2013

Modifications to classification and regression trees to solve problems related to imperfect detection and dependence

Mark McKelvey
Iowa State University

Modifications to classification and regression trees to solve problems related to imperfect detection and dependence

by

Mark Wesley McKelvey

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Program of Study Committee:

Philip Dixon, Major Professor

Petrutza Caragea

Stephen Dinsmore

Heike Hofmann

Daniel Nordman

Iowa State University

Ames, Iowa

2013

DEDICATION

I would like to dedicate this thesis to my wife Beth and to my children without whose support I would not have been able to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like thank my major professor, Dr. Philip Dixon, for his guidance and infinite patience during this long, long journey of researching and writing my dissertation.

ABSTRACT

Classification and Regression Tree (CART) models provide a flexible way to model species' habitat occupancy status, but standard CART algorithms have plenty of room for extensions. One such extension explores the survey error of imperfect detection. When an individual is not detected, that is often taken as sign of non-presence. However, the principle of imperfect detection tells us that just because one cannot find what they are looking for, that does not mean that what they are looking for is not present. We outline four methods for including detection probability in the process of growing the tree, and illustrate these methods using data from a study of mountain plovers (Dinsmore et al. 2003). The results depend on the method used to estimate detection and occupancy. For the mountain plover data, the tree structures produced by three of the methods are identical to that produced by the naïve tree in which detection is ignored. The fourth method yields different splitting choices. Estimates of occupancy probability are consistently lower when using the naïve tree than those computed using detection-adjusted trees. Accounting for imperfect detection is crucial even when occupancy is modeled using a CART tree.

In addition to imperfect detection, another extension to standard CART algorithms deals with spatial correlation. Many studies include a cluster-type sampling design where there is a clear spatial correlation between sampling locations. This correlation causes the variance of the node occupancy estimates in CART to be biased. We suggest a generalized estimating equation (GEE)-based approach in which the naïve variance estimates (calculated as if all locations were independent) are "corrected" based on the data available in each parent node of the tree. The corrected variance estimates are then used to revise the binary-split decision criterion of the tree. The variances of each node in the split are assumed to be unequal. We demonstrate this method using data from a study on rats and also from a study on bird occurrences in Oregon.

When creating alternative methods of growing trees (i.e. how nodes are split) in CART, we expect to see [potentially] different trees. However, using those new methods also means that methodology involved in pruning the trees may need their own corresponding changes. For example, both of the types of methodology proposed above, incorporating imperfect detection and correlated data, led to an examination of current pruning criteria. Taking both of those new algorithms into account, we will discuss several pruning criteria that could be used in conjunction with our proposed CART methodology. We evaluated the performance of each criteria by using simulated examples for each criteria, which resulted in error rates that were used to assess the performance of the pruning criteria.

CHAPTER 1. OVERVIEW

1.1 Introduction

A classification and regression tree (CART) is a flexible alternative to logistic and linear regression models (De'ath and Fabricius 2000, Breiman et al. 1984). Unlike regression, CART can account for interactions in its binary tree structure because it does not require the assumption of additivity. CART allows the simultaneous use of both quantitative and qualitative covariates, makes no distributional assumptions, and is invariant to monotone transformations of variables (Breiman et al. 1984). Another advantage of CART is its ability to handle missing values. Whereas logistic regression would discard any individual with data missing from any one (or more) of the covariates, CART can still use that individual's data to help formulate the model (Clark and Pregibon 1992, Harrell 2001, De'ath and Fabricius 2000, Breiman et al. 1984). These properties, along with its relative ease of construction and interpretation, have lead to widespread use of CART in ecology (e.g. Pesch et al 2011, Davidson et al 2010, Lehmann et al 2011, Maslo et al 2011).

CART creates a binary tree through recursive partitioning of the observations in the data set. A group of observations in the tree is called a "node". At each single "parent" node in the tree, CART attempts to partition the data from the parent node into two more-homogeneous "daughter" nodes. CART uses an exhaustive processing algorithm that uses all of the covariates, checking every possible way to split the observations based on breaks in the covariate distributions.

Homogeneity of a node is often referred to as impurity. A measure of impurity is generally continuous and bounded on the lower end by zero, although it can be any formula which has a value to identify a perfectly homogeneous node (such as zero) and an increasing scale to

measure the relative strength of the split. A measure of zero indicates that no further splitting is required, while measures further from zero indicate an increasing propensity toward splitting. Impurity could also take the form of a statistical deviance (De'ath and Fabricius 2000, Clark and Pregibon 1992) when using a specific statistical model.

Some common measures of impurity for a classification tree (Breiman et al. 1984) are Entropy, Misclassification, Twoing and the Gini Index, which is often the default measure used for splitting in classification trees. When the responses are binary (yes / no), the Gini Index defines impurity at a node with n observations as

$$(2 \times \frac{\#Yes}{n}) * (1 - \frac{\#Yes}{n}) \tag{1.1}$$

A calculated measure is used to rank each of the possible splits of a parent node. This measure is often based on a measure of impurity taken from each node in the triad (the parent node and two daughter nodes). In literature, this value might be referred to as a "decrease in impurity" or a "drop in deviance". We also note that at times in the literature, "Deviance" appears to have referred to impurity in general, a specific function of impurity, or a statistical deviance based on a model. The combined impurity at the two daughter nodes is the sum of the impurity (e.g. the Gini Index) for each node. Then the decrease in impurity of the proposed split is defined as the difference between the impurity of the parent node and the combined impurity of the two daughter nodes. The proposed split with the largest decrease would be chosen for use in the tree.

When creating a CART model, the tree could potentially continue growing until every terminal node has only 1 observation in it. In order to avoid overfitting, CART employs several rules on when to stop growing the tree. The simplest of these rules is to stop splitting if the parent node in question is perfectly homogeneous with respect to the response variable.

Another very basic rule is that of node size. This constraint specifies the minimum parent node size required to even consider splitting, as well as setting the minimum daughter node size required to accept a proposed split. For example, it may be unreasonable to consider splitting a node with only 10 observations. Also, regardless of the homogeneity it may provide, splitting a node with 50 observations into a node with 48 observations and another with 2 observations

may not be statistically prudent.

Another stopping mechanism involves a choice of a value for the complexity parameter (cp). During tree construction, an estimated complexity value (cp*) is calculated for each potential split. If the estimated complexity cp* is not as large as the specified level (cp), then the split being considered is not made. The cp value can also be used post-tree formation to further prune the tree.

The user can also gain extra control through the definition of the measure of node impurity (node deviance) or through the calculation of the drop in deviance of potential splits. For classification modeling, the node deviance (D), along with the complexity parameter (cp), can be used together in a cost-complexity pruning algorithm. If

$$D_{parent} - \sum D_{child} + cp$$

is not greater than zero for at least one set of splits below the parent node, then that initial split is not worth keeping at the current level of cp (Therneau and Atkinson 2011).

Of particular interest to this paper are uses of CART for occupancy modeling (e.g. De'ath and Fabricius 2000; Bourg et al. 2005; Castellón and Sieving 2006). It predicts occurrence or abundance of a species using environmental covariates such as temperature, distance to water, and food availability. Occupancy modeling can be especially useful when the subject is difficult to find (due to mobility, concealment, or if the area being searched is very large in size), thus leading to scenarios with imperfect detection! It can also be used to predict occurrence in un-surveyed areas or potentially to understand the biological mechanisms. Sample data are often represented in clusters, which can cause issues with precision in model estimates.

This thesis addresses two main issues. The first issue being the application of CART to a set of occupancy data collected with the potential for imperfect detection. In order to build a classification (or regression) tree, a general assumption is that the identification status (or quantitative characteristic) of the individuals which you are using to create the tree are known, which then makes the internal calculations fairly straightforward. Modeling occupancy data with detection issues complicates this idea. If a status or characteristic value is uncertain,

that should change the creation process and interpretation of the CART tree. We will propose several methods as solutions to this problem.

The second issue involves any study in which the data were collected from clustered observations. This violates another assumption of CART, which is that of independence. Independence is a highly sought-after commodity when collecting data, and is the default for many statistical methods. Without independence, we are left scrambling to compensate for the changes in variance that are sure to occur, and must adjust the modeling techniques accordingly. We propose a method that will adjust CART for the presence of correlated data.

Chapter 2 will focus on methods to solve the problem of using CART to model occupancy data with imperfect detection. Chapter 3 will be concerned with a proposal to adjust CART for the presence of correlated data. Chapter 4 presents a new way to think about evaluating and reducing CART trees to a manageable size in light of the methodology introduced in Chapters 2 and 3. Finally, Chapter 5 will summarize the findings of the previous three chapters.

CHAPTER 2. Incorporating Detection into Classification and Regression Trees for Occupancy Modeling

Classification and Regression Tree (CART) models provide a flexible way to model species' habitat occupancy status, but standard CART algorithms ignore imperfect detection. We outline four methods for including detection probability in the process of growing the tree, and illustrate these methods using data from a study of mountain plovers (Dinsmore et al. 2003). The results depend on the method used to estimate detection and occupancy. The tree structures produced by three of the methods are identical to that produced by the naïve tree in which detection is ignored. The fourth method yields different splitting choices. Estimates of occupancy probability are consistently lower when using the naïve tree than those computed using detection-adjusted trees. Accounting for imperfect detection is crucial even when occupancy is modeled using a CART tree.

2.1 Introduction

Occupancy modeling (OM) is a widely used tool among ecologists and wildlife researchers. It predicts occurrence or abundance of a species using environmental covariates, such as temperature, distance to water, and food availability. OM can be especially useful when the subject is difficult to find (due to mobility, concealment, or if the area being searched is very large in size). It can also be used to predict occurrence in un-surveyed areas or potentially to understand the biological mechanisms. Recent examples include predicting occupancy for the Siberian flying squirrel (Reunanen et al. 2002), describing the spatial population structure of the leaf-mining moth (Gripenberg et al. 2008), conservation planning for Finnish butterflies at different spatial scales (Cabeza et al. 2010), examining the role of habitat quality on chinook salmon spawning

(Isaak et al. 2007), and modeling nest-site occurrence for the Northern Spotted Owl (Stralberg et al. 2009).

A recent trend in OM research is to develop models that allow imperfect detection. With perfect detection, one model for occupancy at site i is a Bernoulli distribution, $Y_i \sim Bern(\pi_i)$, where $Y_i = 1$ if Seen (i.e. Detected) at site i, and π_i = Prob(Occupancy) at site i = $f(X_1, X_2, \cdots X_k)$. The function $f(X_1, X_2, \cdots, X_k)$ relates Prob(occupancy) to the k environmental covariates $X_1, X_2, \cdots X_k$. However, when detection is imperfect, i.e. Prob(Detect|Occupied) = $\pi_d < 1$, then π_{occ} becomes partially masked by π_d. A better model would be $Y_i \sim Bern(\pi_{occ}\pi_d)$, The expected probabilities of observed data at a site on a given occasion are then:

$$\begin{aligned} P(Y=1) &= \text{probability of 'Seen'} = \pi_{occ}\pi_d \\ P(Y=0) &= \text{probability of 'Not Seen'} = 1 - \pi_{occ}\pi_d \\ &= (1-\pi_{occ}) + \pi_{occ}(1-\pi_d) \\ &= P(\text{Not there}) + P(\text{there but not seen}) \end{aligned}$$

Logistic regression is often used to predict occupancy probabilities. The logit function, $\log[\pi_{occ}/(1-\pi_{occ})]$ relates occupancy probability to the environmental covariates.

$$\log(\frac{\pi_{occ}}{1-\pi_{occ}}) = \beta_0 + \beta_1 * x_1 + ... + \beta_p * x_p$$

Failure to account for imperfect detection in logistic regression models is known to lead to biased estimates of occupancy probability (Tyre at al. 2003, Gu and Swihart 2004).

Logistic regression requires an explicit model for the influence of the environmental covariates. Commonly, the logistic regression model assumes linearity and additivity. The additivity assumption can be relaxed by including interaction terms, but without good knowledge of the appropriate interactions to include, this often leads to a large number of model terms and over-fitting (Hosmer and Lemeshow 2000).

A classification and regression tree (CART) is a flexible alternative to logistic classification models (De'ath and Fabricius 2000, Breiman et al. 1984). Unlike logistic regression, CART can account for interactions in its binary tree structure because it does not require the assumption of

additivity. CART allows the simultaneous use of both quantitative and qualitative covariates, makes no distributional assumptions, and is invariant to monotone transformations of variables (Breiman et al. 1984). Another advantage of CART is its ability to handle missing values. Whereas logistic regression would discard any individual with data missing from one of its covariates, CART can still use that individual's data to help formulate the model (Clark and Pregibon 1992, Harrell 2001, De'ath and Fabricius 2000, Breiman et al. 1984). These properties, along with its relative ease of construction and interpretation, have lead to widespread use of CART in occupancy modeling (e.g. De'ath and Fabricius 2000; Bourg et al. 2005; Castellón and Sieving 2006).

