A METHOD FOR DETECTING OPTIMAL SPLITS OVER TIME IN SURVIVAL ANALYSIS USING TREE-STRUCTURED MODELS

by

Leighton Scott Dean

BS, University of Tennessee - Knoxville, 1993

MBA, West Virginia University, 1995

Submitted to the Graduate Faculty of

the Graduate School of Public Health in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2007

UNIVERSITY OF PITTSBURGH

Graduate School of Public Health

This dissertation was presented

by

Leighton Scott Dean

It was defended on

April 6, 2007

and approved by

Disseration Director:
Stewart J. Anderson, PhD, Professor, Department of Biostatistics,
Graduate School of Public Health, University of Pittsburgh

Vincent C. Arena, PhD, Associate Professor, Department of Biostatistics,
Graduate School of Public Health, University of Pittsburgh

Joseph P. Costantino, PhD, Professor, Department of Biostatistics,
Graduate School of Public Health, University of Pittsburgh

Sheryl F Kelsey, PhD, Professor, Department of Epidemiology,
Graduate School of Public Health, University of Pittsburgh

Sati Mazumdar, PhD, Professor, Department of Biostatistics and Psychiatry,
Graduate School of Public Health, University of Pittsburgh

Stewart Anderson, Ph.D.

A METHOD FOR DETECTING OPTIMAL SPLITS OVER TIME IN SURVIVAL ANALYSIS USING TREE-STRUCTURED MODELS

Leighton Scott Dean, PhD

University of Pittsburgh, 2007

One of the most popular uses for tree-based methods is in survival analysis for censored time data where the goal is to identify factors that are predictive of survival. Tree-based methods, due to their ability to identify subgroups in a hierarchical manner, can sometimes provide a useful alternative to Cox's proportional hazards model (1972) for the exploration of survival data. Since the data are partitioned into approximately homogeneous groups, Kaplan-Meier estimators can be used to compare prognosis between the groups presented by "nodes" in the tree. The demand for tree-based methods comes from clinical studies where the investigators are interested in grouping patients with differing prognoses. Tree-based methods are usually conducted at landmark time points, for example, five-year overall survival, but the effects of some covariates might be attenuated or increased at some other landmark time point. In some applications, it may be of interest to also determine the time point with respect to the outcome interest where the greatest discrimination between subgroups occurs. Consequently, by using a conventional approach, the time point at which the discrimination is the greatest might be missed.

To remediate this potential problem, we propose a tree-structure method that will split based on the potential time-varying effects of the covariates. Accordingly, with our method, we find the best point of discrimination of a covariate with respect to not only a particular value of

that covariate but also to the time when the endpoint of interest is observed. We analyze survival data from the National Surgical Adjuvant Breast and Bowel Project (NSABP) Protocol B-09 to demonstrate our method. Simulations are used to assess the statistical properties of this proposed methodology.

We propose a new method in survival analysis, which is an area of statistics that is commonly used to assess prognoses of patients or participants in large public health studies. Our proposed method has public health significance because it could potentially facilitate a more refined assessment of the effect of biological and clinical markers on the survival times of different patient populations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

I would like to thank the members of my committee: Dr Vincent C Arena, Dr Joseph P Costantino, Dr Sheryl F Kelsey, and Dr Sati Mazumdar for their valuable input and guidance during the course of this work. I would especially like to thank my advisor Dr Stewart Anderson, who I have learned so much from while working with him throughout my graduate studies. His guidance and support over the years has been invaluable.

I would like to thank Dr May Nawal Lutfiyya. It was her mentorship that helped lead me to the path that I follow today.

Finally, I would like to express a special thanks to my family whom without I could never have reached this goal. I would like to thank my parents, Rodney and Donna, and my brother, Chad for their support. I would especially like to thank my wife, Stacey, and my daughter, Ava, for their love, patience and understanding. It has been a long road but it has been worth it with them behind me.

1.0 INTRODUCTION

Tree structure methods have become a powerful tool to study patterns in data analysis. Morgan and Sonquist (1963) first introduced the idea of recursive partitioning for handling interactions in social science data; they called it “automatic interaction detection” (AID). However, the approach only gained popularity about 20 years later with the concepts presented by Breiman, et al. (1984) in their book, "Classification and Regression Trees" (CART). Breiman, et al., and others have extended these CART methods to many types of regression (linear, logistic, survival) problems.

The focus of this dissertation is on survival regression trees. An extension to the CART method developed by Breiman, et al. was proposed by Therneau and Adkinson (1997) and implemented as an algorithm called “RPART”. Our proposed method extends the concept used by Therneau and Adkinson to determine if the effects of predictive markers change over time and if the optimal split-points for these markers change with time. Additionally, we will seek to ascertain when the optimum time to examine such markers occurs. For example, is the “optimal” time to look at an outcome, such as overall survival, a landmark time point (e.g. 5 years or 10 years) or some other time? Our method will not only find the best point of discrimination with respect to the level of a marker but also the time when the endpoint of interest is observed. Determining such a point could be useful in deciding not only *what value* of a marker is optimal for identifying homogeneous subsets of patients but also at *what point in time* should the decision be made. In addition, simulations are used to assess the statistical properties of this proposed methodology.

Survival analysis has become a well-accepted statistical tool in medical research, starting with the developments by Kaplan and Meier (1958) and later by Cox (1972). Survival analysis is used to analyze the time until the occurrence of a well-defined "event". Examples of events are relapse of some disease, the failure of a component of an electrical system, the failure of an individual to take a medication and the death of an individual. A common feature of most survival outcomes is the presence of censoring and truncated observations, that is, observations for which the event of interest has not (yet) been observed. The Kaplan-Meier method provides an estimate of the cumulative survival distribution at time *t*. The Kaplan-Meier survival estimates are typically summarized using a step-function that changes at every distinct survival time.

Since its introduction by Cox in 1972, the proportional hazards model has become the most popular technique for relating a number of covariates to the survival time of a cohort of patients. It doesn't directly model the survival time but, rather, a related quantity known as the hazard function denoted as λ. The hazard function is defined as

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{\Pr(t \leq T < t + \Delta t | T \geq t)}{\Delta t} = \frac{f(t)}{S(t)}$$

where *f(t)* is the instantaneous failure distribution at time *t* and *S(t)* is the cumulative survival distribution at time *t*. The Cox model is represented as:

$$\lambda_i(t) = \lambda_0(t) \exp\{\beta_1 x_{i1} + \ldots\ldots + \beta_p x_{ip}\}$$

where $\lambda_i(t)$ is the hazard function for the *i*th subject, $\lambda_0(t)$ is the baseline hazard function, and β_1 , …, β_p are parameters to be estimated. The Cox model makes no assumptions for the baseline survival distribution but does assume a parametric form for the effect of the predictors on the hazard. Thus, a Cox model is known as a "semi-parametric" procedure. An important aspect of Cox's paper is that the hazard function is the basis of the regression model. The Cox model is

flexible enough to handle cases of time-varying coefficients and time-varying covariates. The baseline hazard is estimated independently of the regression coefficients.

Tree structured methods were originated by social scientists. The use of trees in regression dates back to the Automatic Interaction Detection (AID) program developed by Morgan and Sonquist (1963) as a sequential procedure for the analysis of survey data. It was intended to overcome the problem of interaction between variables used for classification. Their method was based on a series of decision rules and instructions that group observations into homogeneous subsets. In their approach, the sample is initially conceptualized as a single group. A decision is made to find the division of the parent group that provides the largest reduction in the explained sum of squares. These groups are then investigated further to see if they can be divided using the same criterion from the parent group. The process continues until a further subdivision will not reduce the unexplained sum of squares by at least one percent of the total original sum of squares. At this point the subgroup will no longer be investigated. Morgan and Messenger (1973) developed a classification trees program (THAID) that performed multi-level splits when computing classification trees. Kass (1980) extended the methodology of AID to categorical data called CHAID. The predominant statistic used for splits in this procedure is the Chi-square statistic. The classic "Classification and Regression Trees" (CART) algorithm was introduced by Breiman, Friedman, Olshen, and Stone (1984) and is similar to AID in that it uses binary splits to achieve the final classification. However, the splitting mechanism used in CART is very different from that of AID or CHAID. CART uses the Gini-index to measure the homogeneity of cases at a leaf node. As mentioned previously, AID uses the unexplained sum of squares while CHAID uses a Chi-square measure to evaluate splits based on the significance of differences in response distributions between groups.

Tree-based methods, due to their ability to discriminate sub-groups in a hierarchical manner can sometimes provide a useful alternative to the classical linear proportional hazards model of Cox (1972) for the exploration of survival data. Since the data are partitioned into groups that are assumed to be approximately homogeneous, Kaplan-Meier estimators can be used to compare prognosis between the groups presented by "nodes" in the tree. A fruitful application area for tree-based methods is in clinical studies where investigators are interested in grouping patients with differing prognoses.

2.0 THE CART METHOD

A "binary" tree represents the CART algorithm. In the binary tree, each group or subgroup within the scheme can potentially be further subdivided into two groups. The CART methodology consists of three parts. First, a regression tree is grown which overfits the data. Next, branches are pruned off from the overfitted tree. Then, a final tree is selected which represents the best estimate of the regression function for the data. Breiman, et al. used the following notation and terminology to describe the CART methodology. This notation will be carried throughout this paper, as the CART method will be extended to survival regression trees.

2.1 TERMINOLOGY

We start with a collection of M measurements (e.g. age, tumor status, etc.) which is referred to as a measurement vector, denoted by $\boldsymbol{x} = (x_1, x_2, \ldots, x_M)$. A measurement space if defined as the set of all possible measurement vectors is denoted by $\boldsymbol{X}$. Suppose that the subjects can be partitioned into J classes. The set of all classes will be denoted by $C = \{1, 2, \ldots, J\}$. Thus, C contains J disjoint and exhaustive classes will be uniquely assigned to a single class in C for each possible measurement vector $\boldsymbol{x} \in \boldsymbol{X}$. A classifier is a function $d(\boldsymbol{x})$ defined on $\boldsymbol{X}$ that specifies for every $\boldsymbol{x}$, $d(\boldsymbol{x})$ is equal to a particular class $(1, 2, \ldots, J)$. Thus, $d(\boldsymbol{x}) \in \{1, 2, \ldots, J\} \; \forall \; \boldsymbol{x} \in \boldsymbol{X}$. A classifier is constructed based on past data, referred to as a learning sample. The learning sample, L, consists of measurements on N subjects along with their actual classification.

Thus, the learning sample is denoted by

$$L = \{(\mathbf{x}_1, j_1), \ldots, (\mathbf{x}_N, j_N)\}.$$

Two questions arise: does this classifier express the "truth" and how accurate is the estimate? Breiman, et al. were interested in three types of estimation methods to determine the accuracy. These methods are resubstitution estimation, test sample estimation, and cross-validation. A fourth method, called bootstrap estimation, can be used to estimate $R^*(d)$, the misclassification rate, but Brieman, et al. found that the bootstrap estimate might not work well when applied to tree structure classifiers.

The first method, which is the least accurate, utilizes "resubstitution estimation". Let us consider estimating the misclassification rate that we would expect for a classifier $d(x)$. We could use a resubstitution estimate, which is the easiest and most commonly used. If we constructed a classifier $d(x)$ using all of the subjects in the learning sample, the resubstitution estimate would simply be the proportion of that are misclassified by $d(x)$ cases in our sample. If we define an indicator function $Y(\cdot)$ as one if the statement inside the parentheses is true and zero if it is not true, we would then calculate the misclassification rate as

$$R(d) = \frac{1}{N}\sum_{n=1}^{N} X(d(x_n) \neq j_n) .$$

However, the problem with the resubstitution estimate is that it tends to underestimate the true misclassification rate of $d(x)$ when used with new samples. The reason for the underestimation is because the resubstitution estimate and the $d(x)$ were constructed from the same data and $d(x)$ was formed to minimize the proportion of misclassifications in the learning sample.

The test sample method randomly allocates the learning sample L into two parts L_1 and L_2. Most commonly, L_1 consists of 2/3 of the data in the learning sample and L_2 consists of the

other 1/3. *d(x)* is constructed by using L_1 and the misclassification rate is estimated by finding the proportion of cases misclassified by *d(x)* in L_2. The drawback in using this method is that despite reducing the bias found in the resubstitution estimates, a portion of the data is lost for the construction of *d(x)*. This drawback is offset somewhat if large datasets are used since a more accurate estimate is being obtained.

The last method, cross-validation, uses the entire sample to construct *d(x)*. This method works by partitioning the data into equal-sized subsets and holding out one subgroup at a time to construct *d(x)*. Cross-validation will be further discussed in Section 2.6.

2.2 GROWING A TREE

Tree structured classifiers are graphically represented by a binary tree, denoted by *T* (see Figure 2.1). Tree structured classifiers are constructed by splitting the dataset into two subsets. All of the observations in a dataset start in a "root node." That way, every possible unique split of the form "$x < x_0$" (for a continuous x) or "$x \in$ subset i" (for a categorical x) is examined. The goal is to select each split of a subtree so that the data in each of the descendant subtrees are "purer" than the data in the parent subtrees. These splits are generated in the following fashion. Starting with the first variable, x_1, CART splits a variable at all of its possible split points. At each possible split point of the variable, the sample splits into two binary or child nodes. Observations with a "yes" response to the question posed are sent to the left node t_L and the "no" responses are sent to the right node, t_R. It is also possible to define these splits based on linear

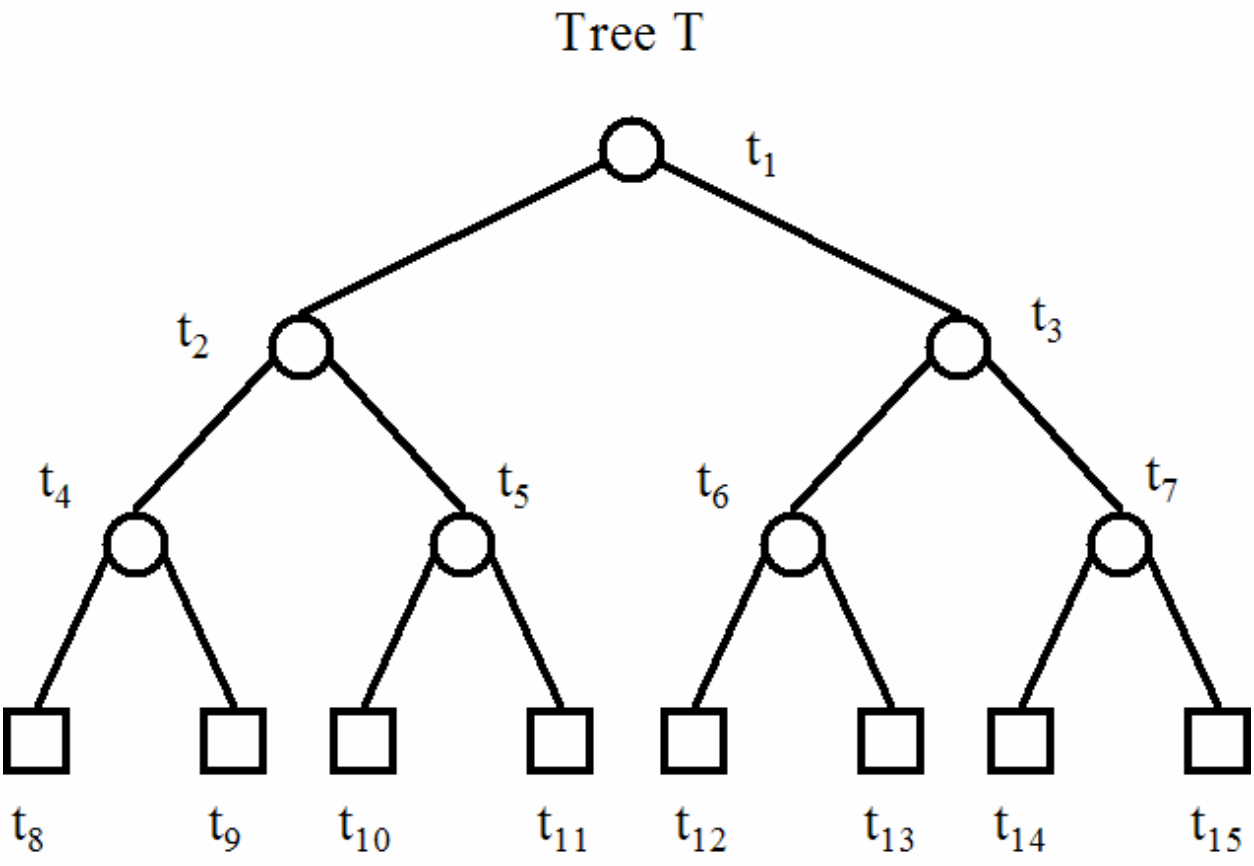


Figure 2.1: Basic Tree

combinations of variables. CART then applies its goodness of split criteria to each split point and evaluates the reduction in impurity, or heterogeneity due to the split. For tree *T*, the data in the "root node" (t_1) is first split into two subsets, t_2 and t_3. These subsets are formed by using with the intention of satisfying some criteria for the class membership of the measurement vectors contained therein. A number of different measures of purity can be selected but the most common splitting criteria is the "Gini", followed by "twoing." Breiman, et al. concluded that within a wide range of splitting criteria the properties of the final tree selected are surprisingly insensitive to the choice of splitting rule. The criterion used to prune or recombine upward is much more important. The subsets are split into smaller subsets at the largest decrease of the

node impurity (which will be described in detail later). The node impurity is the largest when all of the classes are contained with a node and is the smallest when the node contains only one class. This process is repeated for each of the remaining variables at the root node. CART ranks all of the best splits on each variable according to the reduction in impurity achieved by each split. It selects the variable and its split point that most reduced impurity of the root or parent node. CART then assigns classes to these nodes according to a rule that minimizes misclassification costs. Each subset is then considered for an additional binary split so that t_2 could be split into t_4 and t_5 and t_3 could be split into t_6 and t_7. This process is repeated for each subset until the process can no longer be continued. The goal is to have subsets of $\boldsymbol{X}$ that are more "pure", in that the majority of the measurement vectors in each subset belong to the same class.

If the subset can be further split into two more subsets to achieve a more accurate classification tree then the subset is referred to as a nonterminal node, denoted with a circle (see Figure 2.1, node t_2). If the subset does not result in a significant decrease in the node impurity and does not need to be split further then the subset is referred to as a terminal node, denoted with a square (see Figure 2.1, node t_8). When a terminal node, say t_8, has been reached then all of the measurement vectors that belong to this node, $\{\boldsymbol{x} : \boldsymbol{x} \in t_8\}$, are then assigned to the same class. The construction of a tree consists of three elements: selection of the splits, deciding when to declare a node terminal or to continue the splitting process, and assigning a class to each terminal node.

2.3 CLASS ASSIGNMENT

Of the three elements of tree construction, the assignment of classes to terminal nodes is the easiest to perform. Let N denote the total number of observations and N_j denote the number of class j observations. Let $\pi(j)$ be the prior probability of a class j observation. These prior probabilities are estimated from data by using N_j/N or are provided by analysts. Let $N(t)$ be the total number of cases from L with $x_n \in t$, and $N_j(t)$ be the number of class j cases with $x \in t$. Thus, $N_j(t)/N_j$ is an estimate for the probability that a class j observation will fall into node t. Then, the resubstitution estimate for the probability that a case will be class j and fall into node t will be $p(j, t) = \pi(j)*N_j(t)/N_j$. The probability that any case will fall into node t is presented as $p(t) = \sum_j p(j,t)$. The conditional probability that an observation is class j given that it falls into node t is $p(j|t) = p(j, t)/p(t)$. When N_j/N are used as the prior probability estimates, $p(j|t)$ reduces to $N_j(t)/N(t)$.

Now, a classification assignment rule needs to be developed. Suppose that we have constructed a tree T having a set of terminal nodes $\tilde{T}$. A rule is developed that assigns a class $j \in \{1, \ldots, J\}$ to each terminal node $t \in \tilde{T}$. Let $j(t)$ denote the class assigned to a node t. The joint probability of a case being from class j and falling into node t, $p(i|t)$, is estimated from the data as $p(j,t) = \pi(j)N_j(t)/N(t)$. By extension, the resubstitution estimate for the probability of any case falling into node t, $p(t)$ is,

$$p(t) = \sum_j \frac{\pi_j N_j(t)}{N_j}$$

The resubstitution estimate for the probability of misclassification given that an observation falls into node t is given by,

$$p(j|t) = \frac{p(j,t)}{p(t)} with \sum_j p(j|t) = 1.$$

When the $\pi(j)$ are estimated from the data using $N_j(t)/N(t)$, $p(j|t)$ can be estimated by,

$$p(j|t) = \frac{N_j(t)}{\sum_j N_j(t)} = \frac{N_j(t)}{N(t)}.$$

The class assignment rule, which minimizes this misclassification estimate, is a natural choice for $j^*(t)$. If $p(j|t) = \max_i p(i|t)$, then $j^*(t)=j$. If there are two or more classes which achieve the maximum, then $j^*(t)$ is arbitrarily assigned as one of the maximizing classes.

Using this class assignment rule, the resubstitution estimate $r(t)$ for misclassification in node t reduces to $r(t) = 1 - \max_i p(i|t)$. If we denote the joint probability $R(t)$ that a case falls into node t and is misclassified as $R(t) = r(t)p(t)$, then the resubstitution estimate for the overall misclassification rate $R^*(T)$ of the tree T is

$$R(T) = \sum_{t \in \tilde{T}} R(t).$$

Since misclassifying a case might be worse in some situations, the idea of including a set of misclassification costs was introduced. These misclassification costs refer to the penalty that one assigns to the different possible misclassifications. For example, when trying to classify patients into whether they are at high risk or low risk for a certain type of cancer, we may feel that there is little penalty for identifying someone as high risk when in fact they are low risk. However, if a patient is classified as low risk when they are really high risk, the repercussion of this misclassification is much worse. In such cases, we can construct a cost function $C(i|j)$, where:

$$(i)\ C(i|j) \geq 0,\ i \neq j,$$

$$(ii)\ C(i|j) = 0,\ i = j.$$

This function reflects the penalty associated with the possible types of misclassification. To form our class assignment rule *j*(t)*, we want to select *i* to minimize the estimated expected misclassification cost $\sum_{j=1}^{J} C(i|j)p(j|t)$. Thus, the resubstitution estimate of the expected misclassification cost for a node *t* is

$$r(t) = \min_i \sum_{j=1}^{J} C(i|j)p(j|t),$$

and the resubstitution estimate of the misclassification cost of the tree *T* is

$$R(T) = \sum_{t\in\tilde{T}} r(t)p(t) = \sum_{t\in\tilde{T}} R(t)$$

where *R(t) = r(t)p(t)*.

2.4 SPLITTING

The first step in building a tree is choosing the splits of the measurement space ***X***. The general idea is to split ***X*** into subsets that are increasingly "pure". That is, we want each terminal node to have the majority of the subjects within the node to belong to the same class. We need a measure of this impurity to facilitate a splitting rule to be used in building our tree. In developing a methodology to evaluate and compare potential splits, Breiman, et al. (1984) developed the goodness of fit criterion, which was derived from the impurity function:

Definition 2.4.1: An impurity function Φ is defined on the set of all J-tuples ($p_1, p_2, \ldots, p_J$) such that $p_j \geq 0$, $\sum_j p_j = 1$, and

(i) Φ attains a single maximum at the point $(1/j, 1/j, \ldots, 1/j)$

(ii) Φ attains a minimum at the points $(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$

(iii) Φ is a symmetric function of $p_1, \ldots, p_J$.

Given an impurity function and the conditional probabilities for the J classes at any node t, an impurity measure $i(t)$ can be defined as

$$i(t) = \Phi(p(1|t), p(2|t), \ldots, p(J|t)).$$

A candidate split s will be selected based on its reduction of the impurity in the node t. Let the node t be split into t_L and t_R by s. Then p_L and p_R proportions of the cases in t will be placed into nodes t_L and t_R, respectively. Then our measure of the decrease in impurity in node t due to split s is simply,

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R).$$

For each node t, we choose the split s that maximizes $\Delta i(s, t)$. Once a node is split, the children nodes are evaluated to determine if they can be split. This process is repeated until every node contains a small number of subjects.

A stopping rule is a criterion for determining a terminal node. An early stopping rule was to set a threshold β. Then, a node t is declared a terminal node if our split that maximizes $\Delta i(s,t) < \beta$. Another stop-splitting rule is to declare a terminal node if the number of cases assigned to the node is less than some value. Measurement vectors in a terminal node are typically assigned to the class with the largest conditional probability, $p(j|t)$. Note that if the class priors are determined from the training data, i.e., N_J/N, this rule assigns a terminal node to the class with the largest number of measurement vectors falling into the node.

