Continuous	9	Age
	10	Maximum heart rate
	11	Minimum systolic blood pressure (24h)
	12	Maximum creatine kinase
	13	Maximum blood urea nitrogen

24 hours of the onset of the symptoms of acute myocardial infarction, and who survived at least for 24 hours after admission. One goal of the study was to identify high risk patients (patients who will not survive the first 30 days) on the basis of measurements, taken within 24 hours after admission, that were considered indicators of each patient's condition. Henning et al [19] find that nineteen of those variables measured had predictive power for early mortality. Available to us were measurements of thirteen of those variables on 1780 patients, plus a variable indicating which patients did not survive the first 30 days. Table 6.3 separately lists the names of the binary and continuous predictors.

In Section 4.2 we described possibilities how to include continuous predictors in Logic Regression. However, the objective of the analysis was to keep it simple and only find a few subclasses of patients with vastly different prognosis of early mortality after acute myocardial infarction, we decided to dichotomize the continuous predictors instead of fitting them as separate predictors in the model. For simplicity we used regression stumps from decision trees to dichotomize these continuous variables, instead of estimating a split point during simulated annealing. The continuous predictors after being dichotomized now have two classes, say high and low, with the low class being defined in the chart below.

Age	less than 64.
Maximum heart rate	lower than 103.
Minimum systolic blood pressure	lower than 79.
Maximum creatine kinase	lower than 1638.
Maximum blood urea nitrogen	lower than 28.

The models we investigated were logistic regression models of the form

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \times I_{\{L_1 \text{ is true}\}} + \cdots + \beta_k \times I_{\{L_k \text{ is true}\}} \tag{6.7}$$

with at most $k = 3$ Logic Terms, and p denoting the probability of not surviving the first 30 days. Using the deviance as scoring function, we can find groups of patients with similar measurements of condition within groups and different prognosis of early mortality after acute myocardial infarction between groups, as desired. Of the 1780 patients, 315 died within 30 days after admission to the hospital. For those cases, a weight of 3 (compared to weight 1 for the survivors) has been suggested in the literature, which we use in the model fitting process.

We first carried out a ”null model” randomization test as described in Section 5.2.2. Figure 6.8 shows the results from the randomization test fitting three trees, allowing up to 16 leaves in each tree. The null model, simply fitting the intercept β_0 with $\beta_1 = \beta_2 = \beta_3 = 0$ had a score of 3227.9. Using simulated annealing, the best scoring model we found had a score of 2641.7. We permuted the response as decribed in Section 5.2.2, refit the model, recorded the score, and repeated this procedure 250 times. In Figure 6.8 we compare the score for the best scoring model [a], the score for the null model [b], and a histogram of the scores

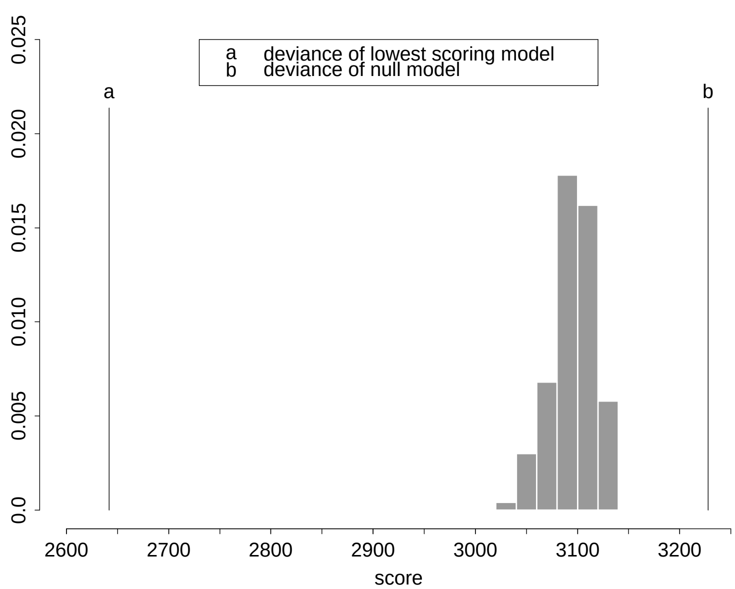


Figure 6.8: The results of the null model randomization test. The scores in the histogram came from a logistic regression model with three trees, allowing up to 16 leaves in each tree.

obtained from the randomization procedure. Since all of those scores are considerably higher than the score [a], we can safely conclude that there is information in the predictors with disrciminatory power for the prognosis of surviving 30 days after admission. The results we got using models with fewer trees were very similar.