CART creates a binary tree by recursively paroning the observations in the data set. At each split, CART attempts to partition the data in a parent node into two more-homogeneous nodes. A calculated measure, often based on node impurity, is used to rank each of the possible splits of a parent node. Some common measures of node impurity for a classification tree (Brieman et al. 1984) are Entropy, Misclassification, Twoing, and the Gini index, which is often the default measure for splitting in a classification tree. When the responses are binary (yes / no), the Gini Index defines impurity at a node with n observations as

$$(2 \times \frac{\#Yes}{n}) * (1 - \frac{\#Yes}{n}) \tag{2.1}$$

The combined deviance at the two daughter nodes is the sum of (2.1) for each node. Then the calculated measure of the proposed split is defined as the drop in deviance from the parent node to the two combined daughter nodes. In general, a bigger change implies a better split.

However, the default use of a CART tree with seen/not seen data to model occupancy does not account for imperfect detection. Our goal is to modify the CART node-split We use a pair of simulated data sets and data on mountain plover sightings to illustrate our method. We present four methods for CART that incorporate detection probability into the tree-splitting process. Each method was evaluated using the same test data set and the results are compared. We present a measure (AIC) to evaluate the quality of the resulting tree. Our results from the mountain plover data suggest that using a single parameter to model detection probability within the whole tree is preferable.

2.1.1 Examples

2.1.1.1 Simulated Examples

We simulated two data sets in which the detection probability varied with environmental variables and their interaction. The Test1 data set has 3 variables. X3 is associated with detection probability, while X1 and X2 are associated with occupancy probability. The Test2 data set has 6 variables. X4 and X5 are associated with detection probability, variables X1, X2, X4, and X6 are associated with the occupancy probability, and X3 was irrelevant.

2.1.1.2 Plovers Example

Our motivating example is a multi-year study of Mountain Plovers in Montana, USA (Dinsmore et al 2003). Mountain Plovers were searched for on prairie dog colonies during three sampling periods each year (20 May-10 June, 11-30 June, and 1-20 July) over a period of 13 years (1995-2007). During each search plovers were either seen (1) or not seen (0) on each prairie dog colony. There were a total of 81 colony sites involved in the survey, although not every site was sampled in every year. In 9 of the 13 years, some covariate information for each colony was collected simultaneously with the occupancy data. The covariates in the data set include the area of the prairie dog colony (AREA) and two colony shape metrics , a patch shape index (PSI) and a measure of perimeter-to-area ratio (PARA).

We removed the third sampling period in each year after exploratory analysis indicated that the detection probability in that period was substantially different from detection in the first two surveys. There is previous evidence for declining detection probabilities within each year (Dinsmore et al. 2003). We chose to work with the data from 2002. That year had a large number of sites visited twice (54) and a relatively low rate of naïve detection (#ones over 2*#sites), thus the probability of misclassifying sites as 'non-occupied' may be higher than in other years.

2.2 Methods

Taking a likelihood-based approach, we propose four possible approaches to incorporate detection into a potential split from a parent node to two daughter nodes. We use π_P, π_L, and π_R to denote occupancy probabilities for the parent, left daughter, and right daughter nodes, respectively. At each node, n_0, n_1 and n_2 are the number of sites with no detections, one detection or two detections. The four approaches are:

1. Separate detection and occupancy parameters (π_d and π_{occ}) at each node (6 total).
   (This method is referred to as parent.v.daughter.v.daughter, or p.v.d.v.d.)
   The likelihood at each node in this situation is

$$L(\pi_d, \pi_{occ}|n_0, n_1, n_2) \propto [(1-\pi_{occ}) + \pi_{occ}(1-\pi_d)^2]^{n_0} * [2\pi_{occ}\pi_d(1-\pi_d)]^{n_1} * [\pi_{occ}{\pi_d}^2]^{n_2}$$

   and the solutions to the score equations at a node are

$$\hat{\pi}_{occ} = \frac{n_1+n_2}{(2\pi_d - {\pi_d}^2)(n_0+n_1+n_2)} \quad (2.2)$$

$$\hat{\pi}_d = \frac{2n_2}{2n_2+n_1} \quad (2.3)$$

2. One detection parameter, π_d, that applies to all 3 nodes (parent, left daughter and right daughter), but 3 separate occupancy parameters (4 parameters total). We consider three estimators of π_d:

   (a) estimate π_d from the root of the tree (all observations)and assume that detection remains constant throughout the tree.
   (This method is referred to as orig.parent.)

   (b) estimate π_d from the parent node for each split, and assume that detection remains constant only within that split.
   (This method is referred to as each.parent.)

   (c) estimate π_d from the two proposed daughter nodes and assume that detection remains constant that split.

(This method is referred to as parent.v.2daughters, or p.v.2d)

The joint likelihood is

$$\begin{aligned} L(\pi_d, \pi_L, \pi_R | n_{0L}, n_{1L}, n_{2L}, n_{0R}, n_{1R}, n_{2R}) \quad \propto \quad & [(1-\pi_L) + \pi_L(1-\pi_d)^2]^{n_{0L}} * [2\pi_L\pi_d(1-\pi_d)]^{n_{1L}} \\ & *[\pi_L {\pi_d}^2]^{n_{2L}} * [(1-\pi_R) + \pi_R(1-\pi_d)^2]^{n_{0R}} \\ & *[2\pi_R\pi_d(1-\pi_d)]^{n_{1R}} * [\pi_R {\pi_d}^2]^{n_{2R}} \end{aligned}$$

Examining the solutions to the score equations ((2) and (3)), it can be seen that in the case when there are no '11' sites (i.e. $n_2 = 0$) and there is at least one '01' or '10' site (i.e. $n_1 > 0$), then $\hat{\pi}_d = 0$ and $\hat{\pi}_{occ} = \infty$. There are other situations (not easily identifiable) when $\hat{\pi}_d \in [0, 1]$ but $\hat{\pi}_{occ} > 1$. The use of optim() and a logit transformation of the probabilities prevents these situations, as well as preventing estimates of exactly 1 or exactly 0, which can cause errors in the calculation of the likelihoods.

For each method, the "drop in deviance" of a split takes the form of the test statistic of a Likelihood Ratio Test (LRT) between the parent node (same occupancy probability in all sites) and the two daughter nodes (two different occupancy probabilities). From all potential splits, the split with the largest test statistic is taken as the split used in the tree.

Akaike Information Criterion (AIC) is used as a measure of model selection between the models produced from each of the four proposed methods. AIC is calculated as -2*$l(\theta)$ + 2k, where $l(\theta)$ represents the maximized value of the Likelihood function based on the estimates $\hat{\pi}_d = 0$ and $\hat{\pi}_{occ}$, and k is the number of parameters in the model. Models are penalized for added complexity (additional parameters). Models with the lowest AIC values are generally preferred over other models.

2.2.1 Implementation

To obtain results using the naïve method, which ignores the issue of detection, we combined the two sampling periods into one set (i.e. one "occasion") of response data. We labeled each site with at least one '1' (i.e. '11', '01', or '10') as 'occupied' and sites that had a '00' received a 'not-occupied' designation. We then used the Gini Index as the measure of impurity to produce the naïve tree.

For the imperfect detection methods, we replaced the Gini Index with a statistical deviance calculated using likelihood functions so that we could include a parameter for the detection probability. We maximized the log-likelihoods using the optim() function in R with the BFGS method. The two parameters (π_{occ} and π_d) were estimated on the logistic scale and then the results were back-transformed, thereby ensuring that our estimates fell within the parameter space (i.e. between 0 and 1) without having to truncate any estimates. Those estimated probabilities were then used to calculate the log-likelihood at each of the three nodes, which in turn was used to calculate the drop in deviance (in the form of a LRT) of the proposed split. This was done for every possible split from a parent node.

We performed the CART analyses with the *rpart()* function from in program R, which can be found in the rpart package (Therneau and Atkinson 2010). We utilized the "user splits" option, which allows the creation and use of non-standard splitting functions and criteria.

2.3 Results

2.3.1 Simulation Results

From the test1 and test2 simulations, the best (using AIC) imperfect detection models were parent.v.2daughters (for test1) and the orig.parent model (for test2) method. We then compared the tree from the best models of each simulation to the trees produced assuming perfect detection. Figure 2.1 displays these results. We show only the tops of the trees in an effort to conserve space. The test1 trees show two differences in splits, in Nodes 2 (1st left-daughter node) and 13 (among the lower-right branches). The nodes in both trees split on the x1 variable, but do so in different places. For Node 2, this may only result in a small change in estimated π_{occ}, since the resulting nodes are still classified as "unoccupied". However, for Node 13, the change in estimated π_{occ}, coupled with allowing $\pi_{det} < 1$, is enough to declare the sites "occupied". Between the test2 trees, we see only one structural difference, in Node 2, that has a similar effect as the Node 2 changes seen in the test1 trees.

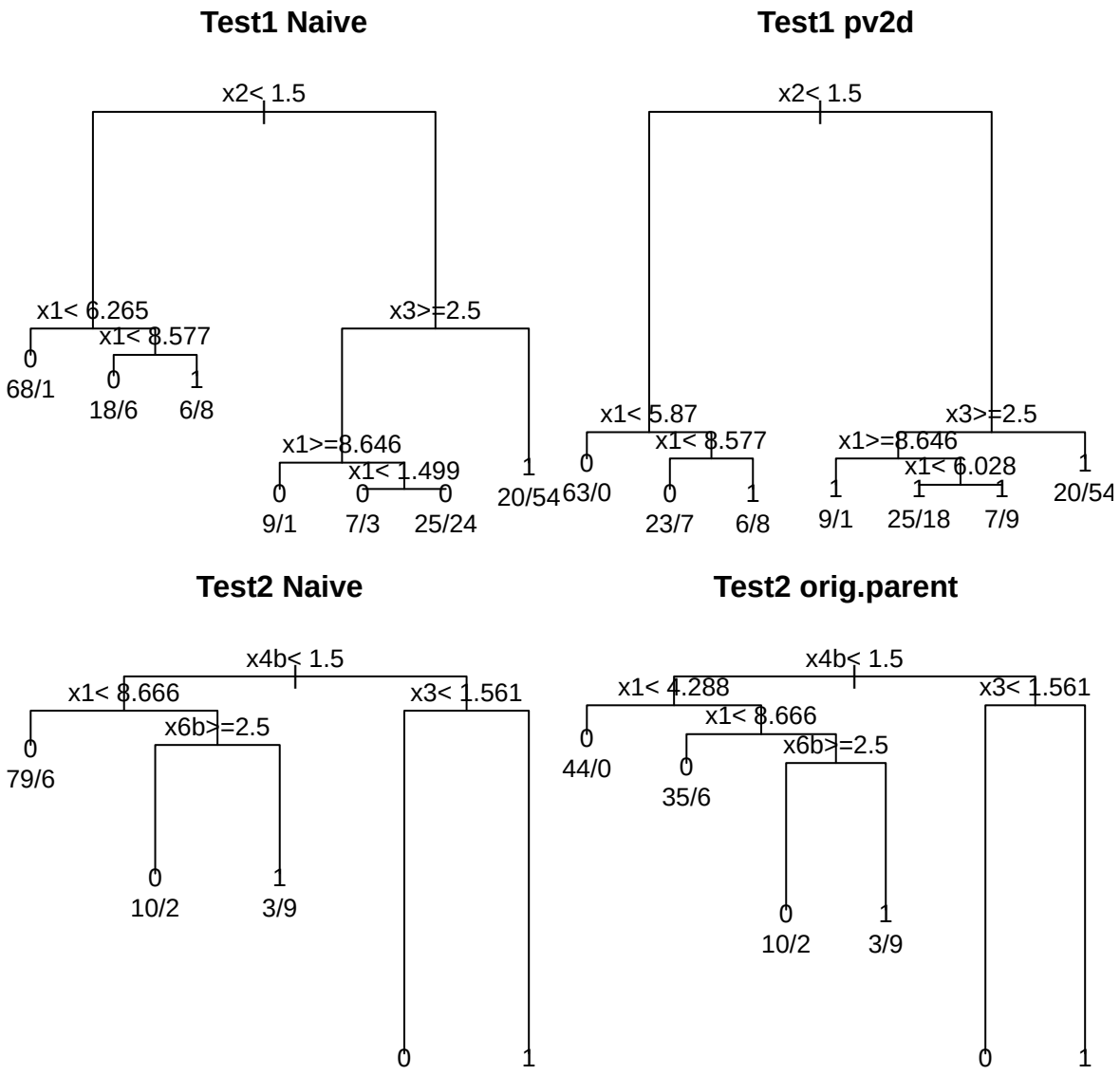

Figure 2.1 Trees produced from the test1 and test2 simulations, comparing the naïve method to the imperfect detection methods parent.v.2daughters (test1) and orig.parent (test2). Both sets of trees display structural changes that can occur when accounting for imperfect detection. In the test1 trees, this occurs in Node 2 (the 1st left-daughter node) and Node 13 (among the lower-right branches). The test2 trees only reveal a structural difference in Node 2.