2.5 COST-COMPLEXITY PRUNING

The concept of the stopping rule was plagued with problems. First, the threshold β could be set too low or good splits farther down the tree might never be reached. These problems lead to the development of pruning. The pruning process begins with a tree that is split until every node contains a small number of cases, forming the tree T_{max}. Then children nodes are selectively recombined (pruned) into the single parent upward toward the root node, creating more general trees.

It is now necessary to clearly define some basic terminology concerning trees. We will start with six definitions.

Definition 2.5.1: T will be referred to as a tree and each element of T will be referred to as a node.

Definition 2.5.2: The minimum element of a tree T is called the root of T, denoted by root (T). If s, $t \in T$ and t = left(s) or t = right(s), then s is called the parent of t. The root of T has no parent, but every other node has a unique parent.

Definition 2.5.3: A node t is called a terminal node if it is not a parent, that is, if left(t)=right(t)=0. Let $\widetilde{T}$ denote the collection of terminal nodes of T. The elements in T- $\widetilde{T}$ are called non-terminal nodes.

Definition 2.5.4: A branch T_t of T with root node $t \in T$ consists of the node t and all descendants of t in T.

Definition 2.5.5: Pruning a branch T_t from a tree T consists of deleting from T all descendants of t, that is, cutting off all of T_t except its root node. The tree pruned this way will be denoted by T- T_t.

Definition 2.5.6: If T' is derived from T by successively pruning off branches, then T' is called a pruned subtree of T and denoted by $T' \prec T$

Using these definitions, a node t is called a descendant of node s if there exists a connected path down the tree from s to t. This also implies that s is an ancestor of t. In Figure 2.2 we can see that t_8 is a descendant of t_1 but not of t_3. T_t is called a branch of T if its root node $t \in T$ and T_t contains all of the descendants of t. Figure 2.2 shows the branch T_{t2}. Pruning a branch T_t from T is defined as removing all of the descendants of the node t from T but retaining the node t itself. The resulting pruned tree will be denoted by $T - T_t$. Once again in Figure 2.2 we see the branch T_{t2} has been pruned from T leaving us $T - T_{t2}$. If a tree T' has been obtained from a tree T through pruning alone, then we use the notation $T' \prec T$ to denote that T' is a pruned subtree of T. A pruned subtree will always contain the same root node as the tree from which it was derived.

In Figure 2.2, $T - T_{t2} \prec T$. The notation $T' \preceq T$ will be defined as T' is either a pruned subtree of T or T' is exactly the same tree as T itself. Now that the methodology for building trees has been explained and the terminology has been established, it is time to approach selection of the "best" tree. The best tree is one that is small, easily interpretable but still retains its ability to correctly classify.

Minimal cost-complexity pruning works in the following way. The pruning process starts with T_{max}. The resubstitution estimate $R(t)$ is computed for each node $t \in T_{max}$. T_{max} is

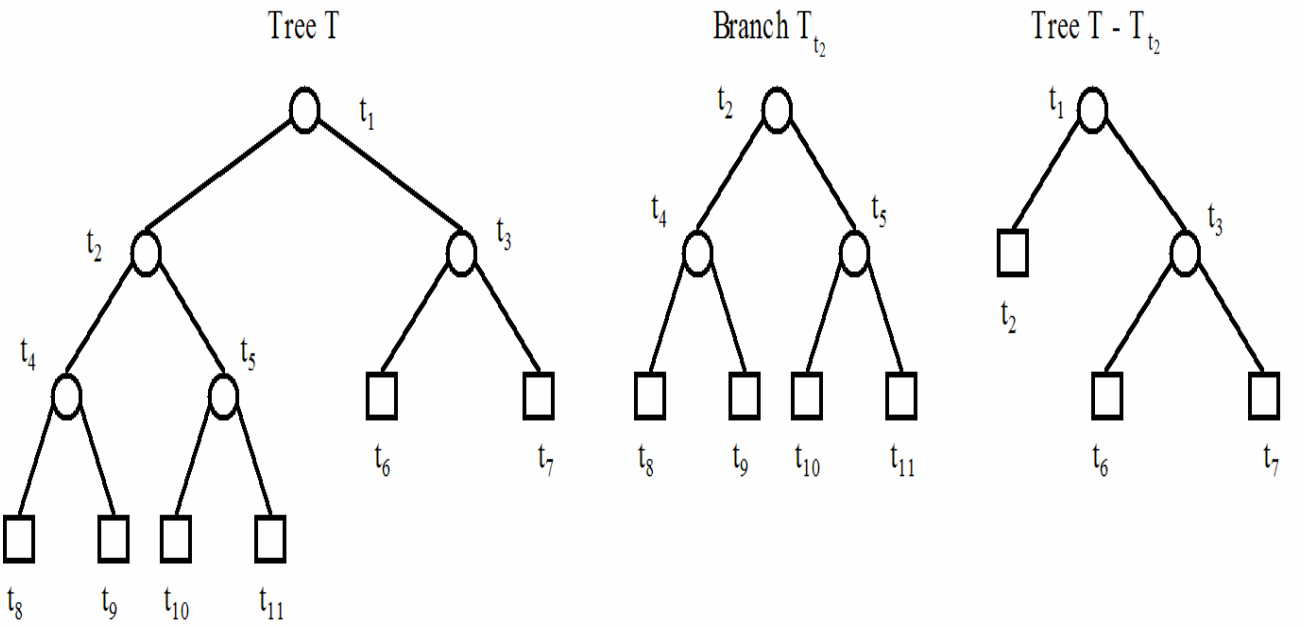


Figure 2.2: Pruning a Tree

progressively pruned upwards to its root node where *R(T)* is as small as possible for each stage of the pruning process. For any tree $T \preceq T_{max}$, define its complexity $\|\tilde{T}\|$ as its number of terminal nodes. Let cost-complexity parameter $\alpha \geq 0$ and define the cost-complexity measure $R_\alpha(T)$ as:

$$R_\alpha(T) = R(T) + \alpha \|\tilde{T}\|.$$

The cost-complexity measure adds a penalty for tree complexity to the overall resubstitution error. For each value of α, the objective is to find the subtree $T(\alpha) \preceq T_{max}$ where $R_\alpha(T)$ is minimized,

$$R_\alpha(T(\alpha)) = \min_{T \prec T_{\max}} R_\alpha(T).$$

When α is small, the penalty for tree complexity is also small and the minimizing subtree $T(\alpha)$ is large. As α increases, the minimizing subtree will have fewer terminal nodes. At some

sufficiently large value of α, $T(\alpha)$ consists of only the root node $\{t_1\}$ and the pruning process is complete. Note that while α is continuous, there are obviously only a finite number of subtrees of T_{max}. Thus, a finite sequence of subtrees $T_1, T_2, \ldots, \{t_1\}$ will be created by the pruning process. Given this finiteness, if $T(\alpha)$ is the minimizing subtree for some value of α, then $T(\alpha)$ will continue to be the minimizing tree until some breakpoint α' is reached. At α' a new tree $T(\alpha')$ will become the minimizing tree until the next breakpoint α'' is reached, etc.

Breiman, et al. was able to solve two critical questions. The first solution was for any value of $\alpha \geq 0$, there exists a unique, smallest minimizing subtree as defined by the following definition.

Definition 2.5.7: The smallest minimizing subtree $T(\alpha)$ for complexity parameter α is defined by the conditions:

i) $R_\alpha(T(\alpha)) = \min_{T \prec T_{\max}} R_\alpha(T)$

ii) If $R_\alpha(T) = R_\alpha(T(\alpha))$, then $T(\alpha) \preceq T$

Thus, there can be no ties in minimizing $R_\alpha(T)$ for any value of α since if two trees have the same value for $R_\alpha(T)$, they must differ in size and the smaller of the two is chosen as the minimizer. The second solution stated, given a tree T_{max} with root node $\{t_1\}$ and increasing values of $\alpha \geq 0$, the finite sequence of minimizing subtrees $T_1, T_2, \ldots, T_k, \{t_1\}$ is a nested sequence such that, $T_1 \succ T_2 \succ \ldots \succ \{t1\}$. These solutions indicate it is not necessary to search all possible subtrees to find the unique minimizer of $R_\alpha(T)$. Instead, the sequence of minimizing subtrees can be found by sequentially pruning a classification tree upward toward the root node.

The minimal cost-complexity pruning algorithm starts with T_1, the smallest tree of T_{max} where $R(T_1) = R(T_{Max})$. This subtree is obtained by pruning every ancestor node t such that

$R(t)=R(t_L) + R(t_R)$. This means that T_1 performs as well as T_{max} with only the splits removed that did not improve the tree's performance. Let the cost associated with a branch T_t of T_1 be

$$R(T_t) = \sum_{s \in \tilde{T}_t} R(s),$$

which is just the sum over the costs of the terminal nodes in branch T_t. For any nonterminal node t in T_1, we have that $R(t) > R(T_t)$. Thus, the cost associated with a node t is strictly greater than the cost associated with the branch T_t, which means the cost of a tree, will always increase as it is pruned. Let $\{t\}$ denote the subbranch of T_t that contains only one node t and the cost-complexity measure be

$$R_\alpha(\{t\}) = R(t) + \alpha,$$

and for any branch T_t define

$$R_\alpha(T_t) = R(T_t) + \alpha \left\| \tilde{T}_t \right\|.$$

At some the value of α, the cost-complexity of branch T_t is equal to that of just the node t. Setting $R_\alpha(\{t\})$ equal to $R_\alpha(T_t)$ and solving for α yields

$$\alpha = \frac{R(t) - R(T_t)}{\left\| \tilde{T}_t \right\| - 1},$$

which is positive due to the relationship $R(t) > R(T_t)$. Now that we have a method for finding the critical value of α for each nonterminal node in T_1, we can now define how we will choose the branch to be pruned. Define $g_1(t)$ for $t \in T_1$, by

$$g_1(t) = \begin{cases} \dfrac{R(t) - R(T_t)}{\left\| \tilde{T}_t \right\| - 1}, t \notin \tilde{T}_1 \\ +\infty, t \in \tilde{T}_1 \end{cases}.$$

Then the weakest link, $\bar{t}_1$, in T_1 is the node satisfying

$$g_1(\bar{t}_1) = \min_{t \in T_1} g_1(t).$$

Now set $\alpha_2 = g_1(\bar{t}_1)$ and prune the branch $T_{\bar{t}_1}$ to obtain T_2. Repeat this process so that the weakest link is found on tree T_2 to obtain T_3 when α reaches α_3. This process is repeated until only the root node $\{t1\}$ remains. Now $\{a_k\}$ are an increasing sequence, $\alpha_1 < \alpha_2 < \ldots < \alpha_k$, of critical values where k is the number of trees in the nested sequence of cost-complexity pruned trees, $T_1 \succ T_2 \succ \ldots \succ \{t1\}$.

2.6 SELECTING THE BEST TREE

Now that we have our decreasing sequence of trees, $T_1 \succ T_2 \succ \ldots \succ \{t1\}$, we now need to determine which is the "best" tree. Each tree in the sequence is best for some range of the complexity parameter α in that it minimizes the cost-complexity function. The method that we will focus on to determine the best tree is cross-validation. Cross-validation (briefly discussed earlier) involves randomly dividing the data up into V roughly equal groups. One of the V portions is left out while the remaining portions are all used to build a model. The portion not used in building the model is used to assess the accuracy of the current model. This process is repeated for each of the other $V-1$ portions and then the V estimates are averaged to get the final cross-validation estimate for model accuracy. For example, a very popular type of cross-validation is the 10-fold cross-validation process in which subtrees of different sizes are constructed with 90% of the data set and their misclassification rates on the remaining are computed. This process is done ten times with each 1/10 of the data held out one at a time. Then, the misclassification rates are aggregated over the replications. The optimal tree size is the one whose aggregated misclassification rate is smallest.

The cross-validation method can be implemented to the complex sequence of trees $T_1 \succ T_2 \succ \ldots \succ \{t1\}$ in the following way. Randomly allocate each subject in the learning sample L into V (nearly) equal sets. Let L_v denote the vth portion of the learning sample and $L^{(v)}$ represent the entire learning sample missing only the vth portion. We already have our sequence of trees and critical values of α based on the entire learning sample. Now we must repeat the tree growing and cost-complexity pruning steps for $L^{(1)}, L^{(2)}, \ldots, L^{(v)}$. For $L^{(1)}$, we will obtain a sequence of trees $T_1^{(1)} \succ T_2^{(1)} \succ \ldots \succ \{t_1\}$ and a sequence of critical values $\alpha_1^{(1)} < \alpha_2^{(1)} < \ldots < \alpha_{k1}^{(1)}$ where k_1 is the number of subtrees in the previous sequence. A similar sequence of trees and critical values will be obtained for the other $V - 1$ sets.

Let $T^{(1)}(\alpha'_j)$ be the minimal cost-complexity tree for complexity parameter α'_j based on $L^{(1)}$. The minimal cost-complexity tree can be found for the other $V - 1$ sets giving us $T^{(1)}(\alpha'_j)$, $T^{(2)}(\alpha'_j), \ldots, T^{(V)}(\alpha'_j)$. Now define

$$N_{ij}^{(v)} = \text{the number of class } j \text{ cases in } L_v \text{ classified as } i \text{ by } T^{(V)}(\alpha'_j),$$

and

$$N_{ij} = \sum_{v=1}^{V} N_{ij}^{(v)} .$$

Since each case in L appears in one and only one test sample L_v, N_j is the total number of class j observations in L. The cross-validation estimate for the probability of classifying a case as i given that it is j can be given by $Q^{CV}(i|j) = N_{ij} / N_j$. Then the cross-validation estimate for the cost associated with class j is given by

$$R^{CV}(j) = \sum_{i=1}^{J} C(i|j) Q^{CV}(i|j)$$

and the cross-validation estimate for the cost of T(α) is given by

$$R^{CV}(T(\alpha)) = \sum_{i=1}^{J} R^{CV}(j)\pi(j).$$

If the data from *L* is used to estimate the priors, then the cross-validation estimate reduces to

$$R^{CV}(T(\alpha)) = \frac{1}{N}\sum_{i,j} C(i|j)N_{ij},$$

In the cases were the unit cost is incorporated, the cross-validation estimates is simply with the proportion of the data misclassified. Selection of the right sized tree is now obtained by finding T(α'_{j0})) such that

$$R^{CV}(T(\alpha'_{j0})) = \min_{j=1}^{k} R^{CV}(T(\alpha'_j)).$$

R^{CV}(T(α'_{j0})) can be used as an estimate of the misclassification cost.

Breiman, et al. state that taking *V*=10 gives adequate accuracy. In some examples, smaller values of *V* also give sufficient accuracy. However, they did not come across any situations where taking *V* larger than 10 gave a significant improvement in accuracy for the tree selected.

2.7 MISSING DATA

An important feature of regression trees is the mechanism to deal with missing predictor values. The easiest approach is to treat the missing attribute as a distinct value and to assign all samples with missing values to the same node (Zhang, et al. 1996). Breiman, et al. (1984) introduced surrogate splits to deal with missing attributes by calculating to what extend alternative splits

resemble the best split in terms of the number of cases that they send the one way. If an observation is missing a value for the best split then it is classified using the first surrogate split. If that value is missing then the second surrogate split is used, and so on. If an observation is missing all the surrogate splits then the default rule *max* $(p_L;p_R)$, where the observation is sent to the child with the largest relative frequency at that node, is used. The surrogate splits are advantageous because they use other available information to make the split. Breiman, et al. (1984) also proposed to rank the importance of variables through surrogate splits.

2.8 REGRESSION TREES

We now shift our focus to regression trees. Many of the concepts transfer over from classification trees to regression trees. However, several things become simpler because there are no priors so each case is weighted equally. The predictor *d(x)* is now a function defined as a real-value function on $\boldsymbol{X}$. Thus *d(x)* produces real numbers not classes. We have a learning sample *L* consisting of the observations $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ where $\boldsymbol{x}$ is a vector of measurements that hopefully explain the real valued response *y*. The misclassification rate of the classifier *d(x)* is denoted by *R(d)*. In the regression setting, this misclassification rate *R(d)* will now be represented by the estimated mean squared error for *d(x)* to measure the accuracy of the predictor. Given a rule *d(x)*, our resubstitution estimate for the mean squared error is

$$R(d) = \frac{1}{N}\sum_{n} (y_n - d(x_n))^2 .$$

The average of y_n for all cases that fall into node *t*, denoted by $\bar{y}(t)$ is used to minimize *R(d)*.

Thus, our estimate for the mean squared error of the tree T is

$$R(T) = \frac{1}{N}\sum_{t\in\tilde{T}}\sum_{n:x_n\in t}(y_n - \bar{y}(t))^2 ,$$

if we let $R(t)$ be the within node sum of squares divided by N, then $R(T)$ can be written as

$$R(T) = \sum_{t\in\tilde{T}} R(T) .$$

Determining what the best split of a current terminal node t is simply done by finding the split of t into t_L and t_R which decreases $R(T)$ the most. Let s be a candidate split in the set of possible splits S. Let

$$\Delta R(s, t) = R(t) - R(t_L) - R(t_R)$$

and find the best split s^* which satisfies

$$s^* = \arg\max_{s\in S} \Delta R(s,t) .$$

The same method to grow a tree with classification trees is used for regression trees. T_{max} is created by splitting to minimize $R(T)$. The tree is declared T_{max} once each terminal node contains at most some small number of observations (usually 5). Minimal error-complexity pruning is performed the same way as minimal cost-complexity in classification trees. Define the error-complexity measure as

$$R_\alpha(T) = R(T) + \alpha \left\|\tilde{T}\right\|.$$

We start by obtaining our sequence of trees and critical values based on the entire learning sample. Let $d_k(x)$ be the prediction rule associated with the tree T_k. We randomly divide L into V groups and find the sequence of trees and critical values for each of the V possible groups formed by leaving out one of the V portions. For each of the V sequence of trees and critical values, we can form the function $T^{(v)}(\alpha)$ that is the minimal error-complexity tree for parameter α in the vth

sequence of trees. The cross-validation estimate for the mean squared error of tree T_k is

$$R^{CV}(T_k) = \frac{1}{N}\sum_{v=1}^{V}\sum_{n:(x_n,y_n)\in L_v}(y_n - d_k^{(v)}(x_n))^2 .$$

which leads to the cross-validation estimate,

$$RE^{CV}(T_k) = R^{CV}(T_k) / R(\bar{y}) .$$

3.0 EXTENSIONS TO THE CART METHOD

One of the most popular uses for tree-based methods is in survival analysis for censored time data where the goal is to identify factors that are predictive of survival. Several extensions of the CART method (Breiman, et al. 1984) have been proposed for censored survival data. Tree structured models for survival data can roughly be divided into two categories. The first category contains methods that use a between-node homogeneity measure to identify distinct nodes. The split function is based on the two-sample log-rank test statistic. The second category contains methods that use a within node homogeneity measure (such as deviance) to identify distinct nodes. Examples of such approaches are provided in Gordon and Olshen (1985), Davis and Anderson (1989), and LeBlanc and Crowley (1992). We will go into more detail with these methods since this is the approach used in our proposed method.

3.1 METHODS THAT USE BETWEEN-NODE HOMOGENEITY

For the first category of tree structure models, alternatives to the CART pruning algorithm and cross-validation are used. The log-rank test is a popular approach for testing the significance of differences between the survival times of two groups. Motivated by this, Ciampi, et al. (1986) and Segal (1988) suggested selecting a split that results in the largest log-rank test statistic. Segal (1988) presented a nonparametric application using the Harrington-Fleming (1982) class of two-sample rank statistics that based the partitioning on between-node separation

instead of within-node homogeneity. Segal (1988) recommended a practical bottom-up procedure. After growing a large tree, they started from the bottom and stepped upwards through the tree assigning each internal node a value that equals the maximum of the log-rank statistics over all splits starting from the internal node of interest. All of the nodes daughters are pruned off if an internal node corresponds to a smaller value than the threshold. Ciampi, et al. (1986) introduced the term "amalgamation". Since even an optimally pruned tree may have many terminal nodes, amalgamation is the process of combining terminal nodes that have similar survival into one group. Ciampi, et al. used the log-rank statistic for combining the terminal nodes.

LeBlanc and Crowley (1993) developed a recursive partitioning method based on maximizing the difference in survival between groups of individuals represented by nodes in a binary tree. They introduced the notion of "goodness of split" complexity as a substitute for cost complexity in pruning the tree. This "goodness of split" complexity is based on two-sample statistics and the trade-off between the overall structure found and tree size. They defined an ordered categorical variable describing the terminal nodes, and apply the recursive partitioning scheme to that single variable to amalgamate the nodes.

Bacchetti and Segal (1995) further extended the tree-structured method by Segal (1988) to allow for right-censoring, left-truncation and time-dependent covariates in survival trees. Huang, Chen and Soong (1998) proposed a method, similar to Bacchetti and Segal, to accommodate the time-dependent covariates in survival trees.

As an alternative to Davis and Anderson (1989), Huang, et al. incorporated time into the model as an argument of the hazard function. Time is then treated as a time-dependent covariate in the recursive partitioning algorithm. Their method splits nodes through the interaction of the

covariate values and time and establishes measures of improvement on the basis of piecewise exponential survival functions. The estimated hazard function at each node summarizes the risk of a group of subjects during each specific time period. Both cross-validation and bootstrap resampling techniques are implemented in the tree selection procedure.

Ahn and Loh (1994) developed a tree-structured model that stratifies data according to selected covariate values and fits separate proportional hazards models to each stratum. They performed the stratification recursively by using a combination of statistical tests and residual analysis. They used two methods (called the "M" and "R" methods) earlier proposed by Loh (1991) for classifying the data into two classes. The "M" method classifies the covariate vectors into two classes according to the size of their associated residuals. The "R" method forms the two classes by using the relative positions of the residuals above and below a least-squares line superimposed on a cumulative hazard plot. They used bootstrapping as a way to control the probability of a Type I error.

Chaudhuri, et al. (1994) introduced a new method of tree-structured regression called SUPPORT (Smoothed and Unsmoothed Piecewise-Polynomial Regression Trees). The method used polynomial models to fit each subset and weighted averaging to combine the piecewise-polynomial regression fits into a smooth one. Chaudhuri, et al. (1995) later used a tree-structured method that recursively partitioned the data according to the signs of the residual from a model fitted by maximum likelihood to each node. They used a split selection strategy based off of the methods of their earlier work for tree-structured least squares regression and Ahn and Loh (1994) for tree-structured proportional hazards regression.

Loh and Vanichsetakul (1988) developed an algorithm (FACT) combining CART and Linear Discriminant Analysis (LDA). FACT differs from CART in that it uses a different

misclassification cost based on discriminant functions. Loh and Shih (1997) extended the method (FACT) by Loh and Vanichsetakul. Loh and Shih's algorithm (QUEST) shares similarities with FACT but has negligible variable selection bias.

Segal (1992) extended the tree-structured method of Breiman, et al. (1984) to repeated measures and longitudinal data. Alexander and Grimshaw (1996) proposed a "treed" regression that used the best simple linear regression models at each leaf. Treed regression models possess many of the desirable qualities of linear regression (no tree structure but a complex linear model at each leaf) and CART (tree structure but a simple linear model at each leaf).

Zhang (1998) generalized the tree-based methodology for classification of multiple binary responses. Ciampi, et al. (2002) proposed a new algorithm for the construction of a tree-structured predictor that used a new approach for dealing with continuous predictors. Their approach was based on using soft nodes where an individual would go right with a certain probability or left with the complimentary probability. Pettitt and Daud (1990) proposed a Cox proportional hazards model which considered time-dependent modulation of the linear predictor. They suggested using plots based on the smoothed residuals of Schoenfeld.

Three of the papers, discussed above, are related to the work proposed here. Bacchetti and Segal (1995) and Huang, Chen and Soong (1998) extended the tree method to accommodate time-dependent covariates. Pettitt and Daud (1990) incorporated time-dependent covariates using a Cox proportional hazards model. Our proposed method will be contrasted to these three papers in section 6.1.