In Figure 6.9 we show the training scores of logistic regression models with one, two and three trees, and varying tree sizes. Again, the results are as expected: comparing two models with the same number of trees allowed, the model involving more leaves has a lower (training) score. The more trees we allow in the model, the lower the score for a fixed number of leaves. Adding a tree seems to have a rather large affect, both in going from one to two and from two to three trees.

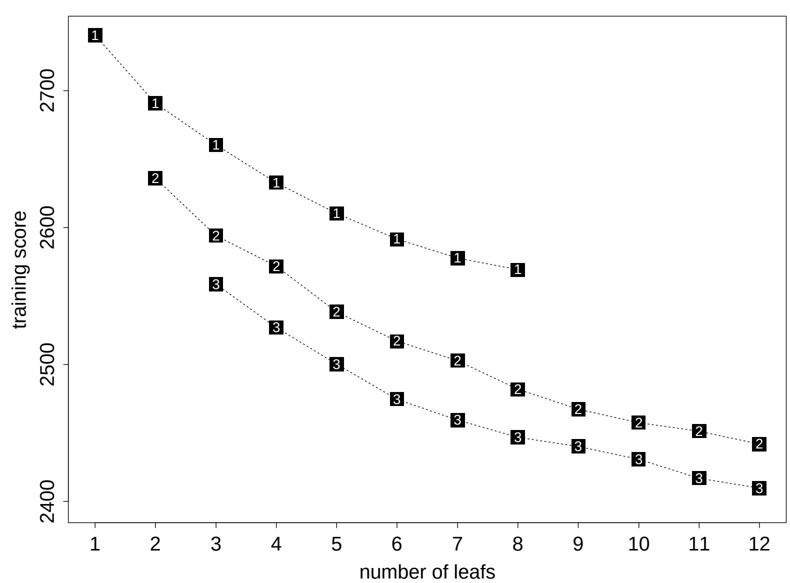


Figure 6.9: Training scores of the best logistic regression models for a fixed number of leaves and trees. The number of trees allowed in the linear model is the white number super-imposed on the black squares.

In the plot of the cross-validated scores for the models with up to three trees (Figure 6.10),

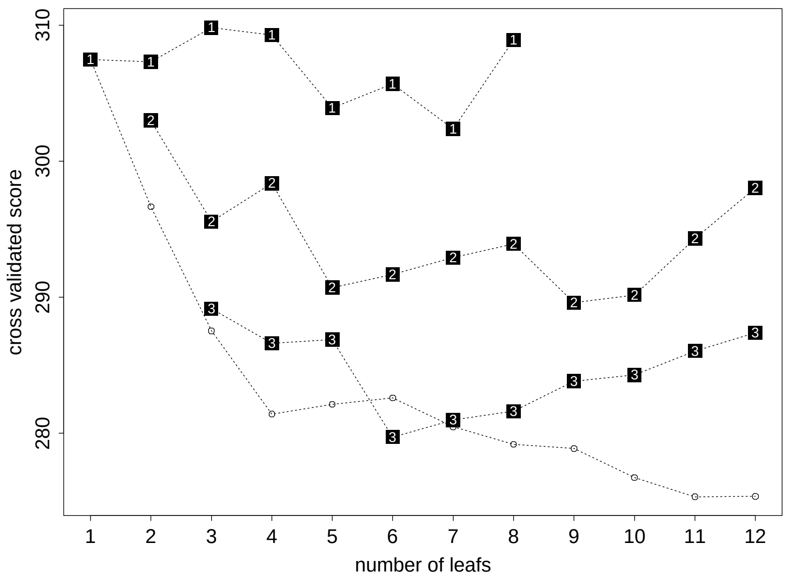
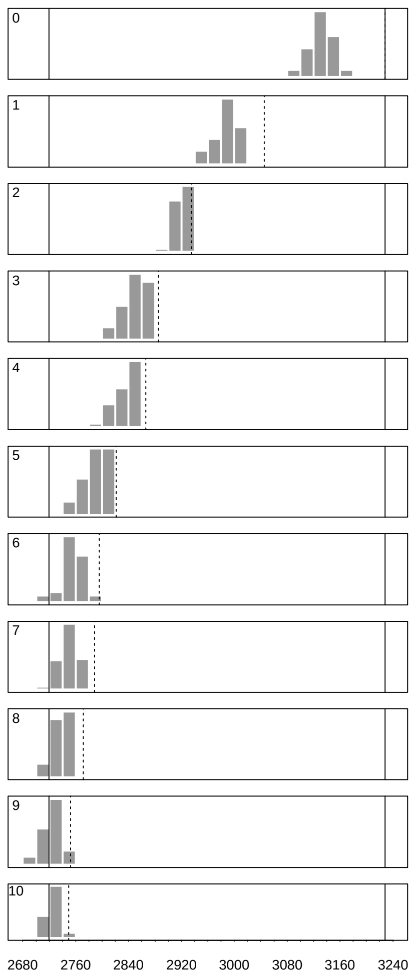