2.3.2 Study Results

From the analysis of the plover data, we see that site estimates of occupancy probability for three of the proposed methods are quite similar (Figure 2.2, Table 2.1), but the fourth method, p.v.d.v.d., produces very different estimates from the other methods (Table 2.2). The naïve method generally produces smaller estimates of occupancy probability than do the proposed methods. However, despite the estimates of occupancy and detection changing between methods, the tree structure for all of the proposed methods except p.v.d.v.d. was identical to that of the naïve tree (Figure 2.3).

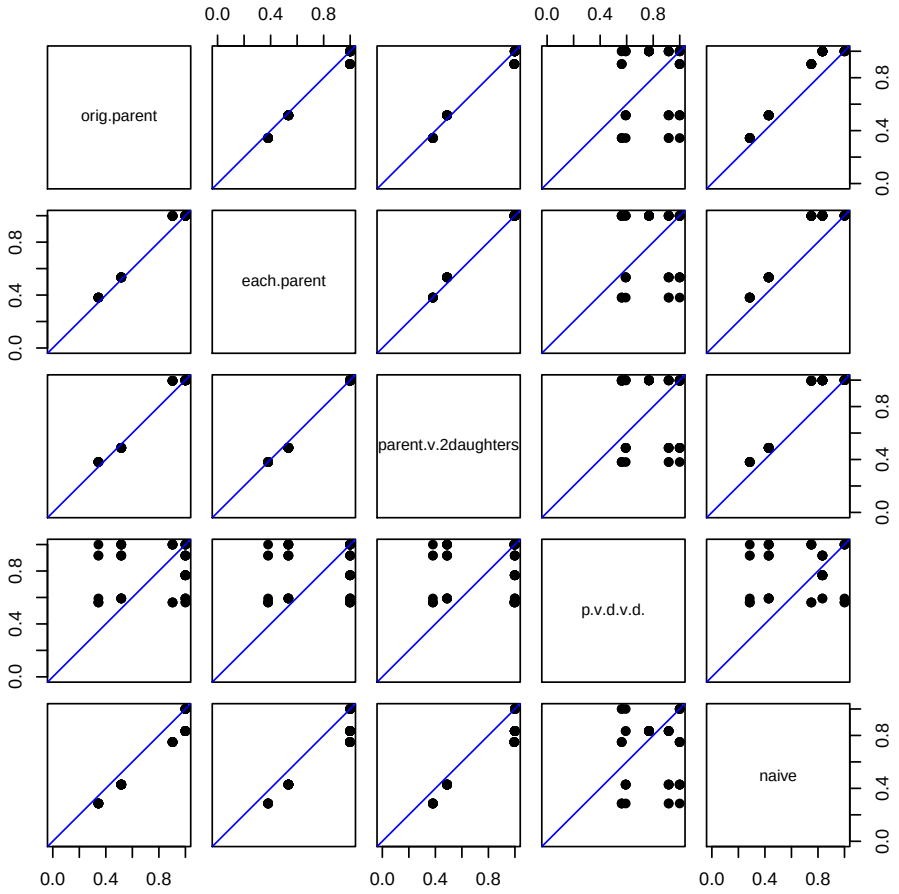

Figure 2.2 Comparing estimates of occupancy at each site between the 5 methods. The straight lines have a slope of 1 and represent equality. Three of the proposed methods are quite similar (orig.parent, each.parent, and parent.v.2daughter), while the fourth proposed method produces very different estimates. The naïve method generally produces smaller estimates of occupancy than the proposed imperfect detection methods do.

Table 2.1 The node estimates of occupancy and detection from the naïve method, as well as from the proposed methods orig.parent (orig.par), each.parent (each.par), and parent.v.2daughters (p.v.2d). A (*) represents a terminal node of the tree. Note that while the tree structure is the same for each of these methods, the node estimates are very different from the naïve method.

			naïve	naïve	Orig.par	Orig.par	Each.par	Each.par	p.v.2d	p.v.2d
tree node	Split	seen/n	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$
1	root	36/54	0.667	1	0.803	0.588	0.803	0.588	0.803	0.588
2	AREA <= 4366.5	21/36	0.583	1	0.702	0.588	0.702	0.588	0.701	0.590
4	PARA <= 4565.0	15/28	0.536	1	0.645	0.588	0.714	0.500	0.713	0.501
8*	PARA > 2038.0	2/7	0.286	1	0.344	0.588	0.381	0.500	0.381	0.500
9	PARA <= 2038.0	13/21	0.619	1	0.745	0.588	0.825	0.500	0.825	0.500
18*	AREA > 1803.0	6/14	0.429	1	0.516	0.588	0.534	0.556	0.488	0.651
19*	AREA <= 1803.0	7/7	1.000	1	0.999	0.588	0.999	0.556	0.999	0.651
5*	PARA > 4565.0	6/8	0.750	1	0.903	0.588	0.998	0.500	0.995	0.501
3*	AREA > 4366.5	15/18	0.833	1	0.999	0.588	0.999	0.588	0.998	0.590

Table 2.2 The node estimates of occupancy and detection resulting from the p.v.d.v.d. method. A (*) represents a terminal node of the tree. While occupancy estimates appear similar to those in Table 2.1, they should not be compared because of the different tree structures.

tree node	Split	seen/n	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$
1	root	36/54	0.8028	0.5882
2*	PSI > 231	5/9	0.5926	0.7500
3	PSI <= 231	31/45	0.8560	0.5581
6*	AREA <= 1261	4/8	0.5625	0.6667
7	AREA > 1261	27/37	0.9250	0.5406
14	AREA <= 6103	19/26	0.9997	0.4617
28*	AREA <= 2835	11/15	0.9998	0.4334
29*	AREA > 2835	8/11	0.9167	0.5454
15*	AREA > 6103	8/11	0.7682	0.7692

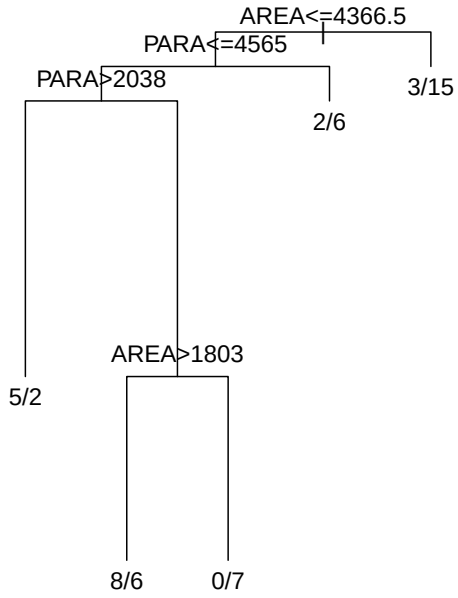

Figure 2.3 The tree produced using methods orig.parent, each.parent, and parent.v.2daughters, shown here, is identical to the naïve tree produced without accounting for imperfect detection. However, Table 2.1 displays the differences in the parameter estimates, which show the advantage of the imperfect detection methods.

Based on the AIC values shown in Table 2.3, the best model for the plover data appears to be the orig.parent model, which only estimates one detection probability parameter that is constant across all nodes (and thus all sites). Assuming a closed population, our original data clearly shows that the naïve method is incorrect; any sites with observed data 0-1 or 1-0 (seen once, not seen once) indicate a need to account for imperfect detection.

Table 2.4 shows the progression of AIC as we continue down the tree. Even though we are adding one additional [occupancy] parameter at each split (one parent parameter becomes two daughter parameters), the AIC remains very similar until the final split, when it drops noticeably.

Table 2.3 AIC values computed using the plover data. AIC is calculated as -2*$l(\theta)$ + 2k. The best model is the orig.parent model.

Model	**AIC**
Orig.parent	116.22
p.v.2d	121.15
Each.parent	121.93
p.v.d.v.d.	131.26

Table 2.4 The results from the proposed orig.parent method, along with AIC computed at each split in the tree. Although the AIC calculation involves both leaves, the value is shown only for the left-daughter node of each split. A (*) represents a terminal node of the tree.

			Orig.parent	Orig.parent	AIC at each split
tree node	Split	seen/n	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$	
1	root	36/54	0.8028	0.5882	121.65
2	AREA <= 4366.5	21/36	0.7024	0.5882	120.03
4	PARA <= 4565.0	15/28	0.6451	0.5882	120.79
8*	PARA > 2038.0	2/7	0.3441	0.5882	120.41
9	PARA <= 2038.0	13/21	0.7454	0.5882	
18*	AREA > 1803.0	6/14	0.5161	0.5882	116.22
19*	AREA <= 1803.0	7/7	0.999979	0.5882	
5*	PARA > 4565.0	6/8	0.9031	0.5882	
3*	AREA > 4366.5	15/18	0.9996	0.5882	

2.4 Discussion

The simulated test1 and test2 data reveal that there can be structural changes caused by failing to account for imperfect detection when using CART, in addition to the less-visible (but still important) changes in parameter estimates at each node. Site classifications, using a cutoff of $\pi_{occ} = 0.5$, were changed from "unoccupied" to "occupied" designations.

The plover data, while an excellent real-world example of a situation where imperfect detection methodology should be applied, failed to lead to any structural differences between the naïve tree and the trees for three of the four proposed methods. It is possible that similar detection probabilities throughout the tree, or the use of so few covariates, contributed to the lack of tree diversity.

The p.v.d.v.d. method classified all sites as being occupied (using a cutoff of $\pi_{occ} = 0.5$). This seems very unusual, and you might think that allowing detection probabilities to change throughout the tree caused the tree structure to change. This is partially true, but allowing detection to differ at each node leads to erroneous use of the LRT. For the p.v.d.v.d. method, the LRT is a test of ANY difference between nodes (i.e. splitting decisions could be due to differences in detection and not just occupancy).

The idea of incorporating detection into classification and regression trees can be extended to random forests (Breiman 2001). Unfortunately, the random forest approach does not result in one final tree; it only reports an estimate for each individual, aggregated over all trees (the modal group for classification or the average for regression). A random forest would estimate π_{occ} for each site, averaged over a collection of trees, but it will not identify a single tree structure. A single-tree CART approach may be more desirable due to its interpretability and ability to visually represent results (Prasad et al., 2006). Another CART extension that could benefit from using a detection parameter is boosted regression trees. Elith et al (2006), in their review of more than a dozen models for predicting species' distributions from occurrence data, count boosted regression trees among the best models tested for various scenarios. These trees could still be improve by using detection to help estimate occupancy during the creation of each individual decision tree.

2.5 Conclusion

Classification and regression trees have proven their worth as an effective statistical modeling tool for many situations. One area where they have previously lacked the ability to create an accurate model is in situations involving multiple surveys with imperfect detection. Being able to account for detection probability in the pursuit of predicting presence/absence of a desired individual (or characteristic, etc.) allows CART to be more accurate in its predictions. While they are by no means the final say in modifications to CART for detection, the four alternative methods presented here enable CART to be extended to statistical analyses which would otherwise use a different modeling tool.

2.6 Literature Cited

Bourg, Norman A., William J. McShea, Douglas E. Gill. 2005. Putting a CART Before the Search: Successful Habitat Prediction for a Rare Forest Herb. *Ecology.* Vol. 86, No. 10, p. 2793-2804.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. 1984. Classification and Regression Trees. Wadsworth International Group, Belmont, CA.

Breiman, Leo. 2001. Random Forests. *Machine Learning.* Vol. 45, p. 5-32.

Cabeza, Mar, Anni Arponen, Laura Jttel, Heini Kujala, Astrid van Teeffelen and Ilkka Hanski. 2010. Conservation planning with insects at three different spatial scales. *Ecography.* vol. 33, No. 1, p. 54-63.

Castellón, Traci D., and Sieving, Kathryn E. 2006. Landscape History, Fragmentation, and Patch Occupancy: Models for a Forest Bird with Limited Dispersal. *Ecological Applications.* Vol. 16, No. 6, pp. 2223-2234.

Clark, L.A. and D. Pregibon. 1992. Chapter 9: Tree-based Models *In*: J.M. Chambers and T.J. Hastie (eds.). *Statistical Models in S.* Wadsworth and Brooks, Pacific Grove, CA.

Davidson, T. A., Sayer, C. D., Langdon, P. G., Burgess, A. and Jackson, M. 2010. Inferring past zooplanktivorous fish and macrophyte density in a shallow lake: application of a new regression tree model. *Freshwater Biology* Vol. 55, p. 584599.

De'ath, Glenn, and Fabricius, Katharina E. 2000. Classification and Regression Trees: A Powerful Yet Simple Technique for Ecological Data Analysis. *Ecology.* Vol. 81, No. 11, pp. 3178-3192.

Dinsmore, Stephen J., Gary C. White, Fritz L. Knopf. 2003. Annual Survival and Population Estimates of Mountain Plovers in Southern Phillips County, Montana. *Ecological Applications.* Vol. 13, No. 4, pp. 1013-1026.