3.2 METHODS THAT USE WITHIN NODE HOMOGENEITY

The extensions in the second category focus on within node homogeneity measure to identify distinct nodes, which is present in the minimal error-complexity pruning developed by Breiman, et al. (1984). Gordon and Olshen (1985) were the first to extend the CART algorithm to censored survival data, where the splitting criteria used the distance measures between the Kaplan-Meier curves of the left and right daughter nodes. They seek to estimate the conditional expectation, $E(g(Y)|X)$, where g is an unspecified function, Y is the response, and $\boldsymbol{X}$ is the vector of covariates. Since they do not restrict g to a specific form, this method will work in very general situations. They used the L^p Wasserstein metrics, $d_p(F_z,F_w)$, as the measure of discrepancy between the two survival functions F_z and F_w. Wasserstein metrics were used because they handle censoring simply and focus on the tails of distributions. For a survival function S, let δ_S be the survival function that has mass at at most one finite point and minimizes the Wasserstein distance to S. $D(S, \delta_S)$ is used to measure the variability of S. For a node B, $\hat{S}^t$ is the Kaplan-Meier estimate for the survival function within t. The splitting criterion tries to maximize the difference between the variance of the parent node and the weighted average of the daughter nodes. This difference is represented as,

$$P(B) * D(\hat{S}^B, \delta_{\hat{S}^B}) - [P(L(B)) * D(\hat{S}^{L(B)}, \delta_{\hat{S}^{L(B)}}) + P(R(B) * D(\hat{S}^{R(B)}, \delta_{\hat{S}^{R(B)}})].$$

The error-complexity equation becomes

$$\sum_{B \in \tilde{T}} P(B) * D(\hat{S}^B, \delta_{\hat{S}^B}) + \alpha \left\| \tilde{T} \right\|,$$

Gordon and Olshen (1985) used the same pruning and cross-validation methods as was used in CART. Therefore, the final estimator consists of a Kaplan-Meier estimate at each terminal node.

Davis and Anderson (1989) provided an extension of the methods of Breiman, et al. (1984) to censored survival data. They used a modified exponential model $h(y) = \lambda_j$, for all y in group j where the groups are defined by the vector of covariates x. These groupings will define the nodes in the tree T. Their algorithm partitions each node based on the exponential log-likelihood loss. The split chosen is that which minimizes the loss among the possible splits define by the covariates. Their proposed loss function $R(t)$ is defined as

$$R(t) = D_t - D_t \log(D_t/Y_t)$$

where $D_t = \sum_{i \in t_t} d_i$ is the number of observed deaths and $Y_t = \sum_{i \in t_t} y_i$ is the total observation time in node t.

They used the same pruning method as CART, however they used a slightly different method for cross-validation. An obstacle to cross-validation using their method is the chance of having an estimate of the hazard equaling zero. For example, suppose a tree is grown which contains a node with only censored observations. The hazard estimate within this node would be zero. If an observation that has complete survival information is sent down the tree and falls in the node with a zero hazard estimate, we would obtain an infinite value for the cross-validation estimate of loss within the node. Because of this possibility that a node might only contain censored data, they modified the cross-validation to eliminate the possibility of zero hazard estimates. They used an estimate $\lambda^* = 1/(2Y_t)$ for the hazard within this node for the cross-validation. Their final tree selection was based on the chi-square distribution instead of the one standard error rule in CART since the one standard error rule does not work for censored data.

Therneau, Grambsch, and Fleming (1990) proposed a method in which Martingale residuals from the Cox model were used directly in the CART regression algorithm with squared error loss being the cost in the cost-complexity scheme. Thus, martingale residuals were used in place of the response values and then the CART regression method was conducted, without any changes to the algorithm, to grow a regression tree.

LeBlanc and Crowley (1992) utilized the proportional hazards regression framework to develop a tree-structured method for censored survival data using only the first step of a full likelihood estimation procedure. In their method, LeBlanc and Crowley use the hazard function

$$\lambda(y \mid \boldsymbol{x}) = \lambda_0(y)s(\boldsymbol{x})$$

where $\lambda_0(y)$ is the baseline hazard and $s(\boldsymbol{x}) \geq 0$. LeBlanc and Crowley let $s(\boldsymbol{x})$ represent the relative risk function instead of the traditional use of letting $s(\boldsymbol{x})$ be a loglinear function of x. They used the first step of a full likelihood estimation for the proportional hazards model to grow and prune trees. They employed the deviance as an estimator of the within-node error, which wass used as their pruning criteria. Once a tree was chosen, the full likelihood estimates was obtained through iteration.

The methods of LeBlanc and Crowley assume that we have survival data $(T,\delta,\boldsymbol{X})$, where T is the observation time, δ is the indicator of failure, and $\boldsymbol{X}$ is the vector of M covariates. Suppose that the true survival time U has a distribution F and the true censoring time V has a distribution G so that $\delta = I_{[U \leq V]}$ and $T = min(U, V)$. Assume that U and V are independent given $\boldsymbol{X}$. The learning sample L consists of the independent, identically distributed vectors $\{(t_i, \delta_i, x_i) : i = 1, 2, \ldots, N\}$. The full likelihood of L for a given tree can be expressed as

$$L = \prod_{h \in \tilde{T}} \prod_{i \in S_h} \lambda_h(t_i)^{\delta_i} e^{-\Lambda h(ti)}$$

where $\lambda_h(t)$ and $\Lambda_h(t)$ are the hazard and cumulative hazard functions for node *h*, and S_h is the set of observations $\{i : x_i \in X_h\}$. S_h is simply all of the measurement vectors that are assigned to terminal node *h*. Assuming that the proportional hazards model holds, then

$$\lambda_h(t) = \theta_h \lambda_0(t)$$

where $\theta_h \geq 0$ is a parameter for the node relative risk and $\lambda_0(t)$ is the baseline hazard. Then, the full likelihood for the data given tree *T* as

$$L = \prod_{h \in \tilde{T}} \prod_{i \in S_h} (\lambda_0(t_i)\theta_h)^{\delta_i} \exp(-\Lambda_0(t_i)\theta_h)$$

where $\Lambda_0(t)$ denote the baseline cumulative hazard. Using the baseline cumulative hazard, the maximum likelihood estimator of $\{\theta_h : h \in \tilde{T}\}$ can be obtained by

$$\tilde{\theta}_h = \frac{\sum_{i \in S_h} \delta_i}{\sum_{i \in S_h} \Lambda_0(t_i)}.$$

Since the baseline cumulative hazard is not known, estimates $\hat{\theta}_t$, are used to create an estimate $\hat{\Lambda}_0(t)$ represented as

$$\hat{\Lambda}_0(t) = \sum_{i:t_i \leq t} \frac{\delta_i}{\sum_{h \in \tilde{T}} \sum_{i:t_i \geq t_i, i \in S_{ht}} \hat{\theta}_h}.$$

Replacement of $\Lambda_0(t)$ with $\hat{\Lambda}_0(t)$ in the full likelihood score equation results in the partial likelihood score equation. Iterating over the two estimators will result in convergence to estimates for $\{\theta_h : h \in \tilde{T}\}$ with the property that the ratios of the estimates between nodes is unique. Only the first iteration is used to grow the tree. The Nelson cumulative hazard estimate is used to form $\hat{\Lambda}^1_0(t)$. A one-step estimate for the terminal node relative risk is then formed as

$$\widetilde{\theta}_h^1 = \frac{\sum_{i \in S_h} \delta_i}{\sum_{i \in S_h} \hat{\Lambda}^1{}_0(t_i)}.$$

This estimate can be interpreted as the observed number of deaths in node *h* divided by the expected number of deaths in node *h* under the assumption of no structure in survival times.

LeBlanc and Crowley use the deviance for node *h* as a measure of the error in the model, which is given by

$$R(h) = 2(L_h(saturated) - L_h(\widetilde{\theta}_h)),$$

where $L_h(saturated)$ is the log-likelihood for the saturated model, which allows for one parameter for each observation and $L_h(\widetilde{\theta}_h)$ is the maximized log-likelihood under the condition that $\Lambda_0(t)$ is known.

The estimated deviance residual for observation *i* in node *h* is given by

$$d_i = 2[\delta_i \log(\frac{\delta_i}{\Lambda_0(t_i)\hat{\theta}_h}) - (\delta_i - \Lambda_0(t_i)\hat{\theta}_h)].$$

The recursive partitioning procedure uses the estimates of Nelson estimate $\hat{\Lambda}^1{}_0(t)$ and one-step parameter $\{\hat{\theta}_h^1 = 1{:}h \in \widetilde{T}\}$ to calculate the deviance residuals. This estimator is referred to as the one-step deviance. All possible splits for each node are evaluated and the split that maximizes the reduction in the one-step deviance is chosen. Thus, the split *s* at node *h* is the one that maximizes the improvement given by

$$R(s,h) = R(h) - [R(l(h)) + R(r(h))]$$

where

$$R(h) = \frac{1}{N}\sum_{i \in S_h} [\delta_i \log(\frac{\delta_i}{\hat{\Lambda}^1{}_0(t_i)\hat{\theta}_h}) - (\delta_i - \hat{\Lambda}^1{}_0(t_i)\hat{\theta}_h)]$$

is chosen as the best split.

LeBlanc and Crowley used the same cost-complexity method that was employed by CART where they defined the error-complexity function as

$$R_\alpha(T) = \sum_{h \in \tilde{T}} R(h) + \alpha \left\| \tilde{T} \right\|$$

where α represents a nonnegative complexity parameter and *R(h)* is the impurity of node *h*. The same obstacle existed for cross-validation as that which Davis and Anderson faced. LeBlanc and Crowley proposed a similar solution. For a node *h* that contains only censored observations, replace the zero observed deaths by 0.5 leading to an estimate of the relative risk for node with no observed deaths as

$$\tilde{\theta}_h^1 = \frac{1}{2\sum_{i \in S_h} \hat{\Lambda}_0^1(t_i)}.$$

After the right sized tree is obtained using the cross-validated estimate of the expected one-step deviance, the maximum likelihood estimates for the relative risks of each terminal node are obtained by iteration.

LeBlanc and Crowley compared the performance of their full likelihood method against the exponential likelihood-based method of Davis and Anderson (1989) and also against the martingale residual based method of Therneau, et al. (1990). They demonstrated that using deviance residuals in regression trees is similar to the survival tree methods presented by Segal (1988) and Ciampi, et al. (1986). They also demonstrated that using deviance residuals seem to be better than using the martingale residual method. Along with LeBlanc and Crowley, Ciampi, et al. (1988) used a method based on the assumption that the hazard functions in two daughter nodes are proportional, but unknown.

4.0 RPART ALGORITHM

The recursive partitioning algorithm that we use for the examples presented in this dissertation was developed by Therneau and Atkinson (1997) and is called RPART, an acronym for **R**ecursive **Part**itioning. RPART allows users to create their own splitting rules using code written in Splus. The RPART routines implement many of the CART ideas developed by Breimen, et al. (1984). The RPART algorithm builds binary trees using a two-stage procedure. In the first stage, the single variable that best splits the data into two groups is found. The data is separated and then this process is applied to each of these groups recursively and any subsequent subgroups until no improvement can be made or a minimum size is obtained. During the second stage, cross-validation is used to prune the full tree until a subtree with the lowest estimate of risk is obtained.

4.1 TREE BUILDING

RPART builds the binary tree by using a measure of impurity of a node. Let f be some impurity function and define the impurity of a node A as

$$I(A) = \sum_{i=1}^{C} f(p_{iA})$$

where p_{iA} is the proportion of those in A that belong to class i for future samples. Since we would like $I(A) = 0$ when A is pure, f must be concave with $f(0) = f(1) = 0$. Two candidates for f are the

information index, $f(p) = -p\ log(p)$ and the Gini index, $f(p) = p(1-p)$. Then a split will occur when maximal impurity reduction

$$\Delta I = p(A)I(A) - p(A_L)I(A_L) - p(A_R)I(A_R)$$

is obtained.

4.2 PRUNING AND CROSS-VALIDATION

RPART uses a likelihood based pruning criterion as a way to prune branches of the overfitted tree. Let $T_1, T_2, ..., T_k$ be the terminal nodes of a tree T and $R(T_0)$ be the risk for the zero split tree. Define the cost for the tree to be

$$R_\alpha(T) = R(T) + \alpha|T|$$

where $|T|$ = the number of terminal nodes and the risk of T is

$$R(T) = \sum_{i=1}^{k} P(T_i)R(T_i) .$$

Therefore the risk of T is the sum of all the probabilities of each split tree multiplied by the risk of each split tree. T_α is the subtree of the full model that has minimal cost. Therefore, we can define T_α as the smallest tree T for which $R_\alpha(T)$ is minimized.

RPART then uses cross-validation to choose the best value of α. The data is randomly divided into roughly equal V sized groups where V is an integer usually between 5 and 10. $V-1$ groups, the training set, are used to generate the model and the remaining portion, the test set, is used to evaluate the model. This step is repeated until all test sets have been used in model evaluation. The results of these trees are averaged for all the combinations where one of the

groups is withheld. The one with the smallest risk is chosen as the best-pruned tree. The default for RPART is V=10 or 10-fold cross-validation.

4.3 MISSING DATA

RPART handles missing data differently than many other types of procedures. As long as an observation has a value for the dependent variable and at least one independent variable it will be included in the modeling. RPART, by default, removes only those rows for which either the response y or all of the independent variables are missing. This ability to retain partially missing observations is perhaps one of the most useful features of RPART models. With missing data, the object is still to maximize

$$\Delta I = p(A)I(A) - p(A_L)I(A_L) - p(A_R)I(A_R)$$

The first term stays the same for all variables and splits irrespective of missing data because it is calculated over all observations in node A. However, the last two terms, dealing with the right and left daughters, are modified. Only observations that are not missing a particular variable are used to calculate the impurity for both the left and right daughters $(I(A_L),I(A_R))$. The probabilities for both the left and right daughters $(p(A_L),p(A_R))$ also use only the observations which are not missing a particular variable but they are later adjusted so that their sum will be equal to the probability of node A $(p(A))$. This ensures that the probabilities of the terminal node will sum to one.

When missing values are encountered for variables that are being considered for a split, the missing values are ignored and the probabilities and impurity measures are calculated from

the non-missing values of that variable. For any observations with a missing value for the split variable, surrogate splits are used to allocate the missing cases to the daughter nodes. If the observation is missing the first surrogate split variable then a second surrogate is used and so on. RPART does impose a constraint during the construction of surrogates. Surrogates must send at least two cases down each branch (right and left).

4.4 SURVIVAL TREES

RPART implements what is called "exponential scaling" for survival data. This method stretches the time axis so that the time variable is a straight-line curve for log(survival) under a parametric exponential model. This method assumes that the baseline hazard is linear between observed deaths. The exponential scaling causes earlier splits in RPART's Poisson model to be equivalent to the local full likelihood tree model by LeBlanc and Crowley (1992) that was discussed earlier. This exponential scaling is the default in RPART.

5.0 EXAMPLE OF THE RPART TECHNIQUE

5.1 BACKGROUND: NSABP PROTOCOL B-09

In this section, we illustrate the RPART technique of partitioning survival data according to a number of covariates. In Fisher, et al. (2005), data from National Surgical Adjuvant Breast and Bowel Project (NSABP) Protocol B-09 was used to partition overall survival (OS), disease-free survival (DFS), and recurrence-free survival (RFS) according to clinical and pathologic variables. In this work, all deaths were considered in the calculation of OS. For the DFS endpoint, all recurrences of breast cancer, second primary cancers and deaths were considered whereas only breast cancer recurrences were considered for the RFS endpoint.

The goal of Protocol B-09 was to compare combined chemotherapy of L-phenlalanine mustard (L-PAM) and 5-Fluorouracil (5-FU) with and without Tamoxifen in the management of positive node patients with surgically curable breast cancer. The specific aims of this study were to determine:

(1) whether L-PAM + 5-FU + tamoxifen (PFT) with surgery is more effective than surgery with L-PAM + 5-FU (PF);

(2) whether tamoxifen in combination with L-PAM + 5-FU (PFT) is more effective in patients whose tumors had positive estrogen-receptor sites.

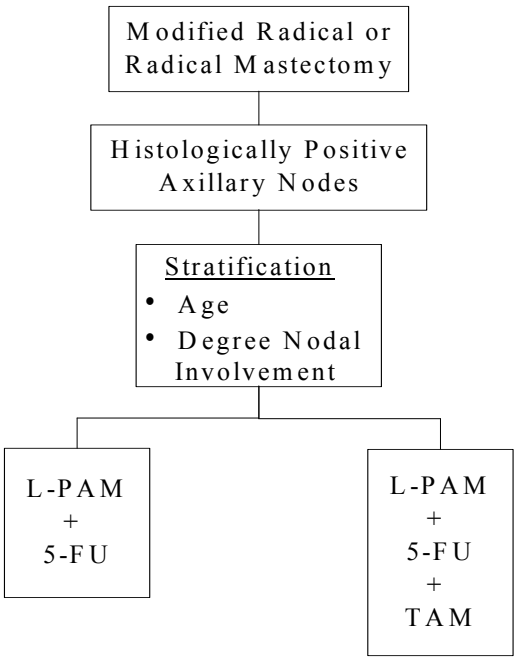


Figure 5.1: Protocol B-09 Schema

1,891 patients had been randomized between January 1, 1977 and May 16, 1980. After May 16, 1980, another 806 patients were registered on the PFT arm. When the protocol was closed on May 31, 1981, 2,697 patients were accrued. Based on interim analyses of disease-free survival and survival, three treatment modifications were instituted:

(1) in 1981, tamoxifen administration was discontinued in patients < 50 years at mastectomy with tumor ER or PR < 10 fmol;

(2) in 1982, tamoxifen was discontinued in patients 50 - 59 years of age at mastectomy with tumor PR < 10 fmol;

(3) later in 1982, an additional 12 months of tamoxifen (without PF) was specified for registered patients who completed two years of PFT therapy.

The protocol was closed to follow-up in April 14, 2004.

5.2 USING RPART IN THE ANALYSIS OF PROTOCOL B-09

The concept for this dissertation was derived while working on the Fisher, et al. (2005) paper. The focus of that work was to determine if there was any optimal choice between the basic immunohistochemical (IHC) methods for determining and scoring estrogen receptor (ER) and progesterone receptor (PR). Part of this process involved relating IHC methods and the quantitative methods and other clinical and pathologic characteristics to the primary outcomes of interest, that is, OS, DFS and RFS. The analysis used data from the NSABP B-09 protocol, described above, which included a subset of 402 patients whose pathologic material was considered to be adequate.

IHC receptors were scored independently by two observers based on percent, intensity, and any-or-none algorithms. Results from these evaluations and from the standard assay using Dextran Coded Charcoal (DCC), along with a computer assisted evaluation and other common pathologic characteristics (listed in Table 5.1) were analyzed using a tree-structured model. Optimum splits were obtained for positive reactions in both univariate and multivariate analyses for OS, DFS and RFS at 5 and 10-year cutoff points.

The median potential time on study for these patients was 23.2 years. The “optimal” splits were calculated by RPART routines developed by Therneau and Adkinson. The RPART routines were performed by using S-PLUS 2000 software.

Table 5.1: Measures and other clinical and pathologic characteristics

Measure	Description	Variable Name	Values
ER/PR	DCC	ptsh/ptsv	continuous 0 – 99998
	Any/none	rae/rap	0=None, 1=Any
	Intensity	rie/rip	0=Negative,1=Dim, 2=Moderate, 3=Strong
	Percent	r%e/r%p	0=None, 1=Slight (<33%), 2=Moderate (34-66%), 3=Marked (>66%)
	Percent * Intensity	rhe/rhp	0-9
	Percent + Intensity	rqe/rqp	0-6
	CAS %	sea/spa	Continuous 1-99
	CAS Quantity	seq/spq	continuous 0.1 – 999
	CAS Intensity	ses/sps	Continuous 1-99
ER only	ACIS Average Intensity	avginten	Continuous 0-999
	ACIS Maximum Intensity	mxinten	Continuous 0-999
	ACIS Average Percent	avgperc	0-100
	ACIS Maximum Percent	mxperc	0-100
Others	Nuclear Grade	nucgr2	0=good, 1=poor
	Number of Positive Nodes	pnod	continuous 1 – 98
	Tumor Size – Clinical	mcsiz	continuous 0.1 – 9.8
	Tumor Size – Pathological	mpsiz	continuous 0.1 – 9.8
	Patient age (years)	age	continuous 20 - 98

During the course of this investigation, we noticed that the effects of the markers on OS, DFS and RFS were attenuating with time. This occurred regardless of what cutoff value was used for each marker. After much discussion, we decided to examine information only at 5 years and at 10 years. An example of the trees that were developed for the paper is shown in Figure 5.2.

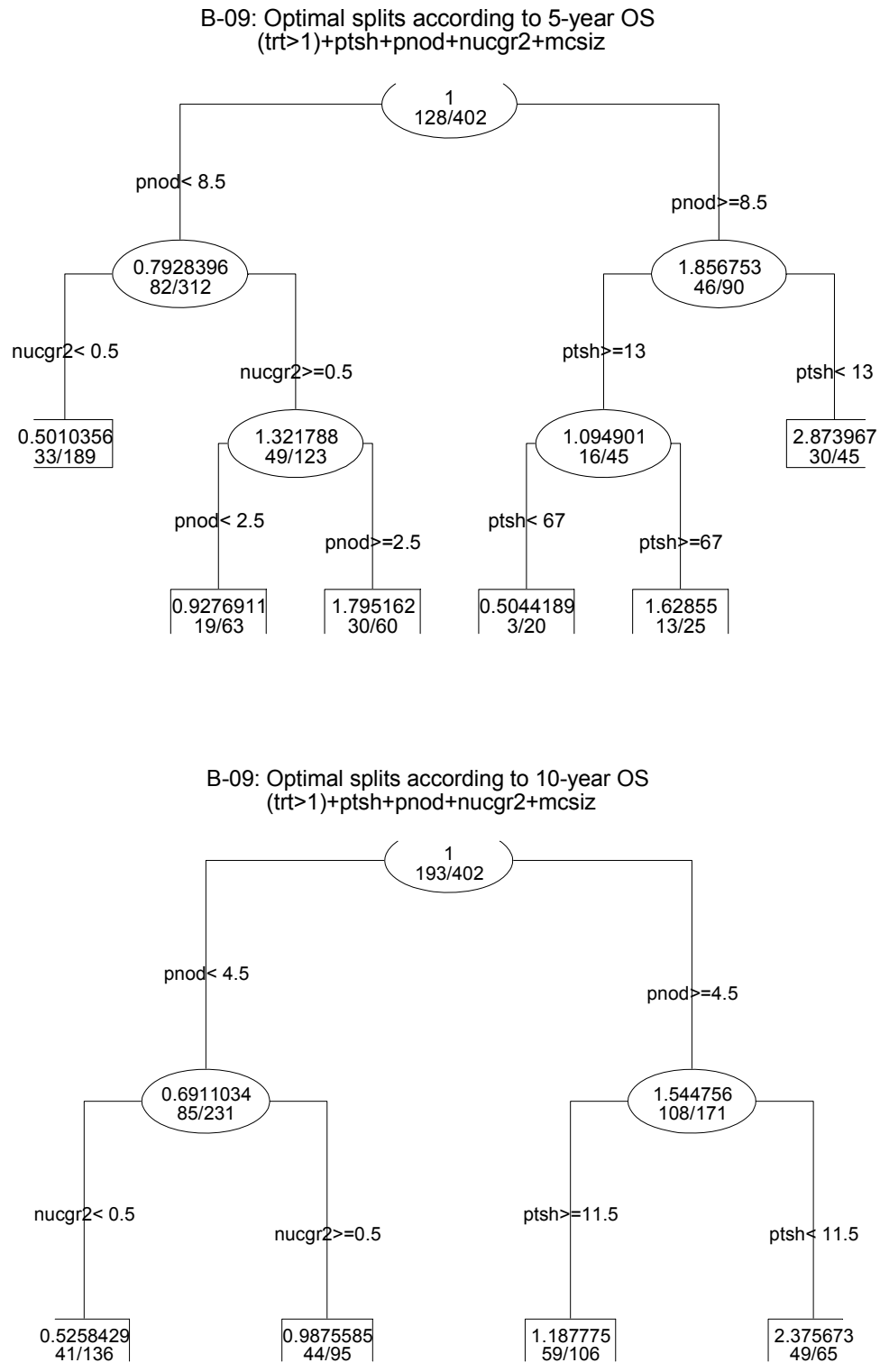


Figure 5.2: 5-year and 10 year Overall Survival Trees from Fisher, et al.

The top tree in figure 5.2 represents the optimal splits for 5 year overall survival as determined by RPART. The root node (first node at the top) contains a sample of 402 patients. The root node is split on the variable "pnod" (number of positive axillary nodes) at 8.5. The number of positive nodes variable contains integers so it does not contain any decimals. The reason that RPART lists it at 8.5 is so that it is clear which observations go to the left and right daughter nodes. Thus, for the number of positive nodes at year 5, those observations with 8 or less positive axillary nodes will go to the left daughter node (the better prognosis group) and those with 9 or more positive axillary nodes go to the right daughter node (the worse prognosis group). The better prognosis group is further split on nuclear grade (nucgr2). Those patients with good nuclear grade go to the left daughter while those with poor nuclear grade go to the right daughter. The worse prognosis group is split on ptsh (ER determined by DCC). Those patients with a dextran coded ER score of 13 or more go to the left daughter while those with a dextran coded ER score of 12 or less go to the right daughter.