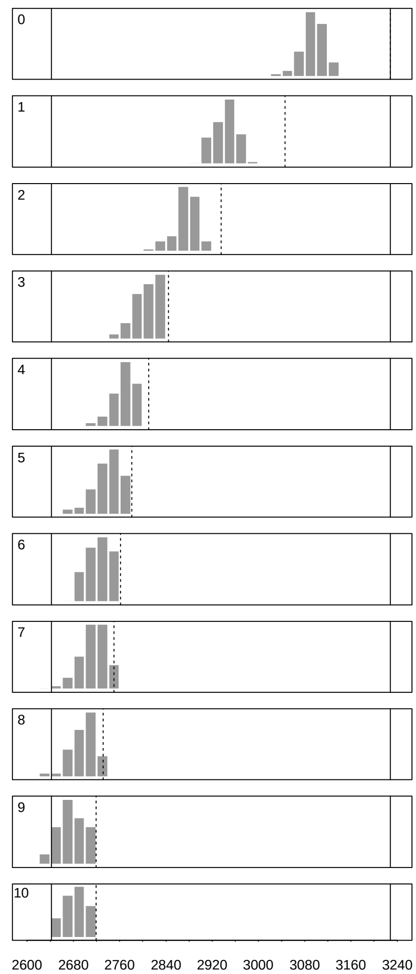
Figure 6.10: Test scores of the logistic models obtained from the training set for a fixed number of leaves and trees. Again, the number of trees allowed in the logistic model is the white number super-imposed on the black squares. The open circles indicate cross-validated scores obtained from ”standard” logistic regression models.

we aligned the cross-validated scores obtained from ”standard” logistic regression [2] (open circles) such that the number of predictors involved (but not the number of parameters fitted) in the models are the same. The cross-validated scores improve with the number of trees allowed in the model. For models with three trees six leaves seem to be optimal, for models with only two trees the decision is not very clear. More informative are the plots in Figure 6.11, showing the results of the randomization test for model selection (see Section 5.2.2). For both cases, logistic models with two and three Logic Trees, we get a substantial improvement in the randomization score when we increase the model size we condition on, until about models with six leaves. After that, adding a seventh, eighth, etc leaf to condition on still improves the randomization scores somewhat, but not as much

[2] Again, we used the *step* function in the Splus. See the previous section for a more detailled description.



(a) Results for logistic models with two Logic Trees.



(b) Results for logistic models with three Logic Trees.

Figure 6.11: The results of the randomization test for selection, of logistic models with two and three tress. The right bar in each panel indicates the score of the null model, and the left bar indicates the lowest overall score found for the original data. The number in the upper left corner of each panel indicates the model size we condition on in the randomization test.

as we saw before. The cross-validation scores in Figure 6.10 are the averages obtained from the ten-fold cross-validation. The variation of the cross-validation scores for any model is actually much larger than the differences of means we observe between models with the same number of trees and similar sizes (number of trees), which explains why the cross-validation scores do not strictly decrease with the increase of model size, as the randomization scores in Figure 6.11 illustrate.