Elith, J., Graham, C.H., Anderson, R.P., Dudk, M., Ferrier, S., Guisan, A., Hijmans, R.J., Huettmann, F., Leathwick, J.R., Lehmann, A., Lucia, J.L., Lohmann, G., Loiselle, B.A., Manion, G., Moritz, C., Nakamura, M., Nakazawa, Y., Overton, J.McC., Peterson, A.T., Phillips, S.J., Richardson, K., Scachetti-Pereira, R., Schapire, R.E., Sobero n, J., Williams, S., Wisz, M.S., and Zimmermann, N. 2006. Novel methods improve prediction of species distributions from occurrence data. *Ecography* Vol. 29, p. 129155.

Gripenberg, Sofia, Otso Ovaskainen, Elly Morrin, and Tomas Roslin. 2008. Spatial population structure of a specialist leaf-mining moth. *Journal of Animal Ecology.* Vol. 77, p. 757-767.

Gu, W. and Swihart, R.K. 2004. Absent or undetected? Effects of non-detection of species occurrence on wildlife-habitat models. *Biological Conservation* vol. 116, p. 195-203.

Harrell, Frank. 2001. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis.* Springer-Verlag, New York, NY.

Hosmer, David W. and Lemeshow, Stanley. 2000 *Applied Logistic Regression.* John Wiley and Sons, Inc., New York, NY.

Isaak, Daniel J., Russell F. Thurow, Bruce E. Rieman, Jason B. Dunham. 2007. Chinook Salmon Use of Spawning Patches: Relative Roles of Habitat Quality, Size, and Connectivity. *Ecological Applications.* Vol. 17, No. 2, pp. 352-364.

Lehmann, C. E. R., Archibald, S. A., Hoffmann, W. A. and Bond, W. J. 2011. Deciphering the distribution of the savanna biome. *New Phytologist.* Vol. 191, p. 197209.

Maslo, Brooke, Handel, Steven N. and Pover, Todd. 2011. Restoring Beaches for Atlantic Coast Piping Plovers (Charadrius melodus): A Classification and Regression Tree Analysis of Nest-Site Selection. *Restoration Ecology.* Vol. 19, p. 194203.

Pesch, Roland, Gunther Schmidt, Winfried Schroeder, Inga Weustermann. 2011. Application of CART in ecological landscape mapping: Two case studies. *Ecological Indicators.* Volume 11, No. 1, p. 115-122.

Prasad, A.M., L.R. Iverson and A. Liaw. 2006. Newer Classification and Regression Tree Techniques: Bagging and Random Forests for Ecological Prediction. *Ecosystems.* Vol. 9, No. 2, p. 181199.

Reunanen, Pasi, Ari Nikula, Mikko Mnkknen, Eija Hurme, Vesa Nivala. 2002. Predicting Occupancy for the Siberian flying squirrel in old-growth forest patches. *Ecological Applications.* Vol. 12, No. 4, pp. 1188-1198.

Stralberg, Diana, Katherine E. Fehring, Lars Y. Pomara, Nadav Nur, Dawn B. Adams, Daphne Hatch, Geoffrey R. Geupel, Sarah Allen. 2009. Modeling nest-site occurrence for the Northern Spotted Owl at its southern range limit in central California. *Landscape and Urban Planning.* Vol. 90, Issues 1-2, p. 76-85.

Therneau, Terry and Atkinson, Elizabeth. R port by Brian Ripley. 2010. rpart: Recursive Partitioning. R package version 3.1.-46. http://CRAN.R-project.org/package=rpart

Tyre, A.J., Tenhumberg, B., Field, S.A., Niejalke, D., Parris, K., and Possingham, H.P. 2003. Improving precision and reducing bias in biological surveys: estimating false-negative error rates. *Ecological Applications*. vol. 13, p. 1790-1801.

2.7 Extra material: R Code

2.7.1 User Splits

```
# Mark McKelvey  attempting to use the "user splits" option in R
#               Requires 3 pieces:  Init, Eval, and Splits
#             Optional 4th part called 'parms' to pass in other information
#                      *NOTE: parms MUST be part of the call to rpart().
#                             It will not work from the global environment.
#              I also change init$functions$print and init$functions$text


options(warn = 1)  #prints warnings as they occur,
 #      rather than waiting until the end
options(digits=4)  # controls number of digits/decimals (default is 7)
                   #  if digits is set too low, numbers may go to
                   #   scientific notation
library(rpart)
set.seed(7)

#####################################################################
# The 'evaluation' function.  Called once per node.
#  Produce a label (1 or more elements long) for labeling each node,
#  and a deviance.  The latter is
#       - of length 1
#       - equal to 0 if the node is "pure" in some sense (unsplittable)
#       - does not need to be a deviance: any measure that gets larger
#            as the node is less acceptable is fine.
#       - the measure underlies cost-complexity pruning, however

###############
###############  Mark's eval() code
temp1 <- function(y, wt, parms) {
 print("***** START: Evaluating *****")
     Ns <- timesseen(y);
```

```
     # NOTE: Ns[1] = n0 = never seen, Ns[2] = n1 = seen once,
     #       Ns[3] = n2 = seen twice...

     # If using orig.parent, I'd like to report the same prob.det
     # being used in the split function; also the corresponding prob.occ
     if(parms$mygoodness==1){ param.parent <- optim(c(0.5), lnl.t.fixed,
       gr=NULL, method="BFGS", control=list(fnscale=-1),
       prob.det = parms$prob.det, Ns = Ns )$par
         param.parent <- backt(param.parent)
         prob.occ <- param.parent[1]
         prob.det <- parms$prob.det
                                  }
      # Anything else, I will report the node-specific prob.det and prob.occ
      # Note that for each.parent and p.v.2d methods, this does not reflect
      #  the values being used in the split() function
         else{
         param.parent <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
                             control=list(fnscale=-1), Ns=Ns )$par
         param.parent <- backt(param.parent)
         prob.occ <- param.parent[1]
         prob.det <- param.parent[2]
                }


    labels <- matrix(nrow=1, ncol=5)
    # labels[1] is the fitted y category
    # labels[2] is sum(y == 0)   i.e. the "unseen" sites
    # labels[3] is sum(y >= 1)   i.e. the "seen" sites
    # labels[4] is prob.occ
    # labels[5] is prob.det
labels[1] <- ifelse(prob.occ >= parms$cutoff, 1, 0)
      labels[2] <- Ns[1]
      labels[3] <- sum(Ns[-1])
      labels[4] <- prob.occ
      labels[5] <- prob.det
 print(labels)
    dev <- ifelse(prob.occ >= parms$cutoff, Ns[1], sum(Ns[-1]))
    ret <- list(label=labels, deviance=dev)
 print("***** END: Evaluating *****")
    ret
    }

################ end Mark's eval() code
################

# The split function, where most of the work occurs.
#   Called once per split variable per node.
```

```
# If continuous=T
#   The actual x variable is ordered
#   y is supplied in the sort order of x, with no missings,
#   return two vectors of length (n-1):
#      goodness = goodness of the split, larger numbers are better.
#                 0 = couldn't find any worthwhile split
#        the ith value of goodness evaluates splitting obs 1:i vs (i+1):n
#      direction= -1 = send "y< cutpoint" to the left side of the tree
#                  1 = send "y< cutpoint" to the right
#         this is not a big deal, but making larger "mean y's" move towards
#         the right of the tree, as we do here, seems to make it easier to
#         read
# If continuous=F, x is a set of values defining the groups for an
#   unordered predictor.  In this case:
#       direction = a vector of length m= "# groups".
#       direction actually displays the names/labels for each group.
#        It asserts that the best split can be found by
#         lining the groups up in this order and going from left to right,
#         so that only m-1 splits need to be evaluated rather than 2^(m-1)
#       goodness = m-1 values here.
#
# The reason for returning a vector of goodness is that the C routine
#   enforces the "minbucket" constraint. It selects the best return value
#   that is not too close to an edge.

###############
###############  Mark's split() code

temp2 <- function(y, wt, x, parms, continuous) {
  #print("***** START: Splitting *****")
  if(parms$mygoodness==1){mygoodness=LRT.orig.parent}
  if(parms$mygoodness==2){mygoodness=LRT.each.parent}
  if(parms$mygoodness==3){mygoodness=LRT.parent.v.2daughters}
  if(parms$mygoodness==4){mygoodness=LRT.parent.v.daughter.v.daughter}

   idx <- order(x);  x <- x[idx];   y <- y[idx,];
     #Just in case ordering is not already done elsewhere
   y <- cbind(y)
     # If y is a vector, this allows me to calculate n using only one method
   n <- nrow(y)
   parent <- y   # In my code, the node being split is called the parent
    if (continuous) {     # continuous x variable

    # Get the goodness
          ## MAKE SURE IT IS A VECTOR!!
          ## Because rpart does the min. node size requirements elsewhere,
          ##   I just have to compute n-1 deviances here.
```

```
      possibles <- rep(0,n-1)
      direction <- rep(-1, n-1)
      prob.occ.L = prob.occ.R <- rep(0, n-1)

      for (i in 1:(n-1)) {
         left <- matrix(parent[1:i,], ncol=ncol(parent))
         right <- matrix(parent[(i+1):n,], ncol=ncol(parent))
         if(x[i]==x[i+1]) {next}
            ### NOT allowed to split up observations with the same x value
       info <- mygoodness(parent, left, right, orig.prob.det=parms$prob.det)
         possibles[i] <- info[1]  # test stat. for a likelihood ratio test
         prob.occ.L[i] <- info[2]
         prob.occ.R[i] <- info[3]

    # Get the direction   ALSO A VECTOR!!
      if(prob.occ.L[i] > prob.occ.R[i]){ direction[i] <- 1}
      # Compares occupancy probabilities, sends the higher one to the right
                           } # end 'for' loop
    goodness <- possibles
    ret <- list(goodness=goodness, direction=direction)
    #print("***** END: Splitting *****")
    ret
    }

   else {
# Categorical X variable
         #  we can order the categories by their means
         #   (i.e. estimated prob.occ values)
      #  then use the same code as for a non-categorical
ux <- sort(unique(x))
  # Sort does smallest to largest (either numerical or alphabetical)
      m <- length(ux)

      occs <- 0
      for(i in 1:m){
         group <- y[x==ux[i], ]
         Ns <- timesseen(group);
         param <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
                                    control=list(fnscale=-1), Ns=Ns )$par
         param <- backt(param)
         prob.occ <- param[1]
         occs[i] <- prob.occ
                    }  # end 'i' loop

ord <- order(occs)   #tells where each number belongs in order
   #  e.g. 2 1 4 3  means that the first number is second-lowest,
   #                   2nd number is smallest
```

```
    # Get the goodness
            ## MAKE SURE IT IS A VECTOR!!
            ## Because rpart does the minimum node size elsewhere,
            ## I just have to compute m-1 deviances here.
      possibles <- rep(0,m-1)
      prob.occ.L = prob.occ.R <- rep(0, m-1)

      for (i in 1:(m-1)) {
         left <- parent[ x<=ux[i], ]
         right <- parent[ x>ux[i], ]

      info <- mygoodness(parent, left, right, orig.prob.det=parms$prob.det)
         possibles[i] <- info[1]
         prob.occ.L[i] <- info[2]
         prob.occ.R[i] <- info[3]    }

    # Get the direction   ALSO A VECTOR!!
       direction <- ux[ord]

    goodness <- possibles
    ret <- list(goodness=goodness, direction=direction)
    #print("***** END: Splitting *****")
    ret
}
   }

###############  end Mark's split() code
###############

# The init function:
#   fix up y to deal with offsets
#   return a parms list--this can be passed in from the call to rpart(),
#      but it MUST be reproduced (or changed) in init()
#      parms includes cutoff (for predictions/labeling),
#                     occasions (# sampling times),
#                     prob.det (if specified by the user), and
#                     goodness (which imperfect detection method is used)
#   numresp is the number of values produced by the eval routine's "label"
#   numy is the number of columns for y
#   summary is a function used to print one line in summary.rpart
#      yval is the matrix "yval2" in tree$frame
#      each row contains predicted value, deviance, n, prob.occ, prob.det
#   text is a function used to put text on the plot in text.rpart
#  *NOTE: The split information printed is NOT controlled by the text
#              function in init()
#         Only the terminal node information comes from this text function
```

```
# In general, this function would also check for bad data, see
#           rpart.poisson for example

###############
############### begin Mark's init() code

temp3 <- function(y, offset, parms, wt) {
    print("***** START: Init *****")
    if (!is.null(offset)) y <- y-offset

    # IF method is orig.parent and prob.det is not specified:
      if(parms$mygoodness==1 & is.null(parms$prob.det)==TRUE) {
      # Calculate orig.prob.det from the very first parent node
      #   (i.e. all of the data)
          Ns <- timesseen(y)
          param.parent <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
                                  control=list(fnscale=-1), Ns = Ns)$par
          param.parent <- backt(param.parent)
          #orig.prob.occ <- param.parent[1]
          orig.prob.det <- param.parent[2]
          parms$prob.det <- orig.prob.det
              }  # end 'if' statement

    ret <- list(y=y, parms=parms, numresp=5, numy=parms$occasions,
        summary= function(yval, dev, wt, ylevel, digits ) {
          paste("predicted value=", yval[,1], "deviance=", dev, "prob.occ=",
              round(yval[,4],digits), "prob.det=", round(yval[,5],digits) )
                          }, #end summary

        text= function(yval, dev, wt, ylevel, digits, n, use.n ) {
              nclass <- (ncol(yval) - 1)/2
              group <- yval[, 1]
              counts <- yval[, 1 + (1:nclass)]
              if (!is.null(ylevel)) {group <- ylevel[group] }
              temp1 <- format(counts, digits)
              if (nclass > 1) { temp1 <- apply(matrix(temp1,
                              ncol = nclass), 1, paste, collapse = "/") }
              if (use.n) { out <- paste(format(group,
                                  justify = "left"), "\n", temp1, sep = "") }
              else {out <- format(group, justify = "left") }
              return(out)
                             },     #end text

        print= function(yval, ylevel, digits){
            if (is.null(ylevel)) {temp <- as.character(yval[, 1]) }
              else {temp <- ylevel[yval[, 1]] }
            nclass <- (ncol(yval) - 1)/2
```

```
            if (nclass < 5) {yprob <- format(yval[, 1 + nclass + 1:nclass],
                              digits = digits, nsmall = digits)}
          else {yprob <- formatg(yval[, 1 + nclass + 1:nclass], digits = 2)}
    if (is.null(dim(yprob))) {yprob <- matrix(yprob, ncol = length(yprob)) }
            temp <- paste(temp, " (", yprob[, 1], sep = "")
        for (i in 2:ncol(yprob)) temp <- paste(temp, yprob[, i], sep = " ")
            temp <- paste(temp, ")", sep = "")
            temp
                           }  #end print

         )  # end ret
    print("***** END: Init *****")
    ret
    }
```