The tree on the bottom, in Figure 5.2, represents the splits for 10-year survival. Once again, the root node contains a sample of 402 patients. The root node is split on the variable pnod (number of positive nodes) at 4.5. For 10-year OS, those observations with 4 or less positive nodes will go to the left daughter node (the better prognosis group) and those with 5 or more will go to the right daughter node (the worse prognosis group). The better prognosis group is further split on nuclear grade (nucgr2). Those patients with good nuclear grade go to the left daughter while those with poor nuclear grade go to the right daughter. The worse prognosis group is split on "ptsh" (DCC ER). Those patients with a dextran coded ER score of 12 or more go to the left daughter while those with a dextran coded ER score of 11 or less go to the right daughter.

5.3 FINDINGS

The motivation for examining time-varying effects of covariates on outcome in this dissertation came about when examining the multivariate splits at both 5 years and 10 years. While examining these splits, we noticed that different optimal splits were produced at these times. Our deliberations about when to examine the information led to the following questions:

- Is there a time point (or several time points) at which there are optimal differences in levels of a marker for OS, DFS and RFS? and
- Does the optimal "split point" of a marker change with time?

5.3.1 Univariate Results Examined by Time

To get a better understanding of the splits, we first looked at several markers (treatment, nuclear grade, number of positive nodes, clinical tumor size and dextran coated ER) univariately over time. The first step was to use RPART to find the split points for each of the markers. An optimal binary tree was produced and the split points were recorded. Once the optimal split points were obtained for each marker univariately, z-values were calculated using each of the split points in a Cox proportional hazards model. This two-step procedure was performed for each year of survival. Because we wanted to get a perspective over time, the z-values and split points for each marker are displayed below according to various cutoff times in Figure 5.3. For clarification, the split points are displayed by halves (e.g. number of positive nodes split point at year 10 is 4.5). Again, RPART does this so that it is clear which observations go to the left and right daughter nodes. Thus, for the number of positive nodes at year 10, those observations with

4 or less will go to the left daughter node and those with 5 or more will go to the right daughter node.

Figure 5.3 shows the z-values and splits for several markers using a univariate model. A separate panel is included for each marker. In these panels, the left axis displays the z-values obtained from the Cox proportional hazards model and the right axis displays the split points that were obtained from RPART. As is shown, the z-values fluctuate over time, sometimes drastically. As we discovered in the paper, the splits can be quite different depending if we were looking at OS at 5 years or at 10 years. For example, the first graph on the left in Figure 5.3 is for the number of positive nodes, which was our strongest predictor and is traditionally most related to outcome. The split for the number of positive nodes as predictors of OS at the 5-year cutoff time was 8 or less while the splits at 10-year cutoff time was 4 or less.

Not only are we interested in where a variable might split, we also want to know at what time would be the optimal time to examine these markers. For example, should we look at overall survival at 3 years, 5 years, 10 years, or 15 years? When we looked at splits at each year (1-year overall survival to 20-year overall survival), we noticed the fluctuation of the split points for some of the markers. As we already knew from looking at only 5-year and 10-year cutoff times, the actual number of nodes that was split on changed over time. What is interesting is that for both 3-year and 4-year OS, the optimal split for number of positive nodes was split at 2 or less, where as at 5-years the optimal split for the number of positive nodes jumps to 8 or less. From 6-year to 20-year overall survival, the optimal split stabilizes at 4 or less.

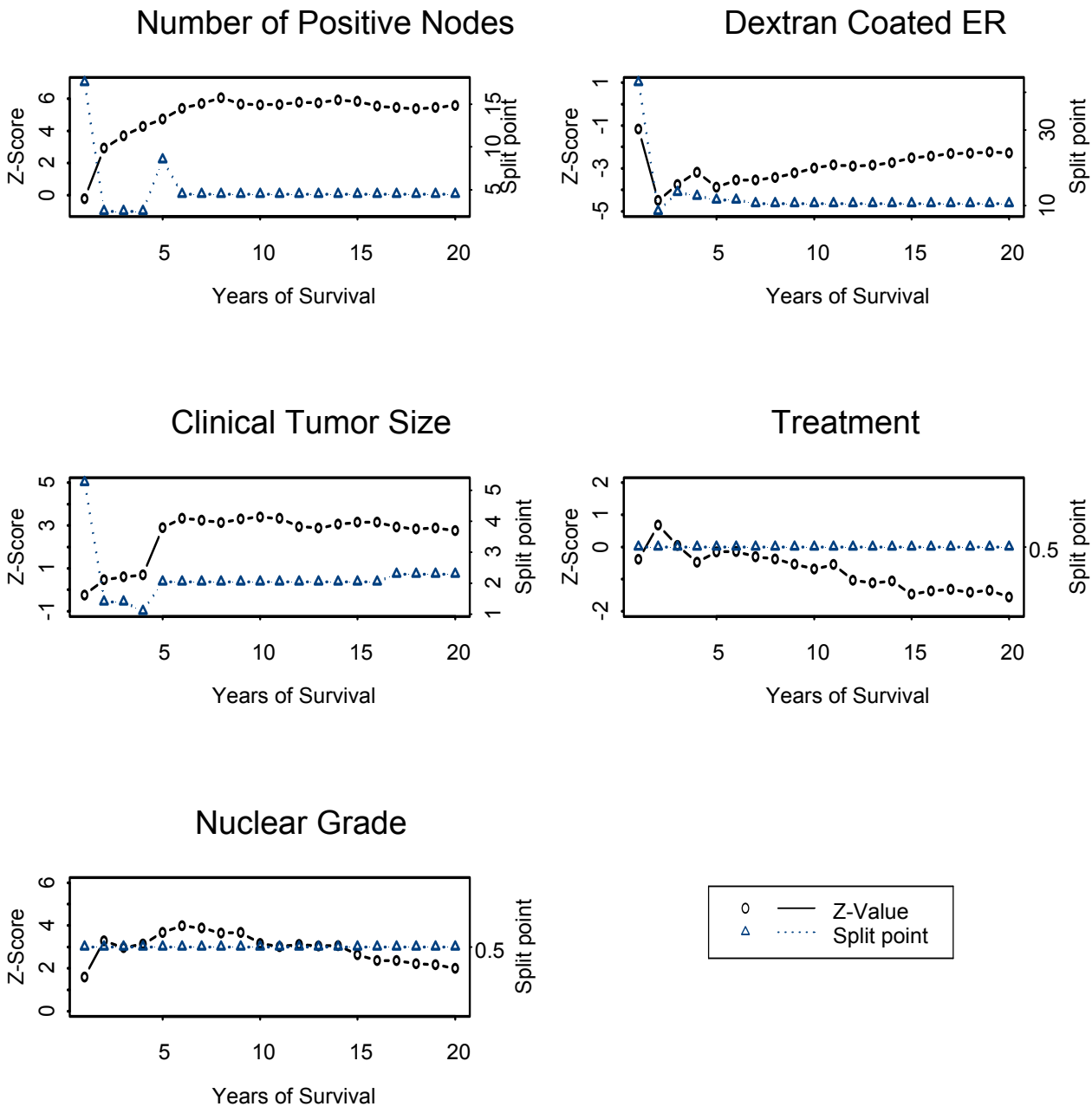


Figure 5.3: Z-values and Splits for Markers (Univariate Model)

The second graph on the left side of Figure 5.3 represents the z-values associated with a Cox PH model and split points for clinical tumor size. The z-values climb at reach a plateau between year 5 and year 11 with a gradual decrease through year 20. The split points fluctuate until it steadies out at year 5. However, the split point does jump up at year 17. The graph at the bottom left of the figure is for nuclear grade. The split points for nuclear grade remain constant because it is a dichotomous variable. The z-values get closer to zero as they approach year 20. The graph at the top right of the figure is for dextran coded ER. This variable was the focal point of the paper with Fisher, et al. and brought about the interest in looking at the other markers. The split points fluctuate until it steadies out at year 7. The z-values get closer to zero as they approach year 20. The second graph on right side of the figure is for treatment. The split points for treatment remain constant because it is a dichotomous variable. The z-values are getting larger as they approach year 20.

5.3.2 Multivariate Results Examined by Time

Next, we examined these markers over time using a multivariate model. Each one of the markers listed above was included in this multivariate model. The first step was to use RPART to find the primary split points for each of the markers using a univariate model. An optimal binary tree was produced using all of the markers in the model and the split points were recorded. Once the optimal split points were obtained for each marker, z-values were calculated using a Cox proportional hazards model that included each of the markers that was dichotomized by its own split point (for example, two groups: number of positive nodes ≤ 8, number of positive nodes ≥ 9). For example at year 5, the Cox proportional hazards model included dextran coded ER (split at 11.5), the number of positive nodes (split at 8.5), clinical tumor size (split at 2.05), treatment

(PF/PFT) and nuclear grade (good/poor). This two-step procedure was performed for each year of survival. To get a perspective over time, the z-values and split points for each marker are displayed below in Figure 5.4. The z-values of the markers represent the values obtained when the primary splits for all of the markers are considered in the model. The z-values and split points were identical to those in the univariate analysis displayed in Figure 5.3. When using all five of the markers in the model from 3-year overall survival through 20-year overall survival, the number of positive nodes was always the first primary split from the root node.

We examined the number of positive nodes further since it was the variable that was the optimal split for 3-year survival through 20-year survival. In RPART, we looked at the improvement value for each split. The improvement value is what determines which variable is split and is derived from the difference in the deviance of the parent node and its daughter nodes. Therefore, the improvement is *n* times the change in impurity index.

As we discussed before, z-values of the variables represent the value obtained when the primary splits for all of the markers are considered in the model. RPART computes the best choices for the primary split and lists the improvement value, which represents the one-step deviance. At the 10^{th} year of survival, the largest improvement is for the variable number of positive nodes (pnod), with an improvement of 31.60. The next best choice is clinical tumor size, with an improvement of 13.09. The improvement for each of the primary splits for marker from the root node is displayed in Figure 5.5.

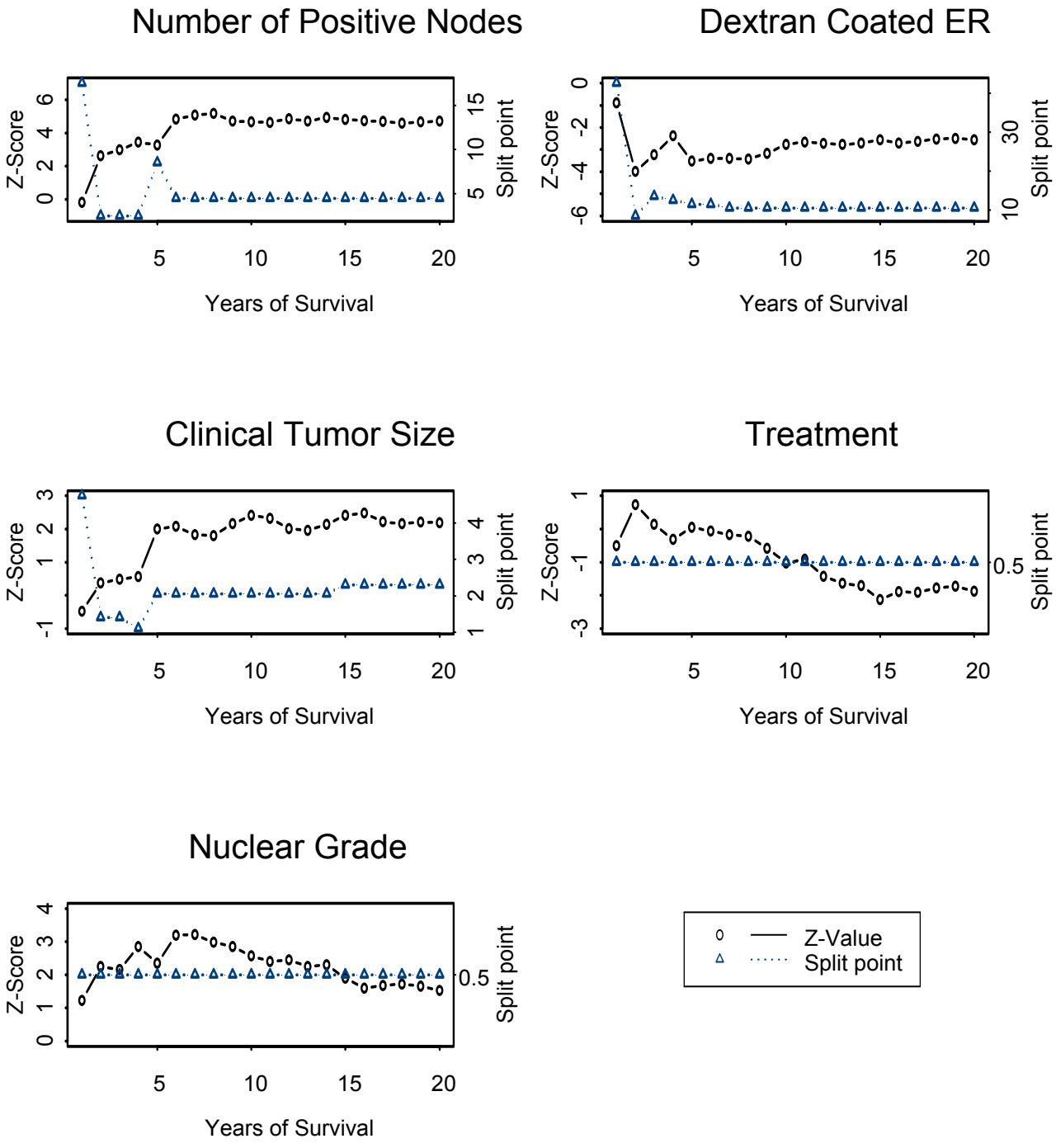


Figure 5.4: Z-values and Splits for Markers (Multivariate Model)

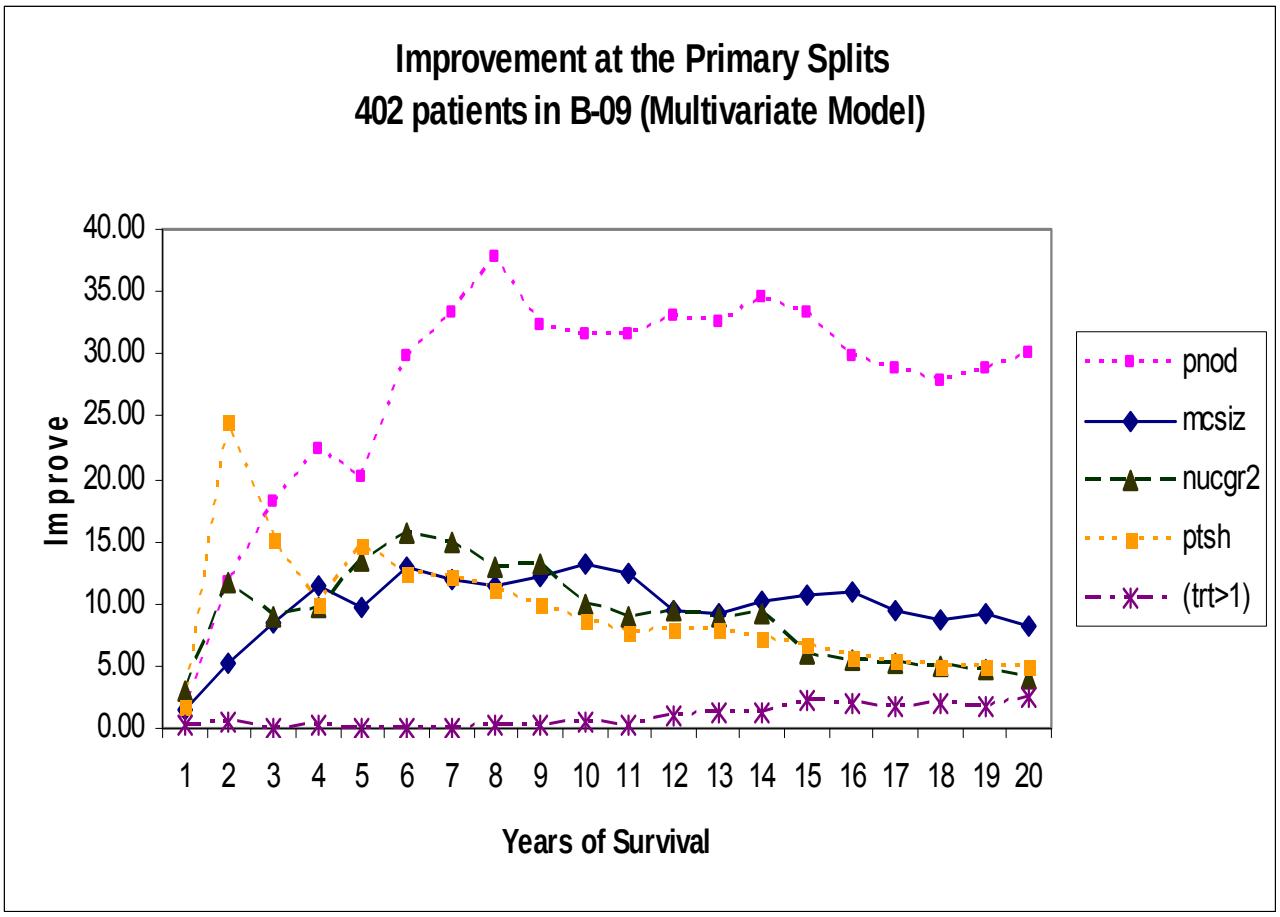


Figure 5.5: Improvement of Primary Splits for All Markers

Figure 5.5 can be used to give us a perspective regarding the improvement at the primary splits over time. The improvement for the marker, number of positive nodes, continues to get stronger until it reaches its peak at year 8. After year 5, the differences in improvement between the best choice (number of positive nodes) and the next best choice are large. As we look at the improvement values for the number of positive nodes, we realize that it is our strongest predictor. When inspecting this figure, the question that arises is "which time point would give us the best split over time?" Specifically, is looking at this data at year 5 or year 10 giving us the optimal split that we can obtain? Remember that the split with the maximum change in the

impurity index will be the optimal split. Therefore in our example, we would have obtained an optimal tree splits if we would have examined the data at year 8 of survival and the optimal split have those with 4 or less positive nodes going to the better prognosis group (left daughter) and those with 5 or more positive nodes going to the worse prognosis group (right daughter).

6.0 PROPOSED METHOD

The idea for this new method started when analyzing a project using the proportional hazards model. The analyses lead to some interesting conclusions. When formulating the analysis plan for the Fisher paper, we decided to take a different approach to analyze the data. Our decision was to use a recursive partitioning model to partition the patient population into homogeneous sub-groups for overall survival (OS), disease-free survival (DFS), and recurrence-free survival (RFS) by the use of different prognostic covariates. We noticed that the effects of the markers on OS, DFS and RFS were attenuating with time. From our additional analyses, we determined optimal times in an ad hoc way and split points for our covariates as they related to OS.

This led us to develop a new method that uses an algorithm that not only tells what the optimal split might be but also at what time this optimal split occurs. The proposed method finds an optimal split point with respect to the level of a marker and across the time points at which the measurements are made. The method was first designed to look at the time continuously. To do this, the RPART algorithm was manipulated so that it would be fixed at time point *t*. All of the split calculations will be performed and saved in memory. This process is done recursively for each of the event times. However, this process can be computationally expensive depending on the sample size. Because of the computation expense, a second process was developed taking more of a life table approach. With the event times being summarized over time intervals, such as year, splits could be obtained at a relatively small number of times.

6.1 BACKGROUND

Three previous papers are closely related to the topic of this dissertation. Bacchetti and Segal (1995) and Huang, Chen and Soong (1998) extended the tree method to accommodate time-dependent covariates. Bacchetti and Segal considered a split based where for a specific value subjects k with $x_{jk}(t) \leq c$ at all times clearly go to the left node, while subjects with $x_{jk}(t) > c$ at all times are assigned to the right node. However there are situations were subjects with $x_{jk}(t) \leq c$ for some failure times and $x_{jk}(t) > c$ at other failure times need to contribute to the left node and some times that they need to contribute to the right node. Bacchetti and Segal defined $X_j(t)$ as non-increasing in t and t_k* as the last time when $x_{jk}(t) \leq c$, with $\tau_k < t_k^*{}_i \leq y_k$, where τ represented the left-truncation time and y represented the end time. They required that subject k be considered part of left daughter node at failure times t_i such that $\tau_k < t_i \leq t_k$* and part of the right daughter node when $t_k^* < t_i \leq y_k$. Therefore, an individual survival experience is split into two pseudo subjects where pseudo-subject k_1 is only at risk up to time t_k* and k_2 is not at risk until after t_k*. A major concern with this method is that the same subject can be assigned to both the left and right daughter nodes.

Huang, Chen and Soong (1998) proposed a model that splits nodes through the interaction of the covariate values and time and establishes measures of improvement on the basis of piecewise exponential survival functions. Their method approximates the hazard function associated with a piecewise exponential failure distribution. The argument t in $\lambda(z,t)$ is treated as a time-dependent covariate in the piecewise exponential survival trees algorithm.

A third paper [Pettitt and Bin Daud (1990)] further developed the use of residuals to estimate time-varying coefficients in a Cox model. They used smoothing techniques on the Schoenfeld residuals to investigate time-dependent effects.

These earlier methods are different from what we propose because they incorporate time-dependent *covariates* to be split in the tree-structured models. In contrast, we propose to split based on the potential *time-varying effects* of the covariates which themselves may either be fixed or time-varying. In this dissertation, we only consider the fixed covariate case. Accordingly, our goal is to find the best point of discrimination of a covariate with respect to not only a particular value of that covariate but also to the time when the endpoint of interest is observed. Hence, determining such a point could be useful in deciding not only *what value of a marker is optimal* for identifying homogeneous subsets of patients but also at *what point in time the decision should be made*.

6.2 METHOD DETAILS

The underlying model associated with the method proposed in this dissertation is the proportional hazard model originally proposed by Cox (1972). The proportional hazards model assumes that the hazards function satisfies the following equation:

$$\lambda_i(t) = \lambda_0(t)\exp(X\beta),$$

where β is a vector of regression coefficients and $\lambda_0(t)$ is the baseline hazard function. By dividing both sides of the equation by $\lambda_0(t)$, one can see that the right hand side of the resulting equation does not depend on time, and thus, provides a proportional hazards structure with a log-

linear model for the covariates. Sometimes the hazard model will incorporate time-varying covariates. With time-varying covariates, the value for the covariate changes over time so that the hazard function satisfies:

$$\lambda_i(t) = \lambda_0(t)\exp(X(t)\beta)$$

A third instance is when you want to incorporate a time effect into the coefficients of the model (e.g. an attenuating treatment effect on an outcome). In this case, the value of the covariate does not change over time but the coefficient does. In other words, treatment does not change over time but the effect of the treatment does. Here the hazard function satisfies:

$$\lambda_i(t) = \lambda_0(t)\exp(X\beta(t))$$

The final case is when the hazard model incorporates both time-varying covariates and time-varying coefficients. The hazard function satisfies:

$$\lambda_i(t) = \lambda_0(t)\exp(X(t)\beta(t)).$$

The new method proposed in this dissertation focuses on third case, that is, the time-varying coefficient model since we are interested in whether the effect of a marker changes over time.

RPART was designed to develop a tree for survival data at one time point. Even though the exponential method in RPART takes into account time to progression, the tree that is constructed is a snapshot at that time point. This snapshot might occur at certain common cutoff points (2 years, 5 years or 10 years) or it might use the longest time point. For example, if the data is censored at 5 years to look at 5 year overall survival, RPART will produce a tree that gives you the optimal split at that time.

Our proposed method is temporal in nature. We begin by developing trees for either all unique event times (as in a Kaplan-Meier analysis) or other pre-specified grouped times (as in a life table analysis). Consequently, instead of considering a landmark time point, e.g. year 5, we

accumulate optimal splits at successive event times and eventually produce a regression tree, which contains the optimal split at the optimal time.

Our method can be implemented using one of two types of filtrations with the first using a "natural filtration" (similar to a Kaplan-Meier analysis) and the second using a filtration that summarizes information over pre-specified time periods (similar to a classic life table analysis). For the first filtration, we consider unique event times. Our algorithm sorts through the unique event times in the data set and arranges them in ascending order. To implement the second filtration, we group the data by year. This allows us to have trees constructed for each year.