The objective of our analysis was to find a few subclasses of patients with different prognosis of early mortality after acute myocardial infarction. We wanted these subclasses to be defined as simple as possible, and considered models with up to ten leaves that generate not more than eight classes. Among those we found that the model fit can be increased by adding on predictors. This is hardly a surprise, considering that all variables had predictive power for the outcome (Henning et al [19]). However, with our desire of simplicity, we decided that the increase in model size does not warrant the smaller and smaller enhancement in model fit after about six leaves, in both models with two or three trees.

As an example and to give an interpretation of our results we searched for the best logistic regression model with two trees and six leaves using the complete data, which yielded the following:

$$\log\left(\frac{p}{1-p}\right) = -0.05 - 1.32 \times I_{\{L_1 \text{ is true}\}} - 1.20 \times I_{\{L_2 \text{ is true}\}} \tag{6.8}$$

with the Logic Terms

$$L_1 = (X_{13} \wedge X_3^c) \wedge (X_9 \vee X_5^c) = X_{13} \wedge X_3^c \wedge [X_9 \vee X_5^c] \tag{6.9}$$

and

$$L_2 = X_{10} \wedge X_{11}^c \tag{6.10}$$

shown as Logic Trees in Figure 6.12.

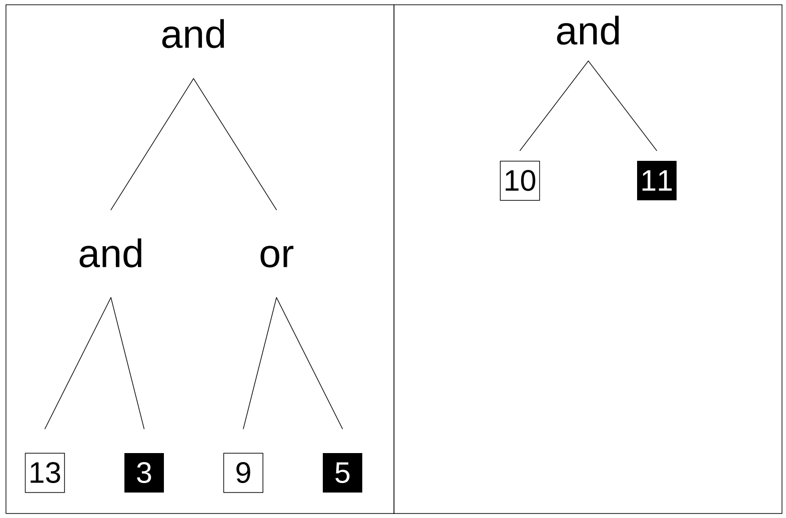


Figure 6.12: The Logic Trees in the model in equation (6.8). The leaves represent the predictors listed in Table 6.3.

Since we used weights in the logistic regression model, we actually first found the Logic Trees, and then refit the model without weights. This leaves the parameters for the Logic Trees unchanged, but yields an intercept that we can use in interpreting the model. The fitted probabilities and counts for the four classes defined by the above model are shown in Table 6.4.

The simple model interpretation is the following:

- If one of the two following conditions are true, the patient has roughly an 80% chance of survival:
  - The patient has a maximum blood urea nitrogen measure of at least 28, no history of previous congestive heart failure and is either at least 64 years of age or does not report persistant pain (or both).

  - The patient's maximum heart rate is at least 103 and his minimum systolic bloodpressure within the first 24 hours after admission is lower than 79.

- If both of the above conditions are true, the patient's chance of survival is higher than 90%.

- If none of the above conditions is true however, the patient's chance of survival is only 50%.

The result that older patients have a better chance of survival seems odd, but in the data we observe indeed an 88% survival rate among patients who are at least 64 years of age, compared to 76% among the younger patients.

Table 6.4: The fitted probabilities of mortality and counts for the four classes defined by the model in (6.8). F stands for false, T for true. For example, for 884 patients both L_1 and L_2 were true. These 884 patients have an estimated 7% mortality chance.

		probabilities		counts	
		L_1		L_1	
		F	T	F	T
L_2	F	50	21	224	209
	T	22	7	463	884