2.7.2 Companion functions

```
# Calculates the total number of times the species was detected at each site
timesseen <- function(y){
   timesseen <- 0
   Ns <- rep(0, (ncol(y)+1)  )

      for (i in 1:nrow(y)){
           timesseen[i] <- sum(y[i, ]) }
      for (j in 1:length(Ns)){
           Ns[j] <- sum(timesseen==(j-1)) }
      # NOTE: Ns[1] = n0 = never seen, Ns[2] = n1 = seen once,
      #       Ns[3] = n2 = seen twice
            return(Ns) }    #end timesseen


# backtransforming parameters when using logistic representation
backt <- function(ln.param) {
           1/(1+exp(-ln.param))
              } #end backt
###########
# lnl using logistic parameterization
#    Used for any node when estimating both prob.occ and prob.det
lnl.t <- function(param, Ns){
             ln.Likelihood(backt(param), Ns)
                  } #end lnl.t

ln.Likelihood <- function(x, Ns){
      prob.occ <- x[1]
      prob.det <- x[2]
      k <- length(Ns)-1     # k = number of sampling occasions
      ln.like <- Ns[1]*log((1-prob.occ) + prob.occ*(1-prob.det)^k)
       # not occupied or occupied and seen 0 times
```

```
      for(i in 1:k){       # occupied, seen 'i' times out of 'k' possible
   ln.like <- ln.like + Ns[i+1]*log( choose(k,i) * prob.occ * (prob.det^i) *
                                    ((1-prob.det)^(k-i)) )
                    } # end 'i' loop
       return(ln.like) }  #end ln.Likelihood

###########
#  Need for nodes when I'm fixing prob.det while optimizing pi.occ
#  this occurs in orig.parent and each.parent
lnl.t.fixed <- function(param, prob.det, Ns){
             ln.Likelihood.fixed(backt(param), prob.det, Ns)
                   } #end lnl.t.fixed

ln.Likelihood.fixed <- function(x, prob.det, Ns){
      prob.occ <- x[1]
      k <- length(Ns)-1   # k = number of sampling occasions
      ln.like <- Ns[1]*log((1-prob.occ) + prob.occ*(1-prob.det)^k)
             # not occupied or occupied and seen 0 times
      for(i in 1:k){       # occupied, seen 'i' times out of 'k' possible
  ln.like <- ln.like + Ns[i+1]*log( choose(k,i) * prob.occ * (prob.det^i) *
                          ((1-prob.det)^(k-i)) )
                    } # end 'i' loop
       return(ln.like) }  #end ln.Likelihood

###########
# lnl for parent.v.2daughters
lnl.star.t <- function(param, Ns.left, Ns.right){
             ln.Likelihood.STAR2(backt(param), Ns.left, Ns.right)
                   } #end lnl.star.t


# lnl for parent.v.2daughters
ln.Likelihood.STAR2 <- function(x, Ns.left, Ns.right){
      prob.occ.L <- (x[1]); prob.occ.R <- (x[2]); prob.det.star <- (x[3]);
      k <- length(Ns.left)-1     # k = number of sampling occasions
      ln.like <- Ns.left[1]*log((1-prob.occ.L) +
                 prob.occ.L*(1-prob.det.star)^k) +
                 Ns.right[1]*log((1-prob.occ.R) +
                 prob.occ.R*(1-prob.det.star)^k )
        for(i in 1:k){ # occupied, seen 'i' times out of 'k' possible
        ln.like <- ln.like + Ns.left[i+1]*log( choose(k,i) * prob.occ.L *
                         (prob.det.star^i) * ((1-prob.det.star)^(k-i)) ) +
                             Ns.right[i+1]*log( choose(k,i) * prob.occ.R *
                            (prob.det.star^i) * ((1-prob.det.star)^(k-i)) )
                       } # end 'i' loop
       return(ln.like) }  #end ln.Likelihood.STAR2
```

```
###########
##############
##############  The following two functions are needed to label
#                       my print output properly

print.rpart <- function(x, minlength=0, spaces=2, cp,
               digits=getOption("digits"), ...) {
    if(!inherits(x, "rpart")) stop("Not legitimate rpart object")
    if (!is.null(x$frame$splits)) x <- rpconvert(x)  #help for old objects

    if (!missing(cp)) x <- prune.rpart(x, cp=cp)
    frame <- x$frame
    ylevel <- attr(x, "ylevels")
    node <- as.numeric(row.names(frame))
    depth <- tree.depth(node)
    indent <- paste(rep(" ", spaces * 32), collapse = "")
    #32 is the maximal depth
    if(length(node) > 1) {
        indent <- substring(indent, 1, spaces * seq(depth))
        indent <- paste(c("", indent[depth]), format(node), ")", sep = "")
    }
    else indent <- paste(format(node), ")", sep = "")

    tfun <- (x$functions)$print
    if (!is.null(tfun)) {
if (is.null(frame$yval2))
yval <- tfun(frame$yval,  ylevel, digits)
else    yval <- tfun(frame$yval2,  ylevel, digits)
}
    else yval <- format(signif(frame$yval, digits = digits))
    term <- rep(" ", length(depth))
    term[frame$var == "<leaf>"] <- "*"
    z <- labels(x, digits=digits, minlength=minlength, ...)
    n <- frame$n
    z <- paste(indent, z, n, format(signif(frame$dev, digits = digits)),
               yval, term)

    omit <- x$na.action
    if (length(omit))
    cat("n=", n[1], " (", naprint(omit), ")\n\n", sep="")
    else cat("n=", n[1], "\n\n")

    #This is stolen, unabashedly, from print.tree
    if (x$method=="class")
         cat("node), split, n, loss, yval, (yprob)\n")
   # NEW PART!!!
```

```
    if (x$method=="user"){ cat("node), split, n, deviance, yval,
              prob.occ, prob.det\n") }
   #####
    else cat("node), split, n, deviance, yval\n")
    cat("      * denotes terminal node\n\n")

    cat(z, sep = "\n")
    return(invisible(x))
    #end of the theft
    }

          # This one is located in treemisc.R
tree.depth <- function(nodes)
{
    depth <- floor(log(nodes, base = 2) + 1e-7)
    as.vector(depth - min(depth))
}
```

2.7.3 Four Proposed Methods

```
LRT.orig.parent <- function(parent, left, right, orig.prob.det){
     # PARENT
      Ns.parent <- timesseen(parent)
        # for example, n0 <- Ns[1]; n1 <- Ns[2]; n2 <- Ns[3]; ...

      # LEFT
      Ns.left <- timesseen(left)

      # RIGHT
      Ns.right <- timesseen(right)

       param.parent <- optim(c(0.5), lnl.t.fixed, gr=NULL, method="BFGS",
             control=list(fnscale=-1), prob.det = orig.prob.det,
             Ns = Ns.parent )$par
         param.parent <- backt(param.parent)
         prob.occ <- param.parent[1]

       param.left <- optim(c(0.5), lnl.t.fixed, gr=NULL, method="BFGS",
                    control=list(fnscale=-1), prob.det = orig.prob.det,
                     Ns=Ns.left)$par
         param.left <- backt(param.left)
         prob.occ.L <- param.left[1]

       param.right <- optim(c(0.5), lnl.t.fixed, gr=NULL, method="BFGS",
                   control=list(fnscale=-1), prob.det = orig.prob.det,
                    Ns=Ns.right)$par
         param.right <- backt(param.right)
```

```
        prob.occ.R <- param.right[1]
       upper <- ln.Likelihood(c(prob.occ, orig.prob.det), Ns.left) +
                 ln.Likelihood(c(prob.occ, orig.prob.det), Ns.right)
       lower <- ln.Likelihood(c(prob.occ.L, orig.prob.det), Ns.left) +
                 ln.Likelihood(c(prob.occ.R, orig.prob.det), Ns.right)
       test.stat <- -2*(upper-lower)
       out <- c(test.stat, prob.occ.L, prob.occ.R, orig.prob.det)
       return(out)  }


LRT.each.parent <- function(parent, left, right, orig.prob.det){
      # LEFT
      Ns.left <- timesseen(left)

      # RIGHT
      Ns.right <- timesseen(right)

       param.parent <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
                    control=list(fnscale=-1),Ns = (Ns.left+Ns.right) )$par
        param.parent <- backt(param.parent)
        prob.occ <- param.parent[1]
        prob.det <- param.parent[2]

       param.left <- optim(c(0.5), lnl.t.fixed, gr=NULL, method="BFGS",
        control=list(fnscale=-1), prob.det = prob.det, Ns=Ns.left)$par
         param.left <- backt(param.left)
         prob.occ.L <- param.left[1]

       param.right <- optim(c(0.5), lnl.t.fixed, gr=NULL, method="BFGS",
            control=list(fnscale=-1), prob.det = prob.det, Ns=Ns.right)$par
         param.right <- backt(param.right)
         prob.occ.R <- param.right[1]

       upper <- ln.Likelihood(c(prob.occ, prob.det), Ns.left) +
                 ln.Likelihood(c(prob.occ, prob.det), Ns.right)
       lower <- ln.Likelihood(c(prob.occ.L, prob.det), Ns.left) +
                 ln.Likelihood(c(prob.occ.R, prob.det), Ns.right)
       test.stat <- -2*(upper-lower)
       out <- c(test.stat, prob.occ.L, prob.occ.R, prob.det)
       return(out)  }


LRT.parent.v.daughter.v.daughter <- function(parent, left, right,
                                                    orig.prob.det){
      # LEFT
      Ns.left <- timesseen(left)
```

```
      # RIGHT
      Ns.right <- timesseen(right)

      param.parent <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
               control=list(fnscale=-1), Ns= (Ns.left + Ns.right) )$par
        param.parent <- backt(param.parent)
        prob.occP <- param.parent[1]
        prob.detP <- param.parent[2]

      param.left <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
        control=list(fnscale=-1), Ns=Ns.left )$par
        param.left <- backt(param.left)
        prob.occ.L <- param.left[1]
        prob.det.L <- param.left[2]

      param.right <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
        control=list(fnscale=-1), Ns=Ns.right )$par
        param.right <- backt(param.right)
        prob.occ.R <- param.right[1]
        prob.det.R <- param.right[2]

       upper <- ln.Likelihood(c(prob.occP, prob.det.L), Ns.left) +
                 ln.Likelihood(c(prob.occP, prob.det.R), Ns.right)
       lower <- ln.Likelihood(c(prob.occ.L, prob.det.L), Ns.left) +
           ln.Likelihood(c(prob.occ.R, prob.det.R), Ns.right)
       test.stat <- -2*(upper-lower)
        out <- c(test.stat, prob.occ.L, prob.occ.R, prob.det.L, prob.det.R)
        return(out)  }


LRT.parent.v.2daughters<-function(parent, left, right, orig.prob.det){
      # LEFT
      Ns.left <- timesseen(left)

      #RIGHT
      Ns.right <- timesseen(right)

        param.parent <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
          control=list(fnscale=-1), Ns = (Ns.left + Ns.right) )$par
        param.parent <- backt(param.parent)
        prob.occ.P <- param.parent[1]
        prob.det.P <- param.parent[2]

       param.star <- optim(c(0.5, 0.5, 0.5), lnl.star.t, gr=NULL,
          method="BFGS", control=list(fnscale=-1), Ns.left=Ns.left,
                 Ns.right=Ns.right)$par
        param.star <- backt(param.star)
```

```
      pi.L <- param.star[1];
      pi.R <- param.star[2];
      prob.det.star <- param.star[3]

       param.upper <- c(prob.occ.P, prob.occ.P, prob.det.star)
       param.lower <- c(pi.L, pi.R, prob.det.star)
      upper <- ln.Likelihood.STAR2(param.upper, Ns.left, Ns.right)
      lower <- ln.Likelihood.STAR2(param.lower, Ns.left, Ns.right)
      test.stat <- -2*(upper-lower)
      out <- c(test.stat, param.lower)
      return(out)  }
```