After the event times are grouped (either by year or unique event time), the algorithm proceeds in the same manner. As previously stated, all possible splits for each node are evaluated and the split that maximizes the reduction in the one-step deviance is chosen. Thus, the split s at node h is the one that maximizes the improvement given by:

$$R(s,h) = R(h) - [R(l(h)) + R(r(h))]$$

where

$$R(h) = \frac{1}{N}\sum_{i \in S_h} [\delta_i \log(\frac{\delta_i}{\hat{\Lambda}^1_0(t_i)\hat{\theta}_h}) - (\delta_i - \hat{\Lambda}^1_0(t_i)\hat{\theta}_h)]$$

is chosen as the best split.

A tree is constructed for each event time or time grouping. As the algorithm steps through the time dimension (either event times or grouped times), split information is collected for the two best trees. For example, consider that the event times were grouped by year. Assume splits were recorded for years 1 and 2 and that year 2 had a higher improvement. Recall that the improvement involves a "one-step deviance" which can be represented as $\text{Deviance}_{\text{parent}} - (\text{Deviance}_{\text{left}} + \text{Deviance}_{\text{right}})$, which is the likelihood ratio test for comparing two Poisson samples. Now at year 3, a new tree is constructed and its improvement is compared to that of the

previous trees (year 2 and year 1, respectively). If the improvement for the year 3 tree were less than both improvement values for year 1 and 2 then the splits information would be discarded. If the improvement were greater than year 1 but not year 2, the split information would be recorded as the second best tree. If the improvement is greater than both year 1 and 2 then the split information for year 3 would be recorded as the best tree while the split information for the year 2 tree would be moved to the second best tree. This process continues throughout the time dimension.

For instance, let k equal the number of event times or the number of pre-specified groupings. As the algorithm steps through the time dimension ($i = 1, \ldots, k$) all of possible splits for each node are evaluated and the split that maximizes the reduction in the one-step deviance is selected for each event time group. Thus, the split s at node h for each event time grouping i, is the one that maximizes the improvement given by:

$$R_i(s,h) = R_i(h) - [R_i(l(h)) + R_i(r(h))].$$

Therefore, the algorithm takes the $\max R_i(s,h)$ as the optimal split point at the optimal time. Figure 6.1 displays a diagram of the algorithm.

Since RPART was not designed to construct more than one tree at a time, the algorithm had to make special accommodations. In RPART if the root node could not be split, a message would be displayed that no splits were found. Originally when the new algorithm was created, any tree that could not be split would cause the algorithm to fail. To accommodate situations where there are no splits, the proposed algorithm now assigns a zero for the value of both the improvement (one-step deviance) and the time of split. This correction allows the algorithm to continue to step through the time dimension allowing us to find the optimal split.

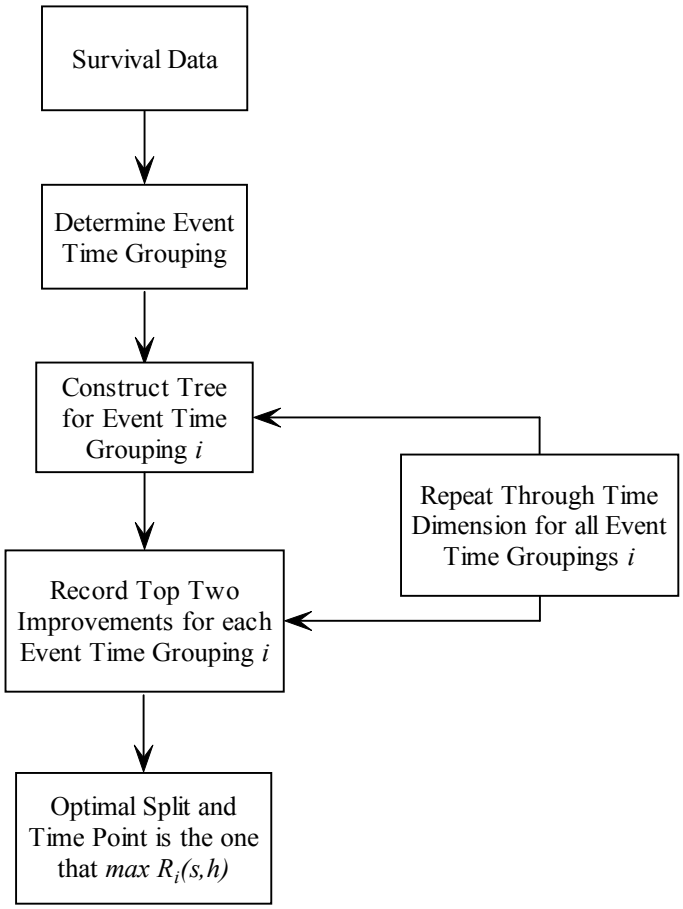


Figure 6.1: Outline of Algorithm for Proposed Method

The computer programming details that were involved in creating the proposed algorithm along with how it was implemented into RPART and S-PLUS, can be found in Appendix 1.

6.3 NSABP EXAMPLE

We first wanted to apply the new method to the NSABP B-09 dataset that was used in the paper by Fisher, et al. (2005). The same covariates (treatment, number of positive nodes, nuclear grade, clinical tumor size and Dextran coded ER), that were used in the multivariate equations of the

paper, were used to test the new method. Both types of filtration (all unique event times and other pre-specified grouped times, e.g. years) were used. The results are displayed in Table 6.1.

The results turned out pretty much as expected. Using years as our pre-specified grouped times, the optimal split occurred at 8 years. Here, the number of positive nodes was split between four and five. Therefore, those patients with four or fewer positive nodes were sent to the left daughter (our better prognosis group) and those patients with five or more positive nodes were sent to the right daughter (worse prognosis group). The second best split occurred at 14 years.

Using all unique event times, both the optimal split and the second best split again occurred between four and five positive nodes. The optimal split occurred at 7.62 years, while the second best split occurred at 7.96 years.

Table 6.1: NSABP Example Results

Filtration	Optimal Split	Covariate	Split	Time of Split
Years	Optimal Split	Number of Positive Nodes	4.5	8 years
	2nd Best Split	Number of Positive Nodes	4.5	14 years
All unique event times	Optimal Split	Number of Positive Nodes	4.5	7.62 years
	2nd Best Split	Number of Positive Nodes	4.5	7.96 years

6.4 SIMULATION

To better understand the statistical properties of the new algorithm, we developed a series of simulations based on both the exponential and Weibull failure distributions. In our simulations, we wanted to assess the effects of distribution, sample size and censoring on the performance of the tree model. The two distributions were used to randomly generate time-to-event data. Four parameter values were generated for each distribution. The strategy was to examine two cases. In the first case, the parameters of the exponential and Weibull distributions were given slightly different values. This gave us a feel for when the optimal split times would be achieved. In the second case, the parameters of the exponential and Weibull distributions were set to be equal for each of the four groups. This second case allowed us to see what happened under a null hypothesis. A simulation consisted of calculating a tree for each of the event times (or grouped times) and gathering data for the "top two trees". Sample sizes of 200, 500, 800 and 1000 were used to construct datasets of simulated events. Using these different sample sizes for each of the two distributions assessed the performance of the method with small, medium and large sample sizes. A simulation was performed for each distribution and each sample using both no censoring and 20% censoring. We simulated 500 datasets for each combination of the distribution, cases (parameters equal and slightly different), sample size and censoring. Therefore, sixteen simulations were conducted for each of the two failure distributions.

Because of the computational time it took to perform the simulation using the unique time event process, we focused our efforts on the life table process, which grouped the event times into years. For each simulation, right censoring was administered after 20 years so that it

would reflect real life data. For each of the simulations, we collected the improvement values, splits (value that the group was split on), and the year (our time point) for the best initial split and the improvement values, splits, and the year for the second best initial split.

The simulations are used to assess the statistical properties of this proposed methodology. The null distribution was compared to an alternative distribution to determine how well the method works.

6.4.1 Exponential Distribution Simulations

The exponential distribution is often used in simulations to try to replicate survival data. The exponential distribution is characterized by a constant hazard rate. The RPART software uses exponential splitting, which stretches the time axis so that the time variable is a straight-line curve for log(survival) under a parametric exponential model. So it is only natural that we would explore how the data would relate to the exponential distribution.

For the exponential distribution, four different strata (called groups in our simulations) using different parameters were created to see how the new algorithm would split the tree. Event times were randomly generated for the exponential distribution using the rexp() function. The following parameters were used: $\lambda_1 = 0.02$, $\lambda_2 = 0.021$, $\lambda_3 = 0.022$, $\lambda_4 = 0.023$. These parameters were used to generate event times that would be similar to what might be seen in cancer data. We wanted to pick values for the parameters that were close together so that we could compare the trees to a null situation, where all of the parameter values are equal. The corresponding algorithm was called using these randomly generated event times as our outcome variable and using group as our categorical predictor. A binary tree that shows the four different strata is displayed in Figure 6.2 below.

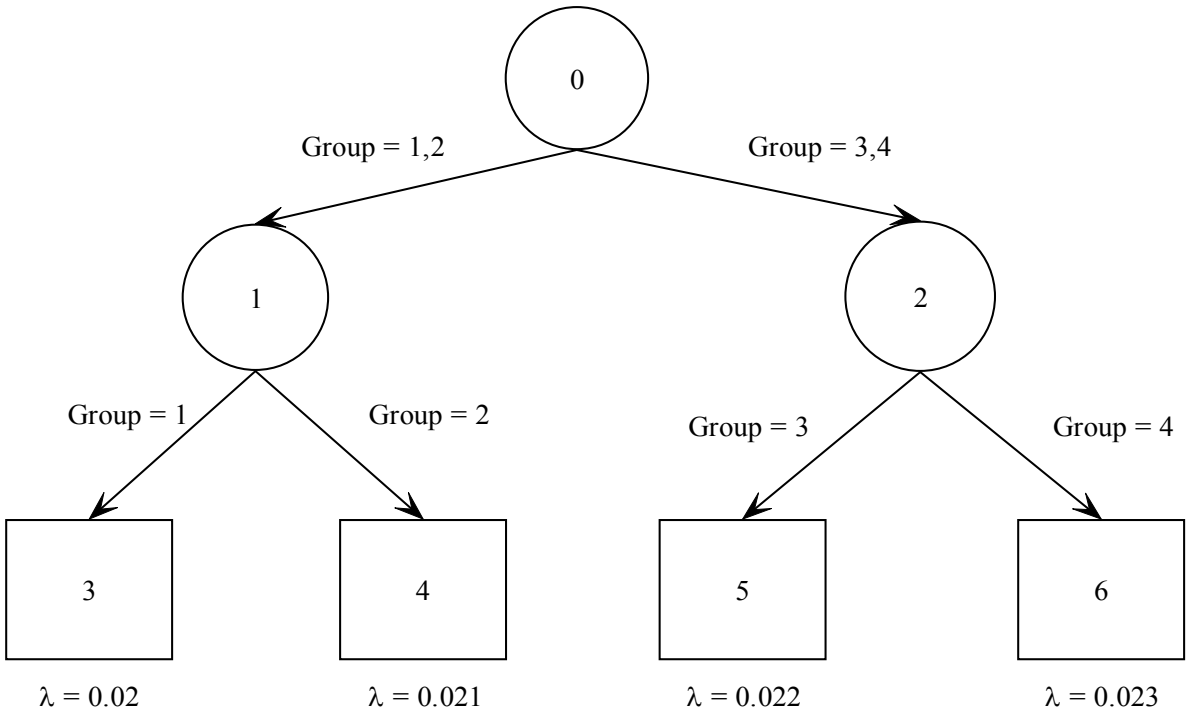


Figure 6.2: Simulation data structure using Exponential Distribution

The overlayed survival curves displayed in Figure 6.3 depict a single realization of the simulated event times for the four different groups. As expected, there is not much difference between the curves for each group. The curves show a quickly accelerating disease were half of the patients have died by the first year.

Another set of simulations was performed where the parameters for each group were equal, $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0.02$. We did this to examine the behavior of the algorithm under a null hypothesis. From this set of simulations, we were able to determine how often splits would occur even though the parameter values were not different, therefore determining the probability of this value occurring by chance.

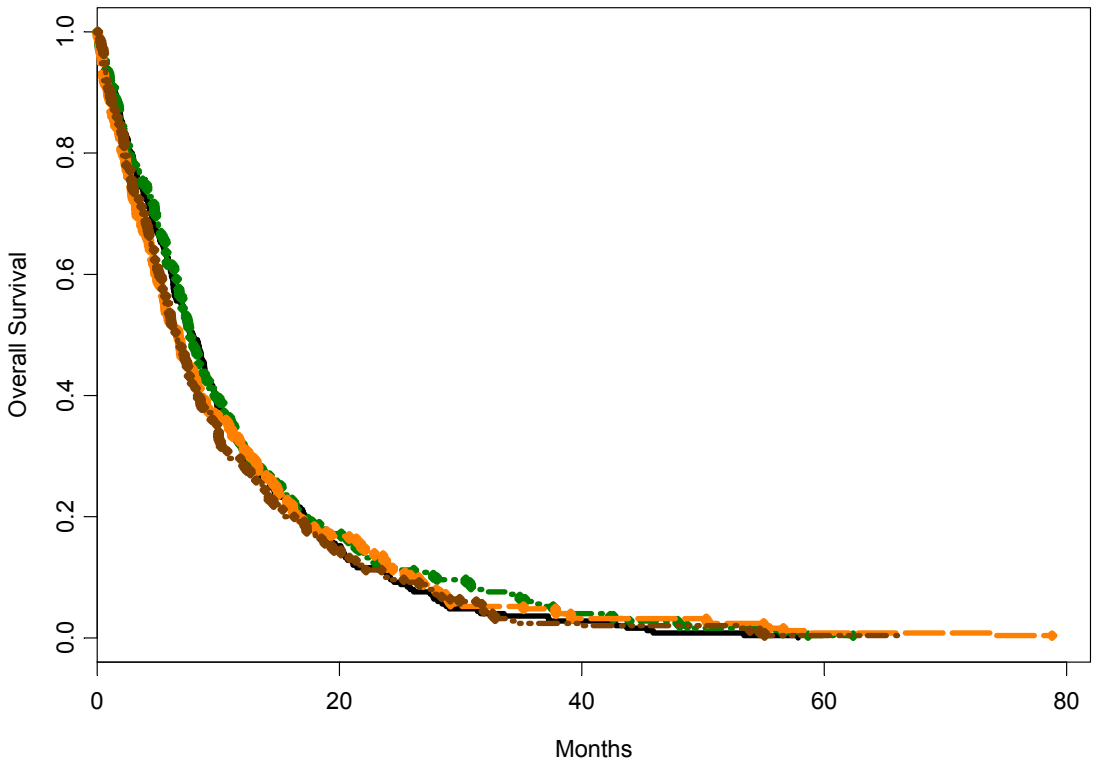


Figure 6.3: Survival Curve for Exponential Distribution Simulation (Parameters not equal)

A summary of the simulated data for the first set of simulations is given in Table 6.2. Here descriptive statistics are given of the event times that were constructed for each group. This gives us an idea of how our optimal tree would look like if we only ran the algorithm once. As expected, the mean number of months until an event occurs was very close among the groups where the parameter values were different.

Table 6.2: Event Times for Tree at First Replication under Exponential Distribution

		Sample Size							
		200		500		800		1000	
	Group	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**
Parameters equal	Group 1 (λ=0.02)	41.4	5.7	45.6	4.0	47.8	3.6	47.3	2.7
	Group 2 (λ=0.02)	47.4	7.2	53.7	4.8	48.8	3.9	47.6	3.1
	Group 3 (λ=0.02)	45.7	6.1	55.9	5.3	51.1	3.4	47.9	3.1
	Group 4 (λ=0.02)	38.9	5.4	57.1	4.7	50.3	3.6	48.9	3.2
Parameters not equal	Group 1 (λ=0.02)	52.2	6.5	48.0	4.1	50.7	3.4	49.4	3.0
	Group 2 (λ=0.021)	49.4	7.5	49.6	4.5	50.8	3.4	48.8	3.4
	Group 3 (λ=0.022)	44.5	7.1	45.1	4.0	45.5	3.5	49.2	2.9
	Group 4 (λ=0.023)	38.2	4.5	42.0	3.6	48.1	3.2	47.1	2.9

Table 6.3 shows the comparison between the null situation (where the parameters are equal) and the alternative (parameter values are slightly different for each group). The null case allows us to see what the probability of a split is by chance. As expect with a large sample size, the number of replications that had a split was very small. For a sample size of 1000 and no censoring, only 2 replications out of 500 (98.6%) had a tree that split on the root node. As the sample size for each replication decreased, the chance that a split happened by chance increased. Our alternative case showed similar characteristics as was shown in the null case. For a sample size of 1000, now 35 replications out of 500 (7%) had a tree that split on the root node. As the sample size for each replication decreases, the total amount of splits detected increase at a quicker rate for the alternative case than the null. Therefore, when a sample size of 500 is used, the percent of splits are quite different (Null - 12.8%, Alternative – 25.2%) between the cases.

Table 6.3: Percent of Replications that did not Split under Exponential Distribution

			Groups			
	Sample Size	**Censoring**	**No Split**	**Split between 1&2**	**Split between 2&3**	**Split between 3&4**
Parameters Equal	**200**	**0%**	32.8%	23.8%	21.4%	22.0%
		20%	30.8%	25.6%	19.8%	23.8%
	500	**0%**	87.2%	3.6%	4.4%	4.8%
		20%	86.2%	4.6%	4.2%	5.0%
	800	**0%**	98.0%	1.2%	0.2%	0.6%
		20%	96.8%	1.2%	1.0%	1.0%
	1000	**0%**	99.6%	0%	0%	0.4%
		20%	98.6%	0.8%	0%	0.6%
Parameters Not Equal	**200**	**0%**	32.0%	21.0%	23.4%	23.6%
		20%	30.2%	24.2%	21.2%	24.4%
	500	**0%**	74.8%	8.8%	9.8%	6.6%
		20%	76.2%	7.6%	9.2%	7.0%
	800	**0%**	89.2%	3.0%	4.8%	3.0%
		20%	91.8%	2.6%	3.2%	2.4%
	1000	**0%**	93.0%	2.6%	2.6%	1.8%
		20%	94.8%	1.0%	2.4%	1.8%

However, when a small sample size (n=200) was used, the percent of splits found was almost equal.

Censoring had a slight effect on whether the algorithm found any splits. For the null case, the 20% censoring had the same effect regardless of the sample size. For each sample size, the number of trees that had a split was slightly higher compared to results where there was not censoring. When the parameters were not equal (alternative case), the trend continued for the small size of 200. However, for the larger sample sizes, the number of trees that had a split was slightly lower compared to results where there was not censoring.

We are not only interested if the tree split or not but also when the optimal splits occur. The percent that the split occurred at each year is displayed in Figure 6.4 for each group. In the figure, the null cases are on the left side and the alternative cases are on the right side. The

number in parentheses, located in the title for each of the graphs, represents the number of replications that did split while "sample" indicates the sample size. The graphs display where and when the splits happen. As can be seen, regardless of which group it is split between, most of the splits happen at year 1. The sample size and case (alternative or null) do not seem to affect the optimal split point or time.

Figure 6.5 displays the percent of splits for each group and year when 20% censoring is administered. For the smaller sample sizes, there does not appear to be any difference in the optimal split point or time between the cases. In the larger sample sizes, there seems to be a little more variability of when the splits occur. However with these larger samples sizes, we are dealing with a small number of replications that actually found an optimal split.

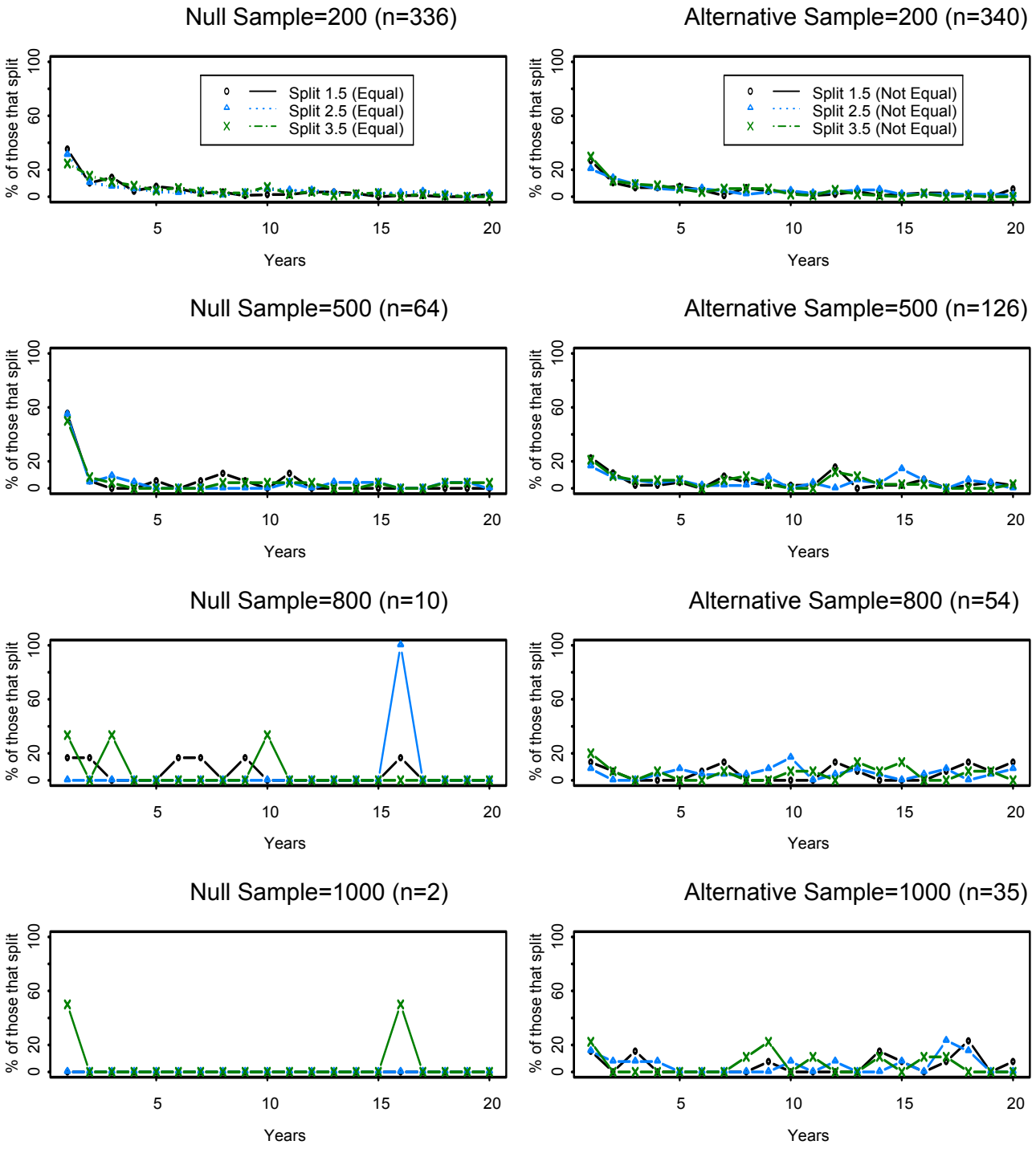


Figure 6.4: Splits by Time Points for Exponential Distribution (No Censoring)

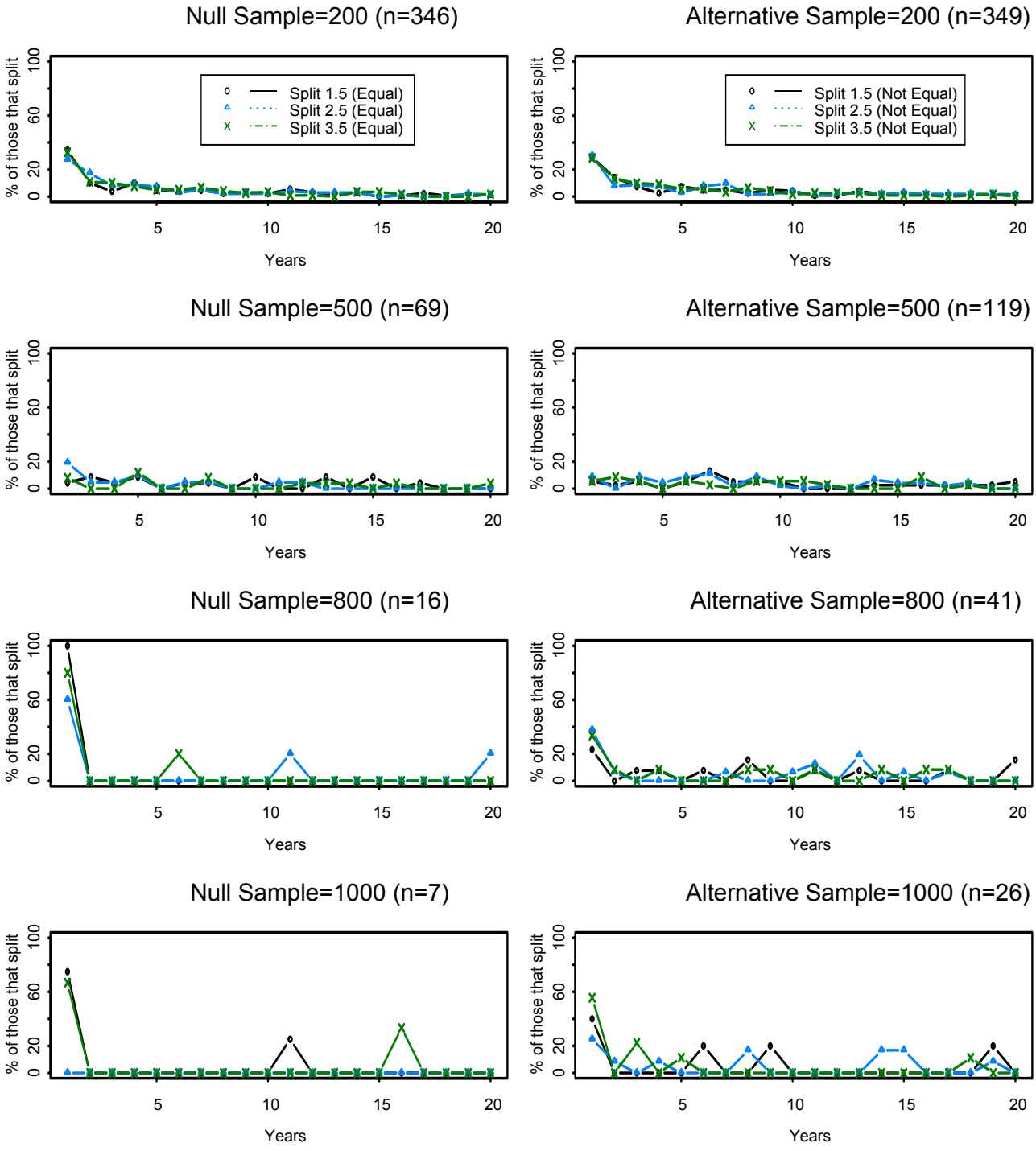


Figure 6.5: Splits by Time Points for Exponential Distribution (20% Censoring)

6.4.2 Weibull Distribution Simulations

We want to see how the new algorithm would work when we used the assumption that the data was under a Weibull distribution. The Weibull distribution is often used because of its flexibility. The Weibull distribution can assume the appearance of several different distributions, one of them exponential. Unlike the exponential distribution, it does not assume a constant hazard rate and therefore has broader application. The Weibull distribution is characterized by two parameters, a shape and scale parameter.

The simulations using the Weibull distribution were constructed in a similar way as the exponential distribution simulations. Four different groups, using the same value (scale=55) for the scale parameter but different value for the shape parameter, were created to see how the corresponding algorithm would handle the splitting of the tree. The following values for the shape parameter were used: $\beta_1 = 2$, $\beta_2 = 2.1$, $\beta_3 = 2.2$, $\beta_4 = 2.3$. Again, the values used for the shape and scale parameters were used to try to generate event times that would be similar to what we might see in cancer data. Like the exponential distribution, we wanted to pick parameter values that were very close together so that we could compare the trees to a null situation, where all of the parameter values are equal. Event times were randomly generated for the Weibull distribution using the rWeibull() function. The new algorithm was called using these randomly generated event times as our outcome variable and using group as our categorical predictor. A binary tree that shows the four different strata is displayed in Figure 6.6.

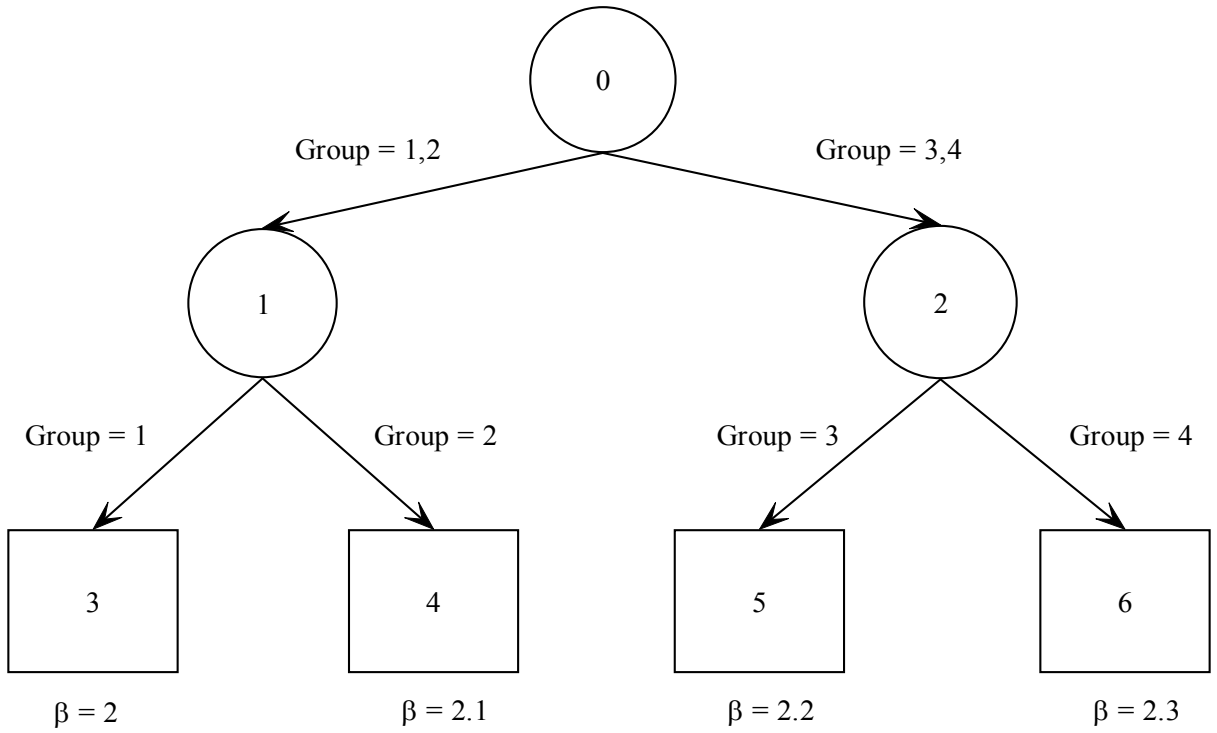


Figure 6.6: Simulation data structure using Weibull Distribution

The survival curve is used to display the overall survival as a function of time for the simulated event times of the four different groups in Figure 6.7. As expected, there is not much difference between the curves for each of the strata. Because of the scaling parameter of the Weibull distribution, the curves show a slower paced disease, than the exponential distribution, were half of the patients have died around the fourth year.

Another set of simulations was performed were the parameters for each group were equal, $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 2$. We did this so that this set of simulations could represent what happens to the algorithm under the null hypothesis.

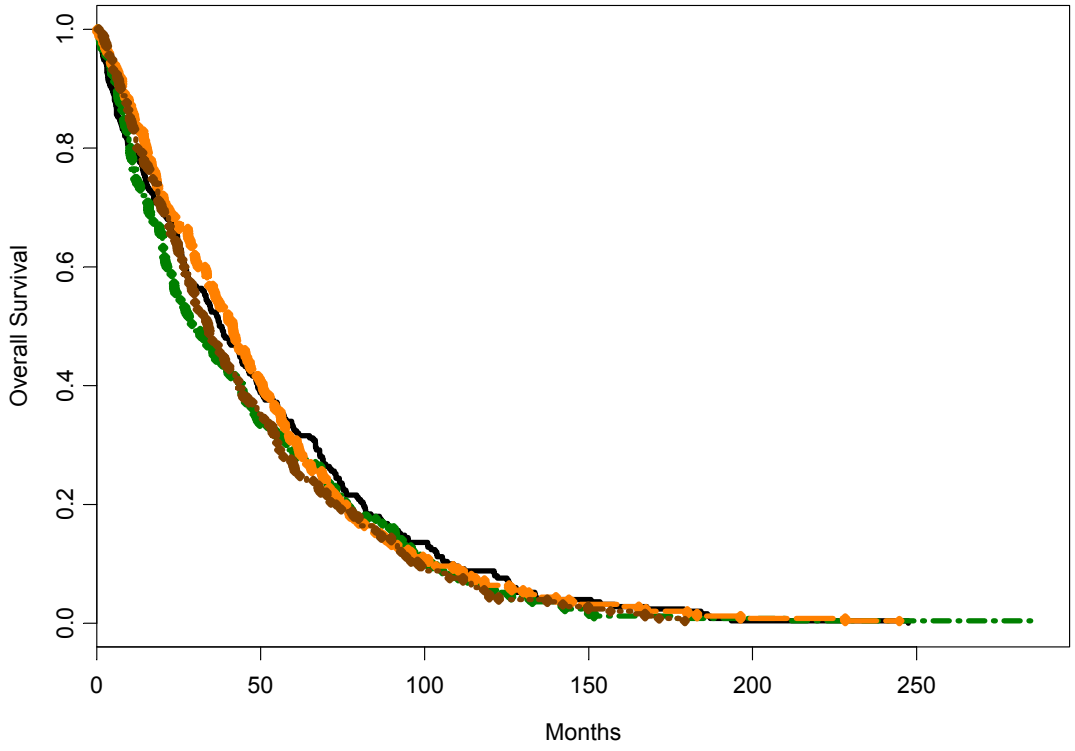


Figure 6.7: Survival Curve for Weibull Distribution Simulation (Parameters not equal)

An example of the groups for the Weibull distribution was given in Table 6.4. Once again the descriptive statistics given were of the event times that were constructed for each group during the first replication of the algorithm. As expected, the mean number of months, until an event occurs, was very close among the different groups.

Table 6.4: Event Times for Tree at First Replication under Weibull Distribution

		Sample Size							
		200		500		800		1000	
	Group	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**	**Mean**	**SE**
Parameters Equal	Group 1 (β=2)	50.5	4.1	48.4	2.2	47.6	1.8	47.6	1.5
	Group 2 (β=2)	47.3	3.4	46.4	2.1	48.4	1.9	47.4	1.8
	Group 3 (β=2)	49.9	3.6	45.2	2.0	48.6	2.0	46.2	1.6
	Group 4 (β=2)	57.2	3.5	43.6	2.1	48.9	1.9	52.2	1.6
Parameters Not equal	Group 1 (β=2)	52.2	3.1	56.7	2.5	45.3	1.6	48.2	1.5
	Group 2 (β=2.1)	48.0	3.0	50.1	2.2	50.2	1.7	47.3	1.4
	Group 3 (β=2.2)	44.2	3.0	51.1	2.2	47.8	1.6	51.0	1.5
	Group 4 (β=2.3)	48.3	3.0	48.7	1.9	47.7	1.6	50.4	1.4

Table 6.5 shows the comparison between the null situation (where the parameters values are equal) and the situation where the parameter values are slightly different for each group using the Weibull distribution. As we saw with the exponential distribution, the number of replications that had a split was very small with a large sample size. The percentage of replications that did not split did differ for the Weibull distribution when compared to the exponential distribution. For a sample size of 1000, only 16.2% of the replications out of 500 had a tree that split on the root node. There was a greater probability of a split happening by chance when using the Weibull distribution. Once again as the sample size for each replication decreased, the chance that a split happened by chance increased.

Table 6.5: Percent of Replications that did not Split under Weibull Distribution

			Groups			
	Sample Size	**Censoring**	**No Split**	**Split between 1&2**	**Split between 2&3**	**Split between 3&4**
Parameters Equal	**200**	**0%**	10.6%	31.0%	28.6%	29.8%
		20%	7.0%	31.4%	34.0%	27.6%
	500	**0%**	50.4%	15.4%	16.4%	17.8%
		20%	43.6%	16.2%	21.2%	19.0%
	800	**0%**	76.2%	7.8%	8.0%	8.0%
		20%	63.4%	14.6%	11.4%	10.6%
	1000	**0%**	83.8%	4.4%	6.0%	5.8%
		20%	80.6%	5.2%	8.2%	6.0%
Parameters Not Equal	**200**	**0%**	5.2%	34.6%	28.0%	32.2%
		20%	3.2%	33.0%	29.0%	34.8%
	500	**0%**	33.6%	24.4%	19.4%	22.6%
		20%	29.4%	25.8%	23.8%	21.0%
	800	**0%**	52.8%	15.8%	17.0%	14.4%
		20%	47.4%	17.6%	18.4%	16.6%
	1000	**0%**	63.8%	9.6%	13.8%	12.8%
		20%	59.8%	14.6%	14.8%	10.8%

Our alternative case showed similar characteristics as was shown in the null case. For a sample size of 1000, now 36.2% of the replications had a tree that split on the root node. As the sample size for each replication decreases, the total amount of splits detected did not increase at a quicker rate for the alternative case than the null as was seen in the exponential distribution. The difference between the two cases did increase when the sample size was decreased to 800. However, as the sample size decreased further, the difference between the cases decreased. This is mainly because the amount of splits detected is so high when a sample size of 500 or smaller is used. Like the exponential distribution, when a small sample size (n=200) is used, the percent of splits found is almost equal. Interestingly, only 5.2% of the replications in the alternative case (compared to 10.6% in the null case) did not detect a split with the small sample size.

Censoring had a slight effect on whether the algorithm found any splits. The Weibull distribution differed from how the exponential behaved. For each sample size, the number of trees that had a split was slightly higher for those that had 20% censoring compared to results where there was not censoring. These slightly higher rates happened regardless of whether the parameters were equal or not. This differs from the exponential distribution were the effect of the censoring was different based on whether the parameters were equal or not.

The percent that each split occurred by each year is displayed in Figure 6.8 for each group. In the figure, the null cases are on the left side and the alternative cases are on the right side. The number in parentheses, located in the title for each of the graphs, represents the number of replications that did split while "sample" indicates the sample size. As can be seen, regardless of which group it is split between, a majority of the splits happen at year 1. The Weibull distribution differs from the exponential because the time points are spread out a little more evenly. In the exponential, very few of the splits happened after the first couple of years. In the Weibull, the splits occurred more often (compared to the exponential) in the later years. The sample size and case (alternative or null) do not seem to affect the optimal split point or time.

Figure 6.9 displays the percent of splits for each group and year when 20% censoring is administered. Sample size does not seem to influence when the splits occur since all of the graphs look similar. The same can be said about the different cases. The results from the case where the parameters are equal match those where the parameters are not equal. The results of these optimal splits seem to mirror the results of the splits when no censoring is administered. Therefore, it censoring does not seem to effect the selection of the optimal split.

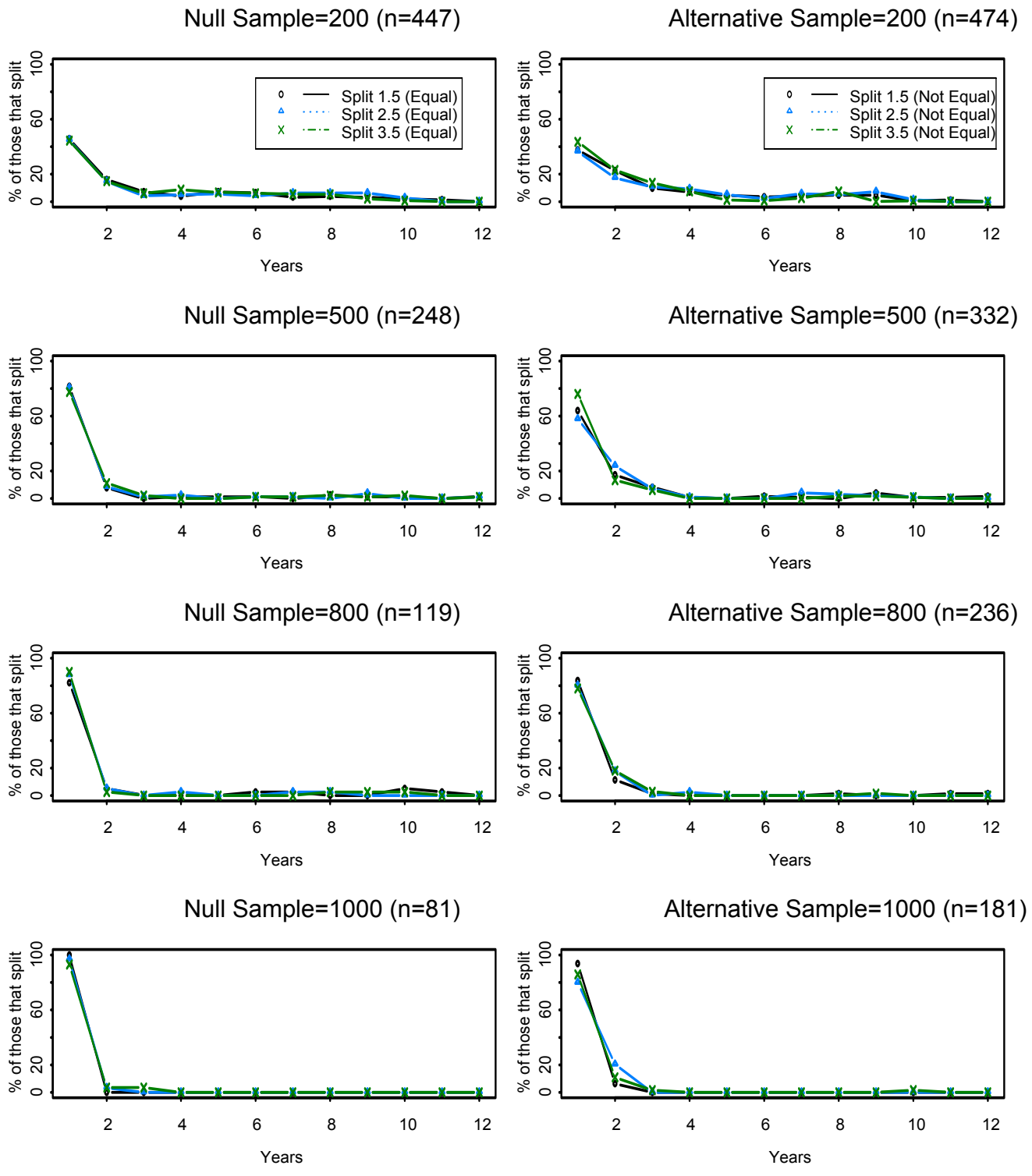


Figure 6.8: Splits by Time Points for Weibull Distribution (No Censoring)

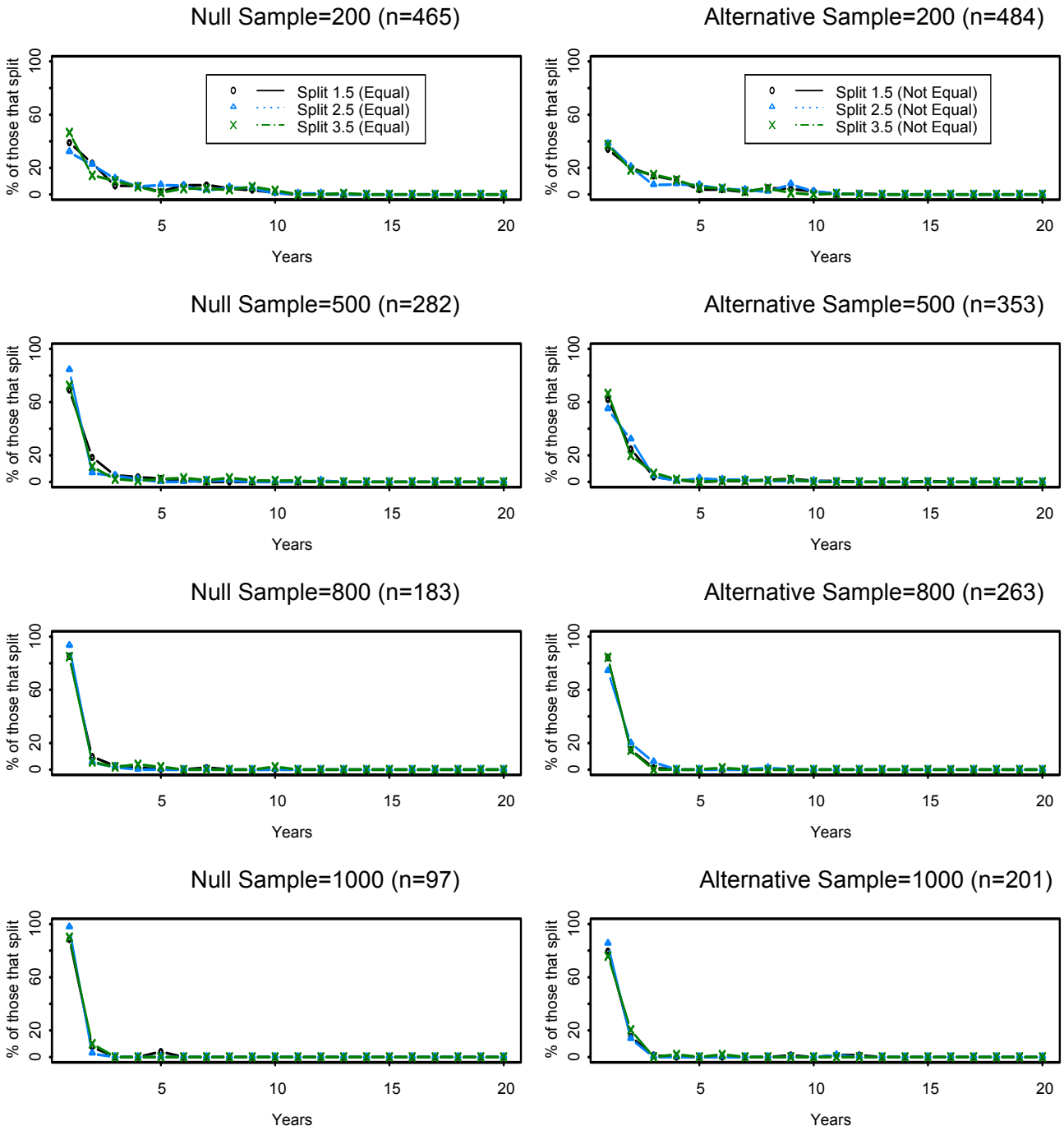


Figure 6.9: Splits by Time Points for Weibull Distribution (20% Censoring)

6.4.3 Summary of Simulation Results

If the underlying failure distribution is exponential, the statistical properties of optimal splitting are not unreasonable. Optimal split times in cases where a split occurred appeared to occur early for small sample sizes and in the few cases where splits occurred for large sample sizes, no such pattern was evident. There was not much difference between the results of the simulations with 0% and 20% censoring.

For the Weibull distribution, more splitting occurred under both the null and alternative cases. If a split was detected, it was almost always detected at an early time point. Once again, censoring did not have much effect on whether a split occurred.

7.0 DISCUSSION

We've extended recursive partitioning in survival analysis so that splitting can accommodate cases where a covariate effect on an outcome is time-varying. Determining an optimal split using our method can be useful in deciding not only *what value of a covariate is optimal but also at what point in time the decision should be made*.

Our proposed method also gives us a measure of robustness of the split points. Robustness is important because it is the measure of the capacity for results to remain unaffected by small variations in underlying assumptions and parameters. In applications such as ours, the split points may vary over time but will usually stabilize after a period of time. By examining these types of patterns, we saw how dichotomized markers affected short and long-term censored outcomes such as survival.

7.1 IMPLICATIONS OF PROPOSED METHOD

This dissertation can be extended across many areas of medicine and public health. Both areas are dependent upon and their practice overlaps around population-based studies. Population-based studies are studies using large samples of patients to draw inferences about the experiences of an entire population based on an evaluation of a sample.

The focus of this dissertation is on survival analysis. Studies that involve survival analysis are population-based studies. Typically, survival analysis allows us to assess time to

event (morbidity, mortality) data. The purpose of it is to determine whether the variable involved increases or decreases the odds of a patient surviving a disease given a certain treatment.

Public health uses these population-based studies to develop public health policies. Public health focuses on protecting the health of an entire population by using education, policies and conducting research.

The practice of medicine has increasingly become evidence-based. With this current focus on evidence-based medicine (EBM), survival research has become increasingly more valuable. Studies focusing on survival analysis provide a perfect example of how population based examination or analysis and information is necessary to the practice of EBM.