2.7.4 Run code

```
#Mark McKelvey
plovers <- read.csv("C://Documents and Settings/Owner/My Documents/
             Research Part I/McKelvey data.csv", header=T)

attach(plovers)
  library(rpart);

############################
############################
  X2002 <- 0
       for (i in 1:81) { X2002[i] <- sum(X2002.1[i] + X2002.2[i])
                         ifelse(X2002[i]>=1, X2002[i] <- 1, X2002[i] <- 0) }
  data.2002 <- data.frame(X2002, A02, X02PARA, X02PSI)
                     #for matching with rpart
    colnames(data.2002) <- c("X2002", "AREA", "PARA", "PSI")
  doubledown <- cbind(X2002, X2002, A02, X02PARA, X02PSI)
     #### REMINDER: FOR my personal (self-written) optim 4 code,
                    completedata.2002 NEEDS TO BE A MATRIX
     ####  THE DATA FRAME IS NEEDED FOR USER.SPLITS
  completedata.mat.2002 <- cbind(X2002.1, X2002.2, A02, X02PARA, X02PSI)
        #for incorporating detection
    colnames(completedata.mat.2002) <- c("X2002.1", "X2002.2",
                                         "AREA", "PARA", "PSI")
  completedata.df.2002 <- as.data.frame(completedata.mat.2002)
  detach(plovers)

########################################
########################################

#Source code progression:
  #source("C://Documents and Settings/Owner/My Documents/Research Part I/
                          MY rpart functions with Optim 4.R")
```

```
        #This includes  myrpartLRT with optim() rather than direct MLE's
        # *Note:  myrpartLRT is the one where I wrote my own massive rpart
        #    function, which matches rpart() for continuous covariates
  #source("C://Documents and Settings/Owner/My Documents/Research Part I/
   MY attempt at user splits.R")
        #This one has the actual user splits code.
        #  NOTE:  It is NOT used in the paper (except for graphing)
        #  --while estimates will match, the extra internal pruning used
        #      within rpart() causes the tree structure to not be as
        #       "complete" as the other** results
      #    3/3/13 ** "Other" refers to fit.02
      #               (rpart(), 1 occasion, Gini, perfect detection)
        #                and to alt.02orig
        # Strangely enough, the naive method specifying Gini as the
        # splitting method does NOT exhibit the same problem...
################################
################################

# The user splits results:
   source("E:/Research/MY attempt at user splits.R")
   source("C://Documents and Settings/Owner/My Documents/Research Part I/
               MY attempt at user splits.R")

    alist <- list(eval=temp1, split=temp2, init=temp3)
  parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=1)
    # mygoodness:  1=orig.parent, 2=each.parent, 3=p.v.2d, 4=p.v.d.v.d.
  fit.user02 <- rpart( cbind(X2002.1, X2002.2) ~ AREA + PARA + PSI,
          data=completedata.df.2002, method=alist, parms=parms, cp=0.005 )
    # Note the use of the data frame here (completedata.df.2002)...
    #  problems exist if using a matrix (column references in code somehow)
################################

# The "myrpartLRT" code (personally-written to emulate rpart while
                           # incorporating detection)
#    results:
  source("C://Documents and Settings/Owner/My Documents/
              Research Part I/MY rpart functions with Optim 4.R")

   # Well, technically this one (fit.02) doesn't use my own code
   #        (it is for perfect detection):
   fit.02 <- rpart(as.factor(X2002) ~ AREA + PARA + PSI,
                   parms=list(split="gini"), data=data.2002, cp=0.005)
             # Now Y is categorical, 1 occasion
             # Could also say "method=class" in the call to rpart.
             # Should do the same as if y is a factor.
  fit.02
  par(mfrow=c(1,2), xpd=NA)
```

```
  plot(fit.02, main="Rpart")
  text(fit.02, use.n=TRUE)
    # Note the use of completedata.mat.2002
    #  (must be in a matrix form for my pers. code for column references)
  #alt.02 <- myrpart1(data.2002, deviance=Gini.det, prob.det=1, cp=.005)
   alt.02pv2d <- myrpartLRT(completedata.mat.2002,
                   deviance=LRT.parent.v.2daughters, occasions=2, cp=.005)
   alt.02pvdvd <- myrpartLRT(completedata.mat.2002,
    deviance=LRT.parent.v.daughter.v.daughter, occasions=2, cp=.005)
   alt.02each <- myrpartLRT(completedata.mat.2002, deviance=LRT.each.parent,
     occasions=2, cp=.005)
   alt.02orig <- myrpartLRT(completedata.mat.2002, deviance=LRT.orig.parent,
     occasions=2, cp=.005)
   fit.02naive <- myrpartLRT(doubledown,
            deviance=LRT.orig.parent, occasions=2, cp=.005)
      # Note: doubledown is a matrix.  Even though this still allows
      #        detection to be chosen, it is estimated as .999999,
      #         so it should be fine
   #class(alt.02) <- "rpart"
    details(alt.02orig)  # My personal plotting and text function

    alt.02orig$mysplits
    alt.02each$mysplits
    alt.02pv2d$mysplits
    alt.02pvdvd$mysplits  # different from other 3
    fit.02naive$mysplits

    myaic(alt.02orig, naive=FALSE)
    myaic(alt.02each, naive=FALSE)
    myaic(alt.02pv2d, naive=FALSE)
    myaic(alt.02pvdvd, naive=FALSE)
    myaic(fit.02naive, naive=TRUE)
    myaic.naivetree(fit.02)   # For use with an rpart()-created tree object
                              #  This corresponds to using fit.02naive.


#####################################################
#####################################################

##  test1 from 3/3/13
test1 <- read.csv("C://Documents and Settings/Owner/My Documents/
                   Research Part I/Dixon test1 data 3_3_13.csv", header=T)
  Y1or2 <- ifelse(test1$y1==1 | test1$y2==1, 1, 0)
test1.naive <- data.frame(test1, Y1or2)
test1.df <- test1;
test1.mat <- as.matrix(test1);
   #1) Set up for naive (perfect detection) and my original code
```

```
#    (imperfect detection)

# Well, technically this one doesn't use my own code
#    (it is for perfect detection):
fit.test1 <- rpart(as.factor(Y1or2) ~ x1 + x2 + x3,
                   parms=list(split="gini"), data=test1.naive, cp=0.005)
           # Now Y is categorical, 1 occasion
           # Could also say "method=class" in the call to rpart.
           # Should do the same as if y is a factor.
   fit.test1
   par(mfrow=c(1,2), xpd=NA)
   plot(fit.test1, main="Rpart, piDet=1")
   text(fit.test1, use.n=TRUE)


#2) Using user splits (first to check results with nominal categorical),
#   but also to get a tree object that will work well with graphing,
#   trimming, etc.
  source("C://Documents and Settings/Owner/My Documents/Research Part I/
   MY attempt at user splits.R")

    alist <- list(eval=temp1, split=temp2, init=temp3)        #
      # mygoodness:  1=orig.parent, 2=each.parent, 3=p.v.2d, 4=p.v.d.v.d.
parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=1)
  fit.user.test1.orig <- rpart( cbind(y1, y2) ~ x1 + x2 + x3,
                    data=test1.df, method=alist, parms=parms, cp=0.005 )
     parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=2)
  fit.user.test1.each <- rpart( cbind(y1, y2) ~ x1 + x2 + x3,
                    data=test1.df, method=alist, parms=parms, cp=0.005 )
     parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=3)
  fit.user.test1.pv2d <- rpart( cbind(y1, y2) ~ x1 + x2 + x3,
                    data=test1.df, method=alist, parms=parms, cp=0.005 )
     parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=4)
  fit.user.test1.pvdvd <- rpart( cbind(y1, y2) ~ x1 + x2 + x3,
                    data=test1.df, method=alist, parms=parms, cp=0.005 )

     options(digits=7)  # the user splits file changes it to 4,
      # but this screws up the naive picture
     par(mfrow=c(1,5), xpd=NA)
    plot(fit.test1, main="Naive");
    text(fit.test1, use.n=T)
      plot(fit.user.test1.orig, main="Orig");
      text(fit.user.test1.orig, use.n=T)
      plot(fit.user.test1.each, main="Each");
      text(fit.user.test1.each, use.n=T)
    plot(fit.user.test1.pv2d, main="pv2d");
    text(fit.user.test1.pv2d, use.n=T)
```

```
    plot(fit.user.test1.pvdvd, main="pvdvd");
    text(fit.user.test1.pvdvd, use.n=T)

  # To prune trees via display (must use a version of R later than 3.1??
  #     Maybe not...)
     library(rpart.plot)
     test1.naive.trimmed <- prp(fit.test1, snip=TRUE)$obj
     test1.pv2d.trimmed <- prp(fit.user.test1.pv2d, snip=TRUE)$obj
       par(mfrow=c(2,2), xpd=NA)
       par(mar=c(0.2, 0.2, 4, 0.2))
       plot(test1.naive.trimmed, main=" Test1 Naive")
       text(test1.naive.trimmed, use.n=T)
       plot(test1.pv2d.trimmed, main="Test1 pv2d")
       text(test1.pv2d.trimmed, use.n=T)

  #3) Set up my four methods with my original code.
  #    Also provides AIC values
     source("C://Documents and Settings/Owner/My Documents/Research Part I/
      MY rpart functions with Optim 4.R")           #

    # Note the use of test1.mat (must be a matrix for my personal code)
    fit.test1.pv2d <- myrpartLRT(test1.mat,
      deviance=LRT.parent.v.2daughters, occasions=2, cp=.005)
    fit.test1.pvdvd <- myrpartLRT(test1.mat,
      deviance=LRT.parent.v.daughter.v.daughter, occasions=2, cp=.005)
    fit.test1.each <- myrpartLRT(test1.mat, deviance=LRT.each.parent,
      occasions=2, cp=.005)
    fit.test1.orig <- myrpartLRT(test1.mat, deviance=LRT.orig.parent,
      occasions=2, cp=.005)
    #class(test1.orig) <- "rpart"  # if I were going to use plot() directly

      par(mfrow=c(1,5), xpd=NA)
      plot(fit.test1, main="Rpart, pi.det=1")
      text(fit.test1, use.n=T)
      options(digits=7)
      details(fit.test1.orig)  # My personal plotting and text function
      details(fit.test1.each)
      details(fit.test1.pv2d)
      details(fit.test1.pvdvd)

      myaic(fit.test1.orig, naive=FALSE)
      myaic(fit.test1.each, naive=FALSE)
      myaic(fit.test1.pv2d, naive=FALSE)
      myaic(fit.test1.pvdvd, naive=FALSE)


## test2 from 3/3/13
```

```
test2 <- read.csv("C://Documents and Settings/Owner/My Documents/
                   Research Part I/Dixon test2 data 3_3_13.csv", header=T)
  Y1or2.test2 <- ifelse(test2$y1==1 | test2$y2==1, 1, 0)
test2.naive <- data.frame(test2, Y1or2.test2)
test2.df <- test2;
test2.mat <- as.matrix(test2);
  # test2$x4b <- as.factor(test2$x4b); test2$x5b <- as.factor(test2$x5b);
  #  test2$x6b <- as.factor(test2$x6b);

#1) Set up for naive (perfect detection)

   # Well, technically this one doesn't use my own code
   #   (it is for perfect detection):
fit.test2 <- rpart(as.factor(Y1or2.test2) ~ x1 + x2 + x3 + x4b + x5b + x6b,
    parms=list(split="gini"), data=test2.naive, cp=0.005)
              # Now Y is categorical, 1 occasion
              # Could also say "method=class" in the call to rpart.
              # Should do the same as if y is a factor.
   fit.test2
   par(mfrow=c(1,2), xpd=NA)
   plot(fit.test2, main="Rpart, piDet=1")
   text(fit.test2, use.n=TRUE)

#2)Using user splits (first to check results with nominal categorical),
  #   but also to get a tree object that will work well with graphing,
  #   trimming, etc.
    source("C://Documents and Settings/Owner/My Documents/Research Part I/
     MY attempt at user splits.R")

      alist <- list(eval=temp1, split=temp2, init=temp3)        #
        # mygoodness:  1=orig.parent, 2=each.parent, 3=p.v.2d, 4=p.v.d.v.d.
  parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=1)
fit.user.test2.orig <- rpart(cbind(y1, y2) ~ x1 + x2 + x3 + x4b + x5b + x6b,
     data=test2.df, method=alist, parms=parms, cp=0.005 )
       parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=2)
fit.user.test2.each <- rpart(cbind(y1, y2) ~ x1 + x2 + x3 + x4b + x5b + x6b,
     data=test2.df, method=alist, parms=parms, cp=0.005 )
       parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=3)
fit.user.test2.pv2d <- rpart(cbind(y1, y2) ~ x1 + x2 + x3 + x4b + x5b + x6b,
     data=test2.df, method=alist, parms=parms, cp=0.005 )
       parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=4)
fit.user.test2.pvdvd <- rpart(cbind(y1, y2) ~ x1 + x2 + x3 + x4b + x5b + x6b,
    data=test2.df, method=alist, parms=parms, cp=0.005 )

        options(digits=7)  # the user splits file changes it to 4,
                           # but this screws up the naive picture
        par(mfrow=c(1,2), xpd=NA)
```

```
plot(fit.test2, main="Naive");
text(fit.test2, use.n=T)
  plot(fit.user.test2.orig, main="Orig");
  text(fit.user.test2.orig, use.n=T)
  plot(fit.user.test2.each, main="Each");
  text(fit.user.test2.each, use.n=T)
plot(fit.user.test2.pv2d, main="pv2d");
text(fit.user.test2.pv2d, use.n=T)
plot(fit.user.test2.pvdvd, main="pvdvd");
text(fit.user.test2.pvdvd, use.n=T)

  # To prune trees via display (must use a version of R later than 3.1??
  #   Or not...)
     library(rpart.plot)
     test2.naive.trimmed <- prp(fit.test2, snip=TRUE)$obj
     test2.orig.trimmed <- prp(fit.user.test2.orig, snip=TRUE)$obj
 par(mfrow=c(1,2), xpd=NA)
       plot(test2.naive.trimmed, main="Test2 Naive")
       text(test2.naive.trimmed, use.n=T)
       plot(test2.orig.trimmed, main="Test2 orig.parent")
       text(test2.orig.trimmed, use.n=T)



#3) Set up for alt. methods with my original code.
#      Also to get AIC values.
  source("C://Documents and Settings/Owner/My Documents/Research Part I/
   MY rpart functions with Optim 4.R")           #

  # Note the use of test2.mat (must be a matrix for for my personal code
 fit.test2.pv2d <- myrpartLRT(test2.mat, deviance=LRT.parent.v.2daughters,
   occasions=2, cp=.005)
 fit.test2.pvdvd <- myrpartLRT(test2.mat,
    deviance=LRT.parent.v.daughter.v.daughter, occasions=2, cp=.005)
 fit.test2.each <- myrpartLRT(test2.mat, deviance=LRT.each.parent,
   occasions=2, cp=.005)
 fit.test2.orig <- myrpartLRT(test2.mat, deviance=LRT.orig.parent,
   occasions=2, cp=.005)
 #class(test2.orig) <- "rpart"  # if I were going to use plot() directly

  par(mfrow=c(1,5), xpd=NA)
  plot(fit.test2, main="Naive, pi.det=1")
  text(fit.test2, use.n=T)
  details(fit.test2.orig)  # My personal plotting and text function
  details(fit.test2.each)
  details(fit.test2.pv2d)
  details(fit.test2.pvdvd)
```

```
  myaic(fit.test2.orig, naive=FALSE)
  myaic(fit.test2.each, naive=FALSE)
  myaic(fit.test2.pv2d, naive=FALSE)
  myaic(fit.test2.pvdvd, naive=FALSE)

    # To graph the test1 and test2 trees for the Part I paper
       par(mfrow=c(2,2), xpd=NA)
      par(mar=c(0.2, 0.3, 5, 0.3))   # Bottom, L, Top, R

#pdf("C://Documents and Settings/Owner/My Documents/Research Part I/
#   typed summary/test1and2_trimmed_file.pdf", width=0.01, height=0.01)
      #par(mfrow=c(2,2), xpd=NA)
      plot(test1.naive.trimmed, main=" Test1 Naive")
      text(test1.naive.trimmed, use.n=T)
      plot(test1.pv2d.trimmed, main="Test1 pv2d")
      text(test1.pv2d.trimmed, use.n=T)
      plot(test2.naive.trimmed, main="Test2 Naive")
      text(test2.naive.trimmed, use.n=T)
      plot(test2.orig.trimmed, main="Test2 orig.parent")
      text(test2.orig.trimmed, use.n=T)
     #dev.off()
     #dev.list()
```