The practice of EBM is the application of the findings from population-based studies to the treatment of individual patients. EBM has been defined as "the conscientious, explicit, and judicious use of current best evidence in making decisions about the care of individual patients. The practice of evidence based medicine means integrating individual clinical expertise with the best available external clinical evidence from systematic research." (Sackett, 1996) The systematic research Sackett references come from population-based studies---the mainstay of public health. The practice of evidence based medicine takes the results of population based studies and applies them to the treatment of individual patients often using the statistical technique numbers needed to treat (NNT) to guide the application. The statistical technique numbers needed to treat represents the number of patients who must be treated for a given period to achieve an event or to prevent an event. The NNT is the reciprocal of the absolute risk reduction, which is difference in event rates for two groups, usually treatment and control. Survival analysis has become an essential component in guiding the practice of EBM by

identifying the number of patients needed to treat using a certain modality in order for the patient to survive or overcome disease.

Clearly, the borders of public health practice and the practice of EBM overlap. Without public health research, the practice of evidence-based medicine would not be possible. Without the population research, you could not infer systematically determined results to single patients. You would not be able to assess probabilistically treatment modalities. Ultimately, incorporating the method proposed in this dissertation could benefit both of the domains of public health practice and the practice of evidence based medicine.

One example of our proposed method being used in the area of medicine and public health would be in the planning of a new study. When, in the initial plan stages of a study, an individual might not have any idea when an appropriate time to look at certain markers, this method could be used to help the planner estimate when to look at the outcome data to best determine the utility of specific markers and what at value(s) of the marker should the "split point(s)" occur to obtain optimal discrimination with respect to outcome. These split points by both time and marker value could then become the baselines used for the new study.

Our proposed method would also be beneficial for public health applications in that it could potentially facilitate a more refined survival analysis. Since we are obtaining both the optimal time point and the optimal split point, it can also allows us to view outcome data at the time the markers are being discriminated the greatest. Split points are used to identify dichotomous markers that optimally distinguish high risk from low risk patients or participants. As we have seen, the effects of certain markers or other predictors attenuate over time. By knowing at what time to look at the data, we could potentially gain insight on the underlying biological properties of the markers.

In summary, this dissertation developed and validated a method to find the best point of discrimination of covariate with respect to both time and split point that could be adapted to any type of recursive software. This method could be generalized to molecular markers and could be used for any covariate as part of an exploratory analysis.

7.2 FUTURE DIRECTION

It is important statistically and for the sake of public health applications that proposed methods are adequately assessed and validated. We attempted to validate our method by using a standard statistical methodology to examine the results. We used a power analysis to examine how our method acts for the null hypothesis and how it reacts in the grey area, where the parameters are slightly different. Other options are available to further validate this method. One option is to follow the simple counting algorithm that was used by Wernecke, et al. (1998). Their method performs validation steps to determine a measure of stability for the underlying data. Large differences in the generated error rates point to unstable feature sets. If certain features show up repeatedly in the validation steps they can be considered stable. Using their method would allow us to get a valuation of the stability of the tree construction and give us confidence that our tree could be applied in a public health setting. Another option is to follow the methods that were developed by Stein, et al. (2001). They developed two algorithms that were an extension of the resampling-method suggested by Wernecke, et al. for the validation of classification trees. Future projects could use either of these validation techniques.

Since this dissertation only covered the case of time-varying effects (coefficients), a future project could be to develop an extension that accommodates both time-varying effects and time-varying covariates. This extension would allow the optimal split to be found when not only the effect of the marker might potentially change over time but the marker itself potentially changes over time.

APPENDIX 1

ADDITIONAL PROGRAMMING DETAILS

S-PLUS and RPART

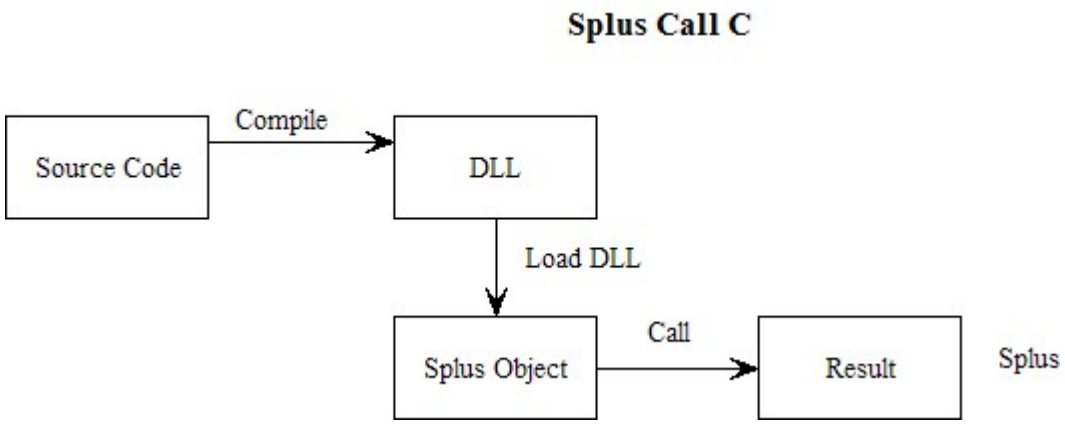
To be able to make the required changes, we needed to have a better understanding of how the RPART package, in conjunction with S-PLUS, actually works. S-PLUS is available as a commercial package from Insightful and is an implementation of the language S developed at Bell Laboratories by Becker, Chamberlain and Wilks. A powerful feature of S-PLUS is that is allows the user to extend its functionality, which enables the user to interface with other languages, namely, C and Fortran. By using other languages, it allows the user to combine the speed and efficiency of compiled code with the robust, flexible programming environment of S-PLUS. While S is an expansive language with a large number of routines already included, there are procedures not covered. Fortunately, the core routines are easily augmented with additional user-written routines, which can be loaded into S-PLUS. These routines are usually provided in what S-PLUS calls a "library". S-PLUS libraries are a convenient way to package up user-created functionality in order to share it with other users (*S-PLUS 2000 Programmer's Guide*).

The RPART package is written in C and implements classification and regression trees defined by Breiman, et al. (1984) and Therneau and Atkinson (1997). RPART is called from within S-PLUS and returns a standard S-PLUS tree object that can then be manipulated using S-PLUS visualization and statistical functions. Therefore, the RPART package contains both C and S code. The purpose of the C code is to improve on the cpu time required for lengthy computations that are performed during the recursive partitioning process.

Typically to use RPART, the user will load the recursive partitioning package into S-PLUS using the *library()* function. At this point, recursive partitioning can be performed using the *rpart()* function. However, since we are interested in determining an optimal time and an optimal split points we need to modify the code within the RPART package.

To modify the code within RPART, the C code will need to be compiled so that a S-PLUS function can call this routine, once it is loaded. To do this a dynamic-link library (dll) will be created to export functions for S-PLUS to use. A dynamic-link library is a binary code file that allows any program that loads it to use the functions it exports. These functions will be written in C++ and compiled with another file (the definition file) that tells the compiler which of the functions it is exporting. At this point, the dll can be load in Splus and the function can be called. This process is displayed in Figure A1.1.

Figure A1.1: S-PLUS Call Process

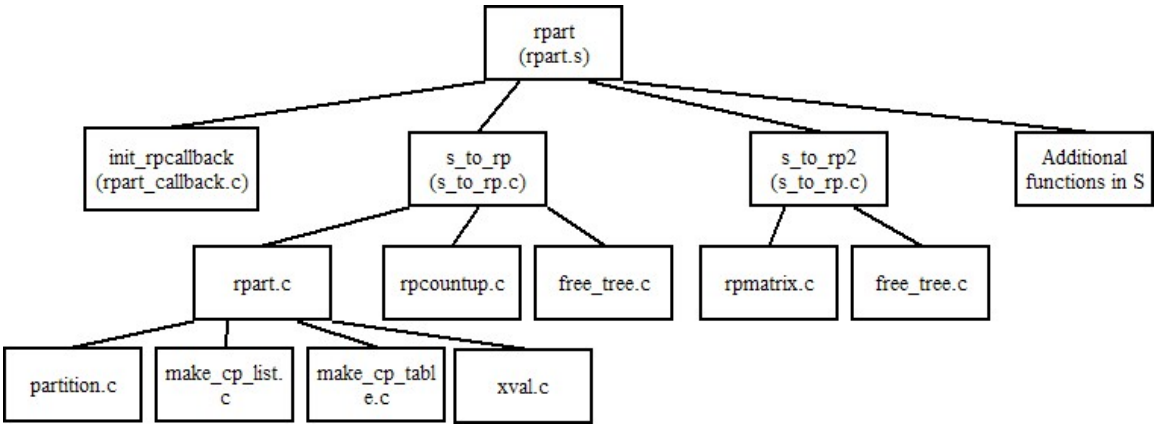


How RPART Works

Now that we understand how the RPART is called, we need to know how the RPART algorithm steps through all of the routines included in the source code. A flow diagram of how

RPART progresses in displayed in Figure A1.2. The following explanations are mainly from the documentation located in the source code of Rpart.

Figure A1.2: Rpart Flow Chart



When the *rpart()* function is called in S-PLUS, S-PLUS starts with the routine in *rpart.s*. The *rpart.s* routine is the recursive partitioning function for S. The first function called in *rpart.s* is *init_rpcallback*. This routine saves the parameters, the location of the evaluation frame and the 2 expressions to be computed within it, and ferrets out the memory location of the 4 "callback" objects). Next, *Rpart.s* calls two functions from *s_to_rp.c*, which is an S interface to the recursive partitioning routines. The first routine in *s_to_rp.c* is *s_to_rp* (calls RPART, count up the number of nodes, splits, categorical splits, and cp's). *Rpart.c* is the main entry point for recursive partitioning routines. It first initializes the splitting functions from the function table. At this point, it is determined which method (anova, exponential, poisson or classification) will be used. Next, *rpart.c* performs the basic tree. To do this, the partition function (located in *partition.c*) is called.

The partition function is the main workhorse of the recursive partitioning module. When called with a node, it partitions it and then calls itself to partition the children it has created. If the node is not able to be split (if there are too few people or if the complexity is too small) it simply returns. The routine may not be able to discover that the complexity is too small until after the children have been partitioned, so it checks if the complexity is too small at the end. The next function that is called by *rpart.c* is *make_cp_list* (located in *make_cp_list.c*). The *make_cp_list* function creates the list of unique complexity parameters. The list is maintained in sorted order. If two parameters are within "*cplist_epsilon*" of one another, then only the larger of them is retained. After the partition routine is done, each node is labeled with the complexity parameter appropriate if that node were the top of the tree. However, if there is a more vulnerable node further up, the node in question will actually have the smaller complexity parameter; it will be removed when its parent collapses. So this routine also adjusts each C.P. to equal the minimum(my C.P., parent's C.P.). This routine is called at the top level by RPART, after RPART has initialized the first member of the linked cp-list, set its number of splits to zero, and its risk to that for no splits at all. This routine allocates and links in the rest of the cp-list. The *make_cp_table* routine then fills in the rest of the variables in the list. When it comes time to cross-validate, we fill in xrisk (cross-validated risk estimate) and xstd (standard deviation of xrisk).

Next, the *make_cp_table* function (located in *make_cp_table.c*) is called by *rpart.c*. Since a cptable list is already initialized with the unique cp's in it, the columns for risk and number of splits are filled. For each terminal node on the tree, it starts down at the bottom of the list of complexity parameters. For each unique C.P. until my parent collapses, the node I'm in adds into

that line of the CP table. So walk up the CP list, adding in, until my parent would collapse; then report my position in the cp list to the parent and quit.

The last function that *rpart.c* calls is *xval* (located in *xval.c*). The *xval* function is responsible for the cross validation of a model. This routine is responsible for filling in two vectors -- xrisk and xstd. A stratified partitioning of the data (NOT random) is used to divide the data into n_xval subgroups. One by one, each of these groups is left out of the partitioning. After partitioning, the risk of each left out subject is determined, under each of the unique complexity parameters. The x-groups are set by the calling S-routine, so they can actually be random, non-random, or whatever, as far as this routine is concerned.

After the recursive partitioning is finished, we return to *s_to_rp*. Now, the *rpcountup* function (located in *rpcountup.c*) is called. The *rpcountup* function counts up the number of nodes and splits in the final result. It does this by gathering up the counts for a node, adds in those of from the nodes' children, and passes the total back to the nodes parent. The last function call made by *s_to_rp* is made to *free_tree* (located in *free_tree.c*). The *free_tree* function frees up all of the memory associated with a tree.

Now that the trees have been calculated, the process returns back to *rpart.s*. The next function that is call is *s_to_rp2*. This function retrieves the complexity table and then gets all of the information for the tree. *S_to_rp2* calls the rpmatrix function (located in *rpmatrix.c*) that converts the linked list data into matrix form for S-PLUS. Now, the details of the trees can be displayed in S-PLUS.

Changes made to Rpart Code

In order to make RPART determine an optimal time and an optimal split points, changes had to be made to the source code. The changes made to both *rpart.s* and *rpart.control.s* and are included in the Appendix 2. The goal was to create an extension to RPART that will allow the user to create trees for each unique event time so that an optimal time and optimal split points can be determined. But since constructing trees for each unique event time can be very time consuming for large datasets, an alternative method was also included. This alternative method constructs trees for each year. For example, trees are constructed at 1-year survival, 2-year survival, 3-year survival, etc.

To allow the user to select which method they want to use, another parameter (newmethod) was added to the *rpart.control* function (located is *rpart.control.s*). Adding the newmethod parameter allows the user to control which method is used in constructing the tree. The newmethod parameter can be used by including the *control=rpart.control(newmethod=3)* into the RPART function.

```
rpart(formula, data, weights, subset, na.action = na.rpart, method,
        model = FALSE, x = FALSE, y = TRUE, parms, control =
        rpart.control(newmethod=3), cost, ...)
```

If the user selects newmethod=1 then RPART will calculate the tree using the default method. The new code is ignored. If the user selects newmethod=2, the dataset examined to determine how many unique time points are in the dataset. Then, a tree for each unique time

point will be calculated. If the user selects newmethod=3, then the time points are converted to years. Therefore, trees will be calculated for each year.

To incorporate the addition of the newmethod parameter, most of the changes were made in *Rpart.s*. The first thing that needed to be added to the code was a loop statement so that the tree creation process could be done over and over. The number of times that the process would go through the loop statement would depend on which value for newmethod was selected. However, before the loop statement could be accessed, a couple of things need to be initialized. To have these things initialized before the loop statement, two sections of code needed to be duplicated. The first duplicated section is code to initialize *y*. This needed to be done so that *y* can be examined to determine how many unique event times there are or to determine how many years need to be used. The second section is to allow access to rpart.control so that the user is able to select which method that will be used.

Now that the user has initialized the newmethod parameter, code is needed to perform each of the options. The first option is to construct the trees by using the default way. This option ignores the loop statement so that RPART acts like it was never modified. The second option is to calculate trees for each of the unique event times. All of the event time points are sorted uniquely so that there is only one record for each of the different event times. A tree will be constructed for each of the unique event times. For example, there could be a tree constructed at 46.8, 51.2, 55.3, etc. months. The third and final option is to calculate trees for each year. The maximum time point is calculated to determine the last year that will be used. Then, a tree will be calculated for each year until the maximum year is reached. For example, we could have a tree for 1-year, 2-year, 3-year, out to 20-year survival.

At this point of the code, the process returns to the original code to construct the trees. After the trees are constructed code needed to be added to create the cp and split tables for each of the 3 methods. Time points had to be added to the output so that the user would know when the optimal time occurred.

The original code only created one tree, so only the information from one tree was ever recorded. Code was added to record the information from two trees. So as the process stepped through the different time points, the information for the best tree was kept. If the tree from another time point was better, the information for the previous best tree is moved to the second best tree spot while the information from this tree will be recorded as the best tree. Keeping the information for the trees at the two best time points allows us to compare the two trees.

APPENDIX 2

CODE

This appendix shows the code that was used to extend the algorithm used in RPART (developed by Therneau and Atkinson and ported to windows by B. D. Ripley). The original RPART software can be at the following internet address:

http://www.stats.ox.ac.uk/pub/MASS3/Winlibs/

```
Initial Date: 4/15/1997
Update#1: 2/23/1998
Update#2: 2/25/2000
Update#3: 2/15/2002

This is the third release of the rpart package for Splus.  This represents a
major revision of the rpart code, driven by the desire to add user-written
split routines.

Terry M Therneau
Beth Atkinson
Mayo Clinic

Copyright 2002 Mayo Foundation for Medical Education and Research. This
software is accepted by users "as is" and without warranties or guarantees of
any kind.  It may be used for research purposes or in relation to projects
with commercial applications or included in commercial packages, but only so
long as it is not relicensed as a stand-alone program, and only so long as
the first two sentences of this paragraph (copyright notice and no warranty)
are reproduced with the software.
Problems/comments/suggestions should be reported to atkinson@mayo.edu



# SCCS  @(#)rpart.s 1.37 03/14/02
#
#  The recursive partitioning function, for S
#
rpart <- function(formula, data=sys.parent(), weights, subset,
          na.action=na.rpart, method, model=F, x=F, y=T,
          parms, control, cost, ...) {


   #Coded added to initialize y before the loop statement         # Added by LSD
   call <- match.call()                                          # Added by LSD
   if (is.data.frame(model)) {                                   # Added by LSD
       m <- model                                                # Added by LSD
       model <- F                                                # Added by LSD
   }                                                             # Added by LSD
```

```
else {                                                          # Added by LSD
   m <- match.call(expand=F)                                    # Added by LSD
   m$model <- m$method <- m$control<- NULL                      # Added by LSD
   m$x <- m$y <- m$parms <- m$... <- NULL                       # Added by LSD
   m$cost <- NULL                                               # Added by LSD
   m$na.action <- na.action                                     # Added by LSD
   m[[1]] <- as.name("model.frame.default")                     # Added by LSD
   m <- eval(m, sys.parent())                                   # Added by LSD
}                                                               # Added by LSD
 Terms <- attr(m, "terms")                                      # Added by LSD
 if(any(attr(Terms, "order") > 1))                              # Added by LSD
stop("Trees cannot handle interaction terms")                   # Added by LSD

 Y <- model.extract(m, "response")                              # Added by LSD

# Code add to access rpart.control before the loop
 extraArgs <- list(...)                                         # Added by LSD
 if (length(extraArgs)) {                                       # Added by LSD
   controlargs <- names(args(rpart.control))  #legal arg names  # Added by LSD
   indx <- match(names(extraArgs), controlargs, nomatch=0)      # Added by LSD
   if (any(indx==0))                                            # Added by LSD
   stop(paste("Argument", names(extraArgs)[indx==0],            # Added by LSD
          "not matched"))                                       # Added by LSD
}                                                               # Added by LSD
 controls <- rpart.control(...)                                 # Added by LSD
 if (!missing(control)) controls[names(control)] <- control     # Added by LSD


# do tree using default                                         # Added by LSD
if (controls$newmethod==1) { nobstime <- 1                      # Added by LSD
                temptime <- matrix(Y[,1],,1)                    # Added by LSD
}                                                               # Added by LSD

#do tree using event time                                       # Added by LSD
 if (controls$newmethod==2) {                                   # Added by LSD
   Ytemp <- Y                                                   # Added by LSD
   Ynobs <- length(Ytemp[,1])                                   # Added by LSD
   for (eventobs in 1:Ynobs){                                   # Added by LSD
      if (Ytemp[eventobs,2]==0) {                               # Added by LSD
          Ytemp[eventobs,1]<- NA                                # Added by LSD
          Ytemp[eventobs,2]<- NA}                               # Added by LSD
   }                                                            # Added by LSD
   Ytimetemp <- matrix(Ytemp[,1])                               # Added by LSD
   Ytime <- round(Ytimetemp,3)                                  # Added by LSD
   Ytime <- sort(unique(Ytime))                                 # Added by LSD
   nobstime <- length(Ytime)                                    # Added by LSD
   temptime <- matrix(Ytime,nobstime,1)                         # Added by LSD
}                                                               # Added by LSD

# do trees by using years                                       # Added by LSD
if (controls$newmethod==3) {                                    # Added by LSD
   Ytime_matrix(Y[,1])                                          # Added by LSD
   Ytime_sort(unique(Ytime))                                    # Added by LSD
   nobstime <- length(Ytime)                                    # Added by LSD
   maxtime<-round(Ytime[nobstime]/12,0)                         # Added by LSD
   if (maxtime > 20) maxtime<-20                               # Added by LSD
   nobstime <- maxtime                                          # Added by LSD
   monthtime<-matrix(1:maxtime)                                 # Added by LSD
   temptime<-monthtime*12                                       # Added by LSD
}                                                               # Added by LSD
```

```
# starts the loop for each tree                                    # Added by LSD
for (itime in 1:nobstime){                                         # Added by LSD
   if (itime==1) {tempimprove<-0                                   # Added by LSD
      tempimprove2<-0                                              # Added by LSD
      tempans2<-NULL                                               # Added by LSD
                tempans<-NULL                                      # Added by LSD
   }                                                               # Added by LSD


#back to the code                                                  # Added by LSD
   call <- match.call()
   if (is.data.frame(model)) {
      m <- model
      model <- F
   }
   else {
      m <- match.call(expand=F)
      m$model <- m$method <- m$control<- NULL
      m$x <- m$y <- m$parms <- m$... <- NULL
      m$cost <- NULL
      m$na.action <- na.action
      m[[1]] <- as.name("model.frame.default")
      m <- eval(m, sys.parent())
   }
   Terms <- attr(m, "terms")
   if(any(attr(Terms, "order") > 1))
   stop("Trees cannot handle interaction terms")

   Y <- model.extract(m, "response")


   wt <- model.extract(m, "weights")
   if(length(wt)==0) wt <- rep(1.0, nrow(m))
   offset <- attr(Terms, "offset")
   X <- rpart.matrix(m)
   nobs <- nrow(X)
   nvar <- ncol(X)


   for (i in 1:nobs) {
      if (Y[i,1]> temptime[itime,1]) {           # Added by LSD
         Y[i,1] <- temptime[itime,1]             # Added by LSD
         Y[i,2] <- 0                             # Added by LSD
      }
   }                                             # Added by LSD


   if (missing(method)) {
      if (is.factor(Y) || is.character(Y))      method <- 'class'
      else if (is.Surv(Y))   method <- 'exp'
      else if (is.matrix(Y)) method<- 'poisson'
      else                   method<- 'anova'
   }

   if (is.list(method)) {
      # User written split methods
      mlist <- method
      method <- 'user'
      if (!is.list(mlist) || length(mlist) !=3)
      stop("User written methods must have 3 functions")
      if (is.null(mlist$init) || class(mlist$init) != 'function')
      stop("User written method does not contain an init function")
```

```
if (missing(parms)) init <- mlist$init(Y, offset, wt=wt)
else                init <- mlist$init(Y, offset, parms, wt)

method.int <- 4      #the fourth entry in func_table.h
if (is.null(mlist$split) || class(mlist$split) != 'function')
stop("User written method does not contain a split function")
if (is.null(mlist$eval) || class(mlist$eval) != 'function')
stop("User written method does not contain an eval function")

user.eval <- mlist$eval
user.split <- mlist$split

numresp <- init$numresp
numy <-  init$numy
parms <- init$parms

#
# expr2 is an expression that will call the user "evaluation"
#   function, and check that what comes back is valid
# expr1 does the same for the user "split" function
#
# For speed in the C interface, yback, xback, and wback are
#  fixed S vectors of a fixed size, and nback tells us how
#  much of the vector is actually being used on this particular
#  callback.
#
if (numy==1) {
   expr2 <- Quote({
      temp <- user.eval(yback[1:nback], wback[1:nback], parms)
      if (length(temp$label) != numresp)
      stop("User eval function returned invalid label")
      if (length(temp$deviance) !=1)
      stop("User eval function returned invalid deviance")
      as.numeric(as.vector(c(temp$deviance, temp$label)))
   })
   expr1 <- Quote({
      if (nback <0) { #categorical variable
         n2 <- -1*nback
         temp  <- user.split(yback[1:n2], wback[1:n2],
         xback[1:n2], parms, F)
         ncat <- length(unique(xback[1:n2]))
         if (length(temp$goodness) != ncat-1 ||
         length(temp$direction) != ncat)
         stop("Invalid return from categorical split fcn")
      }

      else {
         temp <- user.split(yback[1:nback], wback[1:nback],
          xback[1:nback], parms, T)
         if (length(temp$goodness) != (nback-1))
         stop("User split function returned invalid goodness")
         if (length(temp$direction) != (nback-1))
         stop("User split function returned invalid direction")
      }
      as.numeric(as.vector(c(temp$goodness, temp$direction)))
   })
}
else {
   expr2 <- Quote({
      tempy <- matrix(yback[1:(nback*numy)], ncol=numy)
      temp <- user.eval(tempy, wback[1:nback], parms)
      if (length(temp$label) != numresp)
      stop("User eval function returned invalid label")
```

```
            if (length(temp$deviance) !=1)
            stop("User eval function returned invalid deviance")
            as.numeric(as.vector(c(temp$deviance, temp$label)))
         })
      expr1 <- Quote({
         if (nback <0) { #categorical variable
            n2 <- -1*nback
            tempy <- matrix(yback[1:(n2*numy)], ncol=numy)
            temp  <- user.split(tempy, wback[1:n2], xback[1:n2],
            parms, F)
            ncat <- length(unique(xback[1:n2]))
            if (length(temp$goodness) != ncat-1 ||
            length(temp$direction) != ncat)
            stop("Invalid return from categorical split fcn")
         }
         else {
            tempy <- matrix(yback[1:(nback*numy)], ncol=numy)
            temp <- user.split(tempy, wback[1:nback],xback[1:nback],
                parms, T)
            if (length(temp$goodness) != (nback-1))
            stop("User split function returned invalid goodness")
            if (length(temp$direction) != (nback-1))
            stop("User split function returned invalid direction")
         }
         as.numeric(as.vector(c(temp$goodness, temp$direction)))
      })
    }

   #
   # The vectors nback, wback, xback and yback will have their
   #  contents constantly re-inserted by C code.  It's one way to make
   #  things very fast.  It is dangerous to do this, so they
   #  are tossed into a separate frame to isolate them.  Evaluations of
   #  the above expressions occur in that frame.
   #
   eframe <- new.frame(list(nback = integer(1),
           wback = double(nobs),
           xback = double(nobs),
           yback = double(nobs*numy),
           user.eval =  user.eval,
           user.split = user.split,
           numy = numy,
           numresp = numresp,
           parms = parms), protect=T)
   .Call("init_rpcallback", eframe, as.integer(numy),
                        as.integer(numresp),
                        expr1, expr2)
}
 else {
   method.int <- pmatch(method, c("anova", "poisson", "class", "exp"))
   if (is.na(method.int)) stop("Invalid method")
   method <- c("anova", "poisson", "class", "exp")[method.int]
   if (method.int==4) method.int <- 2

   if (missing(parms))
     init <- (get(paste("rpart", method, sep='.')))(Y,offset, ,wt)
   else
     init <- (get(paste("rpart", method, sep='.')))(Y,offset, parms, wt)
}

 Y <- init$y

 xlevels <- attr(X, "column.levels")
```

```
 cats <- rep(0,ncol(X))
 if(!is.null(xlevels)) {
    cats[match(names(xlevels), dimnames(X)[[2]])] <-
      unlist(lapply(xlevels, length))
}

 # We want to pass any ... args to rpart.control, but not pass things
 #  like "dats=mydata" where someone just made a typo.  The use of ...
 #  is just to allow things like "cp=.05" with easier typing
 extraArgs <- list(...)
 if (length(extraArgs)) {
    controlargs <- names(args(rpart.control))  #legal arg names
    indx <- match(names(extraArgs), controlargs, nomatch=0)
    if (any(indx==0))
        stop(paste("Argument", names(extraArgs)[indx==0],
            "not matched"))
}

 controls <- rpart.control(...)
 if (!missing(control)) controls[names(control)] <- control

 xval <- controls$xval
 if (is.null(xval) || (length(xval)==1 && xval==0) || method=='user') {
    xgroups <-0
    xval <- 0
}
 else if (length(xval)==1) {
    # make random groups
     xgroups <- sample(rep(1:xval, length=nobs), nobs, replace=F)
}
 else if (length(xval) == nobs) {
    xgroups <- xval
    xval <- length(unique(xgroups))
}
 else {
    # Check to see if observations were removed due to missing
    if (!is.null(attr(m, 'na.action'))) {
        # if na.rpart was used, then na.action will be a vector
        temp <- as.integer(attr(m, 'na.action'))
        xval <- xval[-temp]
        if (length(xval) == nobs) {
            xgroups <- xval
            xval <- length(unique(xgroups))
        }
        else stop("Wrong length for xval")
     }
    else stop("Wrong length for xval")
}

 #
 # Incorporate costs
 #
 if (missing(cost)) cost <- rep(1.0, nvar)
 else {
    if (length(cost) != nvar)
        stop("Cost vector is the wrong length")
    if (any(cost <=0)) stop("Cost vector must be positive")
}

 #
 # Have s_to_rp consider ordered categories as continuous
 #  A right-hand side variable that is a matrix forms a special case
 # for the code.
```

```
#
tfun <- function(x) {
   if (is.matrix(x)) rep(is.ordered(x), ncol(x))
   else is.ordered(x)
}
isord <- unlist(lapply(m[attr(Terms, 'term.labels')], tfun))
rpfit <- .C("s_to_rp",
       n = as.integer(nobs),
       nvarx = as.integer(nvar),
       ncat = as.integer(cats* !isord),
       method= as.integer(method.int),
       as.double(unlist(controls)),
       parms = as.double(unlist(init$parms)),
       as.integer(xval),
       as.integer(xgroups),
       as.double(t(init$y)),
       as.single(X),
       as.integer(is.na(X)),
       error = character(1),
       wt = as.double(wt),
       as.integer(init$numy),
       as.double(cost),
       NAOK=T )
if (rpfit$n == -1)  stop(rpfit$error)

# rpfit$newX[1:n] contains the final sorted order of the observations
nodes <- rpfit$n          # total number of nodes
nsplit<- rpfit$nvarx      # total number of splits, primary and surrogate
numcp <- rpfit$method     # number of lines in cp table
ncat  <- rpfit$ncat[1]    # total number of categorical splits
numresp<- init$numresp    # length of the response vector

if (nsplit ==0) xval <-0  # No xvals were done if no splits were found
cpcol <- if (xval>0) 5 else 3
if (ncat==0) catmat <- 0
else         catmat <- matrix(integer(1), ncat, max(cats))

rp    <- .C("s_to_rp2",
          as.integer(nobs),
          as.integer(nsplit),
          as.integer(nodes),
          as.integer(ncat),
          as.integer(cats *!isord),
          as.integer(max(cats)),
          as.integer(xval),
          which = integer(nobs),
          cptable = matrix(double(numcp*cpcol), nrow=cpcol),
          dsplit =  matrix(double(1),  nsplit,3),
          isplit =  matrix(integer(1), nsplit,3),
          csplit =  catmat,
          dnode  =  matrix(double(1),  nodes, 3+numresp),
          inode  =  matrix(integer(1), nodes, 6))
tname <- c("<leaf>", dimnames(X)[[2]])

if (controls$newmethod==2) tempyear <- matrix(temptime[itime,1],1,numcp)
         #added by LSD
if (controls$newmethod==3) tempyear <- matrix(itime,1,numcp)
         #added by LSD

if (controls$newmethod==1) {
       #added by LSD
   if (cpcol==3) temp <- c("CP", "nsplit", "rel error")
   else          temp <- c("CP", "nsplit", "rel error", "xerror", "xstd")}
```

```
if (controls$newmethod==2) {                                        #added by LSD
   if (cpcol==3) temp <- c("CP", "nsplit", "rel error","month")
      #month added by LSD
   else          temp <- c("CP", "nsplit", "rel error", "xerror", "xstd","month")}
      #month added by LSD

if (controls$newmethod==3) {                        #added by LSD
   if (cpcol==3) temp <- c("CP", "nsplit", "rel error","year")  #year added by LSD
   else          temp <- c("CP", "nsplit", "rel error", "xerror", "xstd","year") }
      #year added by LSD

if (controls$newmethod>1) rp$cptable <- rbind(rp$cptable, tempyear)   #added by LSD
dimnames(rp$cptable) <- list(temp, 1:numcp)

splits<- matrix(c(rp$isplit[,2:3], rp$dsplit), ncol=5,
        dimnames=list(tname[rp$isplit[,1]+1],
         c("count", "ncat", "improve", "index", "adj")))
if(controls$newmethod==2) splits <- cbind(splits,temptime[itime,1])
if(controls$newmethod==3) splits <- cbind(splits,itime)
if (controls$newmethod==1) dimnames(splits)<-list(NULL,c("count", "ncat",
"improve", "index", "adj"))          # added by LSD
if (controls$newmethod==2) dimnames(splits)<-list(NULL,c("count", "ncat",
"improve", "index", "adj", "month"))    # added by LSD
if (controls$newmethod==3) dimnames(splits)<-list(NULL,c("count", "ncat",
"improve", "index", "adj", "year"))  # added by LSD
index <- rp$inode[,2]  #points to the first split for each node

 # Now, make ordered categories look like categories again (a printout
 #  choice)
 nadd <- sum(isord[rp$isplit[,1]])
 if (nadd >0) {
    newc <- matrix(integer(1), nadd, max(cats))
    cvar <- rp$isplit[,1]
    indx <- isord[cvar]          # vector of T/F
    cdir <- splits[indx,2]               # which direction splits went
    ccut <- floor(splits[indx,4])        # cut point
    splits[indx,2] <- cats[cvar[indx]]   #Now, # of categories instead
    splits[indx,4] <- ncat + 1:nadd      # rows to contain the splits

    # Next 4 lines can be done without a loop, but become indecipherable
    for (i in 1:nadd) {
       newc[i, 1:(cats[(cvar[indx])[i]])] <- -1*as.integer(cdir[i])
       newc[i, 1:ccut[i]] <- as.integer(cdir[i])
     }
    if (ncat==0) catmat <- newc
    else         catmat <- rbind(rp$csplit, newc)
    ncat <- ncat + nadd
}
 else catmat <- rp$csplit

if (nsplit==0) {  #tree with no splits
    frame <- data.frame(row.names=1,
           var=  "<leaf>",
           n =   rp$inode[,5],
           wt=   rp$dnode[,3],
           dev=  rp$dnode[,1],
           yval= rp$dnode[,4],
           complexity=rp$dnode[,2],
           ncompete  = pmax(0, rp$inode[,3]-1),
           nsurrogate=rp$inode[,4],                              #comma added by LSD
            year = itime)                                        # Added by LSD
}
```

```
else {
    temp <- ifelse(index==0, 1, index)
    svar <- ifelse(index==0, 0, rp$isplit[temp,1]) #var number
    frame <- data.frame(row.names=rp$inode[,1],
           var=  factor(svar, 0:ncol(X), tname),
           n =   rp$inode[,5],
           wt=   rp$dnode[,3],
           dev=  rp$dnode[,1],
           yval= rp$dnode[,4],
           complexity=rp$dnode[,2],
           ncompete  = pmax(0, rp$inode[,3]-1),
           nsurrogate=rp$inode[,4],                          #comma added by LSD
            year = itime)                                       # Added by LSD
}
 if (method.int ==3 ) {
    # Create the class probability vector from the class counts, and
    #   add it to the results
    # The "pmax" one line down is for the case of a factor y which has
    #   no one at all in one of its classes.  Both the prior and the
    #   count will be zero, which led to a 0/0.
    numclass <- init$numresp -1
    temp <- rp$dnode[,-(1:4)] %*% diag(init$parms$prior*
                sum(init$counts)/pmax(1,init$counts))
    yprob <- temp /apply(temp,1,sum)   #necessary with altered priors
    yval2 <- matrix(rp$dnode[, -(1:3)], ncol=numclass+1)
   frame$yval2 <- cbind(yval2, yprob)
}
 else if (init$numresp >1) frame$yval2 <- rp$dnode[,-(1:3)]

 if (is.null(init$summary))
    stop("Initialization routine is missing the summary function")
 if (is.null(init$print))
    functions <- list(summary=init$summary)
 else    functions <- list(summary=init$summary, print=init$print)
 if (!is.null(init$text)) functions <- c(functions, list(text=init$text))
 if (method=='user')functions <- c(functions, mlist)

 where <- rp$which
 names(where) <- row.names(m)

if (nsplit ==0) {  # no 'splits' component
    ans <- list(frame = frame,
        where = where,
        call=call, terms=Terms,
        cptable =  t(rp$cptable),
        method = method,
        parms  = init$parms,
        control= controls,
        functions= functions)
}
else {
    ans <- list(frame = frame,
        where = where,
        call=call, terms=Terms,
        cptable =  t(rp$cptable),
        splits = splits,
        method = method,
        parms  = init$parms,
        control= controls,
        functions= functions)
}

if (ncat>0) ans$csplit <- catmat +2
```

```
    if (model) {
        ans$model <- m
        if (missing(y)) y <- F
    }
    if (y) ans$y <- Y
    if (x) {
        ans$x <- X
        ans$wt<- wt
    }
     ans$ordered <- isord
     if(!is.null(attr(m, "na.action")))
         ans$na.action <- attr(m, "na.action")
     if (!is.null(xlevels)) attr(ans, 'xlevels') <- xlevels
     if(method=='class') attr(ans, "ylevels") <- init$ylevels
#    if (length(xgroups)) ans$xgroups <- xgroups
    oldClass(ans) <- c("rpart")
    if (itime==1 && splits[1,3]==0 && tempimprove==0) {               # Added by LSD
           tempans <- ans                                            # Added by LSD
           tempans2<-ans                                             # Added by LSD
    }                                                                # Added by LSD
    if (splits[1,3]>tempimprove) {                                   # added by LSD
        if (itime>1) {                                               # added by LSD
           tempimprove2<-tempimprove                                 # added by LSD
           tempans2<-tempans}                                        # added by LSD
           tempimprove<-splits[1,3]                                  # added by LSD
           tempans <- ans                                          # added by LSD
        }                                                            # added by LSD
        else if (splits[1,3]>tempimprove2) {                         # added by LSD
           tempimprove2<-splits[1,3]                                 # added by LSD
           tempans2 <- ans                                           # added by LSD
        }                                                            # added by LSD
    }
#   ans<-tempans                                                  # commented out by LSD
    if (tempimprove>0){                                              # Added by LSD
        if (tempimprove2==0) {tempans2$splits <- cbind(0,0,0,0,0,0)}   # Added by LSD
           answer <- list(frame = tempans$frame,                     # added by LSD
            where = tempans$where,                                   # added by LSD
            call=tempans$call, terms=tempans$Terms,                  # added by LSD
            cptable =  tempans$cptable,                              # added by LSD
           splits = tempans$splits,                                  # added by LSD
            method = tempans$method,                                 # added by LSD
            parms  = tempans$parms,                                  # added by LSD
            control= tempans$control,                                # added by LSD
            functions= tempans$functions,                            # added by LSD
            frame2 = tempans2$frame,                                 # added by LSD
           where2 = tempans2$where,                                  # added by LSD
           call2=tempans2$call, terms2=tempans2$Terms,               # added by LSD
           cptable2 =  tempans2$cptable,                             # added by LSD
            splits2 = tempans2$splits,                               # added by LSD
           method2 = tempans2$method,                                # added by LSD
           parms2  = tempans2$parms,                                 # added by LSD
           control2= tempans2$control,                               # added by LSD
           functions2= tempans2$functions)                           # added by LSD

        answer                                                       # added by LSD
    }
    else if (tempimprove==0) {                                       # Added by LSD
        tempans$splits <- cbind(0,0,0,0,0,0)                         # Added by LSD
        tempans2$splits <- cbind(0,0,0,0,0,0)                        # Added by LSD
      answer<- list(frame = tempans$frame,                           # added by LSD
           where =tempans$where,                                     # added by LSD
           call=tempans$call, terms=tempans$Terms,                   # added by LSD
           cptable =  tempans$cptable,                               # added by LSD
```

```
          splits = tempans$splits,                                    # Added by LSD
          method = tempans$method,                                    # added by LSD
          parms  = tempans$parms,                                     # added by LSD
          control= tempans$control,                                   # added by LSD
          functions= tempans$functions                                # added by LSD
           frame2 = tempans2$frame,                                   # added by LSD
          where2 = tempans2$where,                                    # added by LSD
          call2=tempans2$call, terms2=tempans2$Terms,                 # added by LSD
          cptable2 =  tempans2$cptable,                               # added by LSD
           splits2 = tempans2$splits,                                 # added by LSD
          method2 = tempans2$method,                                  # added by LSD
          parms2  = tempans2$parms,                                   # added by LSD
          control2= tempans2$control,                                 # added by LSD
          functions2= tempans2$functions)                             # Added by LSD

   answer}                                                            # Added by LSD
}

# end of rpart.s LSD



#SCCS @(#)rpart.control.s  1.10 07/05/01
rpart.control <-
  function(minsplit=20, minbucket= round(minsplit/3), cp=.01,
      maxcompete=4, maxsurrogate=5, usesurrogate=2, xval=10,
      surrogatestyle =0, maxdepth=30, newmethod=1){ # added newmethod by LSD

   if (maxcompete<0) {
       warning("The value of maxcompete supplied was <0;",
          "the value 0 was used instead")
       maxcompete <-0
       }
   if (any(xval<0)) {
       warning("The value of xval supplied was <0;",
          "the value 0 was used instead")
       xval <-0
       }
   if (maxdepth > 30) stop("Maximum depth is 30")
   if (maxdepth < 1)  stop("Maximum depth must be at least 1")

   if (missing(minsplit) && !missing(minbucket)) minsplit <- minbucket*3

   if((usesurrogate < 0) || (usesurrogate > 2)) {
       warning("The value of usesurrogate supplied was out of range," ,
          "the default value of 2 is used instead.")
       usesurrogate <- 2
       }
   if((surrogatestyle < 0) || (surrogatestyle > 1)) {
       warning("The value of surrogatestyle supplied was out of range,",
          "the default value of 0 is used instead.")
       surrogatestyle <- 0
       }

   if((newmethod < 1) || (newmethod > 3)) {
       warning("The value of newmethod supplied was out of range," ,
          "the default value of 1 is used instead.")
       newmethod <- 1
       }                                                  #added by LSD

   # Because xval can be of length either 1 or n, and the C code
   #   refers to parameters by number, i.e., "opt[5]" in rpart.c,
   #   the xval parameter should always be last on the list.
```

```
    list(minsplit=minsplit, minbucket=minbucket, cp=cp,
         maxcompete=maxcompete, maxsurrogate=maxsurrogate,
         usesurrogate=usesurrogate,
         surrogatestyle=surrogatestyle, maxdepth=maxdepth, xval=xval,
    newmethod=newmethod )  #newmethod added by LSD
    }

# end of rpart.control.s LSD
```

BIBLIOGRAPHY

1. Ahn, H., Loh, W.-Y. Tree-structured proportional hazards regression modeling. *Biometrics*, 50(2):471-485, 1994.

2. Alexander, W.P. and Grimshaw, S.D. Treed Regression. *The Journal of Computational and Graphical Statistics*, 5:156-175, 1996.

3. Bacchetti, P. and Segal, M.R. Survival trees with time-dependent covariates: application to estimating changes in the incubation period of AIDS. *Lifetime Data Analysis,* 1(1):35-47, 1995.

4. Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. Classification and regression trees. Wadsworth Publishing Co Inc (Belmont, CA), 1984.

5. Chaudhuri, P., Huang, M.-C., Loh, W.-Y. and Yao, R. Piecewise-polynomial regression trees. *Statistica Sinica*, 4:143-167, 1994.

6. Chaudhuri, P., Lo, W.-D., Loh, W.-Y. and Yang, C.-C. Generalized Regression Trees. *Statistica Sinica*, 5:641-666, 1995.

7. Chen, Y. I. Simple-tree weighted logrank tests for right-censored data. *Ann. Inst. Statist. Math.*, 50:311-324, 1998.

8. Ciampi, A., Couturier, A. and Li, S. Prediction trees with soft nodes for binary outcomes. *Stat Med.*, 21(8):1145-65, Apr 30 2002.

9. Ciampi, A., Thiffault, J., Nakache, J.-P. & Asselain B. Stratification by stepwise regression, correspondence analysis and recursive partition: a comparision of three methods of analysis for survival data with covariates. *Computational Statistics and Data Analysis*, 4:185-204, 1986.

10. Ciampi, A., Hogg, S., McKinney, S. and Thiffault, J. A computer program for recursive partition and amalgamation for censored survival data. *Computer Methods and Programs in Biomedicine*, 26:239-56, 1988.

11. Ciampi, A., Lou, Z., Lin, Q. and Negassa, A. Recursive partition and amalgamation with the exponential family: theory and applications. *Applied Stochastic Models and Data Analysis*, 7:121-137, 1991.

12. Cox, D.R. Regression models and life tables. *Journal of the Royal Statistical Society Series B* 34:187-220, 1972.

13. Davis, R.B. and Anderson, J.R. Exponential survival trees. *Statistics in Medicine*, 8:947-961, 1989.

14. Fisher, E.R., Anderson, S., Dean, S., Dabbs, D., et al. Solving the dilemma of the immunohistochemical and other methods used for scoring estrogen receptor and progesterone receptor in patients with invasive breast carcinoma. *Cancer*, 103:164-73, 2005.

15. Gail, M.H. Evaluating serial cancer marker studies in patients at risk of recurrent disease. *Biometrics*, 37(1):67-78, Mar 1981.

16. Gail, M.H., Wieand, S. and Piantadosi, S. Biased estimates of treatment effect in randomized experiments with nonlinear regressions and omitted covariates. *Biometrika.* 71:431–444, 1984.

17. Gordon, L. and Olshen, R.A. Tree-structured survival analysis. *Cancer Treatment Reports*, 69:1065-1068, 1985.

18. Harrington, D.P. and Fleming, T.R. A class of rank test procedures for censored survival data. *Biometrika*, 69:553-566, 1982.

19. Hastie, T. and Tibshirani, R. Exploring the Nature of Covariate Effects in the Proportional Hazards Model. *Biometrics*, 46(4):1005-1016, 1990.

20. Hastie, T. and Tibshirani, R. Varying-coefficient models. *Journal of the Royal Statistical Society Series B*, 55:757-796, 1993.

21. Huang, X., Chen, S. and Soong, S.J. Piecewise exponential survival trees with time-dependent covariates. *Biometrics*, 54(4):1420-33, 1998.

22. Kaplan, E.L. and Meier, P. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457-481, 1958.

23. Kass, G.V. An exploratory technique for investigating large quantiles of categorical data. *Applied Statistics*, 29:119-127, 1980.

24. Kim, K. and Tsiatis, A.A. Study duration for clinical trials with survival response and early stopping rule. *Biometrics*, 46:81-92, 1990.

25. LeBlanc, M. and Crowley, J. Relative risk trees for censored data. *Biometrics*, 48:411-425, 1992.

26. LeBlanc, M. and Crowley, J. Survival trees by goodness of split. *Journal of the American Statistical Association*, 88:457-467, 1993.

27. Loh, W.-Y. and Shih, Y.-S. Split Selection Methods for Classification Trees. *Statistica Sinica*, 7:815-840, 1997.

28. Loh, W.-Y. and Vanichsetakul, N. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83(403):715-725, 1988.

29. Morgan, J.N. and Messenger, R.C. THAID: A sequential analysis program for the analysis of nominal scale dependent variables. *Technical report, Institute of Social Research, University of Michigan, Ann Arbor*, 1973.

30. Morgan, J.N. and Sonquist, J.A. Problems in the analysis of survey data and a proposal. *JASA*, 58:415-434, 1963.

31. Pettitt ,A. N. and Bin Daud, I. Investigating Time Dependence in Cox's Proportional Hazards Model. *Applied Statistics*, 39(3):313-329, 1990.

32. Sackett, D., et al. Evidence Based Medicine: What It Is and What It Isn't. BMJ, 312:7023, 1996.

33. Segal, M. Regression trees for censored data. *Biometrics*, 44:35-48, 1988.

34. Segal, M. Tree-structured methods for longitudinal data. *Journal of American Statistical Association*, 87:407-418, 1992.

35. S-PLUS 2000 Programmer's Guide, Data Analysis Products Division, MathSoft, Seattle, WA.

36. Stein, J., Kalb, G., Possinger, K., and Wernecke, K.-D. Extension to the Validation of Classification Trees. *Biometrical Journal*, 43:107-116, 2001.

37. Therneau, T. and Atkinson, E.J. An Introduction to Recursive Partitioning Using the RPART Routines. Mayo Foundation, 1997.

38. Therneau, T., Grambsch, P. and Fleming, T. Martingale based residual for survival models. *Biometrika*, 77:147-160, 1990.

39. Wernecke, K.-D., Possinger, K., Kalb, G., And Stein, J. Validating Classification Trees. *Biometrical Journal,* 40:993-1005, 1998.

40. Xu, R., and Adak, S. Survival analysis with time-varying regression effects using a tree-based approach. *Biometrics*, 58(2):305-315, 2002.

41. Zhang, H. Classification trees for multiple binary responses. *Journal of the American Statistical Association*, 93:180-193, 1998.