CHAPTER 3. Incorporating Dependence into Classification and Regression Trees for Occupancy Modeling

Classification and regression trees (CART) are a flexible, frequently-used method for modeling probabilities of events. Many studies include a cluster-type sampling design where there is a clear spatial correlation between sampling locations. This correlation causes the variance of the node occupancy estimates in CART to be biased. We suggest a generalized estimating equation (GEE)-based approach in which the naïve variance estimates (calculated as if all locations were independent) are "corrected" based on the data available in each parent node of the tree. The corrected variance estimates are then used to revise the binary-split decision criterion of the tree. We demonstrate this method using data from a study on rats and also from a study on bird occurrences in Oregon.

3.1 Introduction

A classification and regression tree (CART) is a flexible alternative to linear models for regression and logistic models for classification. In CART, individuals (which could be spatial locations or sites) are separated into groups using covariate information (Breiman et al 1984, De'ath and Fabricius 2000). Each group is then identified by a predicted value of the response variable. CART, like logistic regression, is flexible in that it does not require any distributional assumptions and it allows the use of both categorical and quantitative variables. Unlike logistic regression, it can model interactions and higher-order terms with relative ease–there are no restrictions of additivity or linearity. Another advantage of CART is its ability to handle missing values. Whereas logistic regression would discard any individual with data missing from one of its covariates, CART can still use that individual's data to help formulate the

model (Clark and Pregibon 1992, De'ath and Fabricius 2000, Breiman et al. 1984). These properties, along with its relative ease of construction and interpretation, have lead many researchers to use CART as their tool of choice for modeling. De'ath and Fabricius (2000) used CART to analyze survey data on abundances of soft coral in the Great Barrier Reef. Bourg et al. (2005) used a combination of methods, including CART, to predict habitat for the rare forest herb turkeybeard. Molinaro et al (2004) adapt CART to censored data, touching on both univariate and multivariate responses. Segal (1992) discusses tree-based regression with longitudinal data, and even talks briefly about alternative tree-splitting algorithms.

During the creation of a binary tree, CART attempts to partition the data into homogeneous groups. To do this, CART uses a calculated measure based on impurity (or a statistical deviance) in order to rank each of the possible splits at a node. Some common measures of impurity for a classification tree (Brieman et al. 1984) are Sum of Squares, Entropy, Misclassification, Twoing, and the Gini index, which is often the default measure for splitting in a classification setting. In a study where the data consists of Seen(Yes)/Not Seen(No) responses, the Gini Index defines impurity at a node with n observations as

$$(2 \times \frac{\#Yes}{n}) * (1 - \frac{\#Yes}{n}) \tag{3.1}$$

The collective deviance at the two daughter nodes is the sum of (3.1) calculated at each node. Then the drop in deviance of the proposed split is the difference between the deviance of the parent node and the combined deviance of the two daughter nodes. In general, the proposed split with the largest drop in deviance is chosen.

Models often assume that the observations in the study are independent. However, some studies involve observations that are clustered (either spatially or otherwise related). The characteristics of clustered individuals are likely to be correlated in some way. Therefore, any model examining data where independence is in doubt should account for that [possible] correlation.

Generalized linear models and generalized linear mixed-models are often used to model clustered data. Both of these model types are able to account for correlation between individual data, either explicitly (GLMM's) or through model extensions (e.g. generalized-estimating

equations (GEE's) for GLM's). In a general review of methods for modeling spatially autocorrelated data, Dormann et al (2007) indicate that GLMM's and GEE's are a good, flexible choice for modeling binary response data such as presence/absence data.

Clustered data affects the variance of parameter estimates. If clustering has a constant effect on the variance, then a test statistic approach to splitting would just be a re-scaled version of a deviance method, and would have no effect on the ranking of potential splits. We will show through examples that clustering does not have a constant effect. Depending on the situation (where clustered patches are in relation to each other within the CART tree), clustering may either increase or decrease the variance (relative to the naïve case assuming independence), and it may do so with varying orders of magnitude.

CART has been used in analysis of occupancy studies, which can be useful for describing species-habitat relationships, helping with monitoring programs, or as an alternative to abundance sampling. Given covariate information, CART could be used to either predict occupancy status or the occupancy probability of a location. In spite of the potential for correlation, CART has been used to analyze data that may be clustered. Some examples include Castellón and Sieving (2006), who used classification tree analysis to develop predictive patch occupancy models for an avian species with limited dispersal ability; Murray et al (2008), who relate occupancy to ecological scale in a case study of rock wallabies; and Bel et al (2009), who use a CART algorithm for spatial data with data from a study on presence/absence of tree species.

Previous methods have been developed which attempt to adjust the CART process for correlated data, including one proposal by Li and Claramunt (2006) which replaces the traditional entropy in a tree with a spatial entropy measure, and another by Bel at al (2009) which examines spatial estimates of the quantities involved in the construction of the discriminant rule. Both of these take a different, less general approach than we do. In a more closely related paper, Sela and Simonoff (2012) have proposed an estimation method which blends mixed-effects models for longitudinal and clustered data into CART.

We propose a method to incorporate dependence into the decision-making process of CART in which random effects are used to help model the correlation among related observations. To more accurately assess the variance of the point estimates when there is correlation involved, we

take a generalized estimating equation (GEE) approach. First, we calculate the point estimates and their corresponding variances under the assumption of independence, and then we "correct" the naïve variance estimates by using the correlation found in the sample data set. We base our correction ("patch-up") factor on an analogy with the Huber-White consistent variance estimator.

The suggested method is applied to two data sets: an experimental study on cavities in rats' teeth and an observational study on breeding birds. In both cases the results are compared to those of a naïve approach assuming independence between all patches.

3.1.1 Examples

3.1.1.1 Motivating Example 1: The Rat Data

The first example is taken from Andrews and Herzberg (1985, Table 43.1). One hundred and twenty rats (of which 117 survived) were randomly assigned to one of eight diets. After completing the feeding period, the rats were sacrificed and their teeth were removed and stained. Twenty-eight occlusal surfaces in each rat were examined for cavities and scored according to severity of decay. The response values were either 0,1,2, or 3, where 0 represents no decay, and 1, 2, and 3 represent increasing levels of decay. We reduce the response data to 0 (for decay values of 0 and 1) and 1 (for decay values of 2 and 3).

3.1.1.2 Motivating Example 2: The Bird Data.

Our example is taken from a study done on breeding birds in Oregon (McGarigal and McComb 1995). The study design is a cluster-type survey where patches are sampled within sites. Specifically, there are 3 geographic basins which contain 10 sites apiece; each site consists of a cluster of 30+ patches. At each patch, an observer was sent out four times during a period from May to mid-July. During each survey, the observer recorded a count of each bird species detected. Also recorded were many covariate values corresponding to the different spatial scales; the majority of these described each patch.

Of primary interest to us is the occupancy of each patch, taking note of how the occupancy

of one patch is likely correlated to the occupancy of other patches located in the same site. Each basin was sampled in a different year, leading us to only use the data from a single basin, Drift Creek, so that we could ignore changes in detection (and occupancy) from year to year and from basin to basin. We chose the Golden-crowned kinglet for our analysis. We preferred a species common enough to occupy a decent number of patches, yet not so common that detection is a non-issue. Overall, the Golden-crowned kinglet was seen at 57% of all patches.

3.2 Methods

3.2.1 Methods for Rats Example–dependence only

Each response value corresponds to an occlusal surface within a rat, so each rat is thought of as a subject that consists of a cluster of 28 related observations.
Assuming that correlation exists between responses of observations on the same subject means that the variance of an estimate of decay (in the CART tree) is likely to be different than the variance of an estimate obtained under the assumption of independence. Exactly how different that variance is depends on the choice of correlation structure and the specific observations involved in the proposed split being examined in CART.

Our quantity of interest is $Var(\pi_L - \pi_R)$, the variance of the difference in decay probabilities between the left (L) or right (R) daughter nodes. The estimate of this variance will be used to compute a Wald test statistic to help determine the best possible split from each parent node.

$$H_0 : \pi_L = \pi_R$$

$$\text{test stat} = \frac{\hat{\pi}_L - \hat{\pi}_R}{\sqrt{Var(\hat{\pi}_L - \hat{\pi}_R)}} \tag{3.2}$$

Under the null hypothesis, the variance of each observation from the same subject is the same (i.e. the observation-level variance of each observation will be the same regardless of which daughter node (Left or Right) the observation is sent to next, because $\pi_L = \pi_R = \pi_P$ under H_0).

Letting V_0 represent $Var(\hat{\pi}_P)$, we estimated the naïve (under independence) variance $Var(\pi_L - \pi_R)$ by adjusting V_0 for the size of each daughter node as follows:

$$Var(\pi_L - \pi_R) = Var(\pi_L) + Var(\pi_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) \tag{3.3}$$

In a Bernoulli setting where an observation is either damaged or not, the variance of the observation is simply $\pi(1-\pi)$. When y_{ij} is the observed response of observation j in subject i, then $\bar{Y} = \hat{\pi}$ and the node estimate has a Binomial variance. $V_0 = \frac{\hat{\pi}_P(1-\hat{\pi}_P)}{n_L+n_R}$ in the above equation. Using $Var(\hat{\pi}_P)$ as the basis of an estimate of $Var(\hat{\pi}_L)$ and $Var(\hat{\pi}_R)$, as in Equation 3.3, is done in the interest of increasing speed. Without using $Var(\hat{\pi}_P)$, we would need to compute $Var(\hat{\pi}_L)$ and $Var(\hat{\pi}_R)$ for every potential split from a parent node. When the number of covariates or individuals is large, these multiple calculations can have a noticeable adverse effect on computation time.

3.2.1.1 Patchup Factor

To estimate the correct variance of $Var(\pi_L - \pi_R)$ (i.e. a variance which accounts for spatial dependence), we take a generalized estimating equation (GEE)-based approach: First estimate the naïve variance under the assumption of independence, and then "correct" that naïve estimate based on the sample observations in the data. While there *is* an explicit sandwich variance estimator used with GEE's, the amount of computing time required to run those calculations can be very lengthy (CART calculates a test for every possible split from each parent node, which would necessitate the use of a sandwich estimator many times (as many as *#covariates * #individuals in the node*)). The naïve (assuming independence) variance is simply $Var(\pi_L - \pi_R) = Var(\pi_L) + Var(\pi_R)$, which we estimated as shown above.

3.2.1.2 Equal-variances patchup factor

The corrected variance is calculated by $k * Var(\pi_L - \pi_R)$. The patchup factor, k, needs to not only be a good estimate, but it needs to be calculated quickly, as it will be calculated for every proposed split, not just for every parent node.

There are already many methods for adjusting variance when correlated binary data is present.

For example, Bahjat and Liang (1992), Le Cessie and Van Houwelingen (1994), Liang et al (1992), Neuhaus et al (1991), and Prentice (1988) all discuss various methods, but many of these methods would be fairly intensive computational processes when applied to CART.
We started with a simple linear model $Y = X\beta$, where X is an n x 2 matrix of 0's and 1's that identifies which daughter node each observation goes to in a proposed split, and β represents the parameters in question (for the two daughter nodes), π_L and π_R. Using ordinary least squares (OLS) to solve the linear model, we find that $\hat{\beta} = (X'X)^{-1}(X'Y)$ and $Var(\hat{\beta}) = (X'X)^{-1}(X'\Sigma X)(X'X)^{-1}$. Without correlated data, Σ is a diagonal matrix of observation variances.

Conceptually, we fit a linear model to π_{ij}, the probability of damage for observation j on subject i:

$$\pi_{ij} = X\beta + subject_i + \epsilon_{ij} \tag{3.4}$$

$$\text{where } subject_i \sim N(0, {\sigma^2}_{subject}),$$
$$\epsilon_{ij} \sim N(0, {\sigma^2}_{obs}),$$
$$\text{and } \beta' = [\pi_L \ \pi_R]$$

Applying the OLS thinking to our problem, we can calculate a relatively fast approximation of the correct (non-independence) $Var(\hat{\pi}_L) - \hat{\pi}_R)$ by using the patchup factor k, where

$$k = \frac{C(X'X)^{-1}(X'VX)(X'X)^{-1}C'}{C(X'X)^{-1}C'} \tag{3.5}$$

is the ratio of the OLS variance *with* correlated observations to the variance *without* correlation.

—- C is [1 -1] to represent $\pi_L - \pi_R$

—- X records the destination (left daughter node or right daughter node) of each observation in the proposed split

—- V is the block-diagonal correlation matrix (each block represents a subject), and is formed using the estimated within-subject correlation, $\hat{\rho}$, along with knowing the total number of subjects and observations per subject that exist in the parent node.

For this analysis, we assumed equal (i.e. exchangeable) correlation between all observations on the same subject. Thus, the correlation between two observations on the same subject (under H_0) can be written as:

$$Cor(\pi_{ij}, \pi_{ik}) = \frac{{\sigma^2}_{subject}}{{\sigma^2}_{subject} + {\sigma^2}_{obs}} = p$$

—- the within-subject correlation of observation decay probabilities (used to construct V) is estimated from the data in the parent node (this is explained in Implementation)

The patchup factor technically includes the covariance matrix in the numerator and the variance of an observation (${\sigma^2}_{subject}+{\sigma^2}_{obs}$) in the denominator. Because of the equal-variances assumption, all of the off-diagonal entries in the covariance matrix are exactly the same, which allowed us to pull out and then cancel the common factor (${\sigma^2}_{subject} + {\sigma^2}_{obs}$) from the top and bottom. This left us with the correlation matrix in the numerator (ones on the diagonal and ρ on the non-zero off-diagonals), while the covariance matrix on the bottom (which assumed independence) became the identity matrix, thus reducing the denominator to what is shown above in equation (3.5).

We illustrate the patchup factor for three examples. Each example shows calculations of $Var(\hat{\pi}_L - \hat{\pi}_R)$ under H_0/EQUAL variances, built from $\hat{\pi}_P = 0.7$ and $\hat{p} = 0.35$. As a whole, these examples demonstrate the idea that the patchup factor depends on how the observations within a subject are split between the two daughter nodes. Using 2 subjects with 4 observations per subject, the covariance matrix for each example (under H_0) is identical:

$$\Sigma = 0.21 * \begin{bmatrix} 1 & .35 & .35 & .35 & 0 & 0 & 0 & 0 \\ .35 & 1 & .35 & .35 & 0 & 0 & 0 & 0 \\ .35 & .35 & 1 & .35 & 0 & 0 & 0 & 0 \\ .35 & .35 & .35 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & .35 & .35 & .35 \\ 0 & 0 & 0 & 0 & .35 & 1 & .35 & .35 \\ 0 & 0 & 0 & 0 & .35 & .35 & 1 & .35 \\ 0 & 0 & 0 & 0 & .35 & .35 & .35 & 1 \end{bmatrix}$$

1. Pure "subject-splitting"(all observations from a subject stay together). When the observations go L,L,L,L,R,R,R,R, this leads to a patchup factor of k = 2.05. The naïve variance is

$$Var(\hat{\pi}_L) + Var(\hat{\pi}_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) = 0.7(4+4)(1/4+1/4) = 2.8,$$

which means that the corrected variance is $2.05 * (2.8) = 5.74$

2. A"mixed-split" where exactly half the observations from a subject go left and half go right. When the observations go L,L,R,R,L,L,R,R, this leads to a patchup factor of k = 0.65. The naïve variance is

$$Var(\hat{\pi}_L) + Var(\hat{\pi}_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) = 0.7(4+4)(1/4+1/4) = 2.8,$$

which means that the corrected variance is $0.65 * (2.8) = 1.82$

3. A "mixed-split" where *some* observations from a subject go left and others go right. When the observations go L, L, L, R, L, L, R, R, this leads to a patchup factor of k = 0.743. The naïve variance is

$$Var(\hat{\pi}_L) + Var(\hat{\pi}_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) = 0.7(5+3)(1/5+1/3) = 2.98\bar{6},$$

which means that the corrected variance is $0.743 * (2.987) = 2.219$

3.2.1.3 Unequal-variances patchup factor

If we neither assume equal variances for the two daughter nodes nor specify that we are under H_0, then we must account for the changes in the patchup factor due to having *un*equal variances for the two daughter nodes.

Still using a GEE-related approach, we now take the ratio of the weighted least-squares (WLS) variance *with* correlated observations to the variance *without* correlation.
The corrected variance is again computed by multiplying the naïve variance of $Var(\hat{\pi}_L - \hat{\pi}_R)$ by k, where

$$k = \frac{C(X'WX)^{-1}(X'W\Sigma WX)(X'WX)^{-1}C'}{C(X'WX)^{-1}C'} \tag{3.6}$$

—- Σ is a block diagonal Variance-Covariance matrix (each block represents a subject, each row represents a observation)

—- W is the diagonal matrix of weights such that $w_{ij} = \frac{1}{Var(obs_{ij})}$

With unequal variances, the variance of an observation would depend on whether that observation was sent to the left or right daughter node, and would be estimated as $\hat{\pi}_L(1-\hat{\pi}_L)$ or $\hat{\pi}_R(1-\hat{\pi}_R)$, respectively. After estimating the within-subject correlation, we are then able to construct the covariance matrix from the observation variances and the correlation matrix.

We again illustrate the patchup factor for three examples, this time allowing for UNEQUAL variances. Each example shows calculations of $Var(\hat{\pi}_L - \hat{\pi}_R)$, built from $\hat{\pi}_L = 0.8, \hat{\pi}_R = 0.4$, and $\hat{p} = 0.35$. While we still display 2 subjects with 4 observations per subject, in this situation the covariance matrices will *not* be identical, and as such are displayed separately.

1. Pure "subject-splitting"(all observations from a subject stay together). When the observations go L,L,L,L,R,R,R,R (*notation: R2 implies that an observation from subject 2 went to the Right daughter node*)

$$\Sigma = \begin{bmatrix} L1 & L1 & L1 & L1 & R2 & R2 & R2 & R2 \\ .16 & .056 & .056 & .056 & 0 & 0 & 0 & 0 \\ .056 & .16 & .056 & .056 & 0 & 0 & 0 & 0 \\ .056 & .056 & .16 & .056 & 0 & 0 & 0 & 0 \\ .056 & .056 & .056 & .16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .24 & .084 & .084 & .084 \\ 0 & 0 & 0 & 0 & .084 & .24 & .084 & .084 \\ 0 & 0 & 0 & 0 & .084 & .084 & .24 & .084 \\ 0 & 0 & 0 & 0 & .084 & .084 & .084 & .24 \end{bmatrix}$$

This leads to a patchup factor of k = 2.05 The naïve variance is

$$Var(\hat{\pi}_L) + Var(\hat{\pi}_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) = 0.7(4+4)(1/4+1/4) = 2.8,$$

which means that the corrected variance is $2.05 * (2.8) = 5.74$

2. A "mixed-split" where half the observations from a subject go left and half go right. When the observations go L, L, R, R, L, L, R, R

$$\Sigma = \begin{bmatrix} L1 & L1 & R1 & R1 & L2 & L2 & R2 & R2 \\ .16 & .056 & .0686 & .0686 & 0 & 0 & 0 & 0 \\ .056 & .16 & .0686 & .0686 & 0 & 0 & 0 & 0 \\ .0686 & .0686 & .24 & .084 & 0 & 0 & 0 & 0 \\ .0686 & .0686 & .084 & .24 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .16 & .056 & .0686 & .0686 \\ 0 & 0 & 0 & 0 & .056 & .16 & .0686 & .0686 \\ 0 & 0 & 0 & 0 & .0686 & .0686 & .24 & .084 \\ 0 & 0 & 0 & 0 & .0686 & .0686 & .084 & .24 \end{bmatrix}$$

This leads to a patchup factor of k = 0.664. The naïve variance is

$$Var(\hat{\pi}_L) + Var(\hat{\pi}_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) = 0.7(4+4)(1/4 + 1/4) = 2.8,$$

which means that the corrected variance is $0.664 * (2.8) = 1.86$

3. A "mixed-split" where *some* observations from a subject go left and others go right. When the observations go L, L, L, R, L, L, R, R

$$\Sigma = \begin{bmatrix} L1 & L1 & L1 & R1 & L2 & L2 & R2 & R2 \\ .16 & .056 & .056 & .0686 & 0 & 0 & 0 & 0 \\ .056 & .16 & .056 & .0686 & 0 & 0 & 0 & 0 \\ .056 & .056 & .16 & .0686 & 0 & 0 & 0 & 0 \\ .0686 & .0686 & .0686 & 0.24 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .16 & .056 & .0686 & .0686 \\ 0 & 0 & 0 & 0 & .056 & .16 & .0686 & .0686 \\ 0 & 0 & 0 & 0 & .0686 & .0686 & .24 & .084 \\ 0 & 0 & 0 & 0 & .0686 & .0686 & .084 & .24 \end{bmatrix}$$

This leads to a patchup factor of k = 0.755 The naïve variance is

$$Var(\hat{\pi}_L) + Var(\hat{\pi}_R) \approx V_0(n_L + n_R)(\frac{1}{n_L} + \frac{1}{n_R}) = 0.7(5+3)(1/5 + 1/3) = 2.98\bar{6},$$

which means that the corrected variance is $0.755 * (2.987) = 2.255$

3.2.1.4 Rats Implementation

Calculating within-subject correlation:

At a node: All observations within a certain parent node are potentially involved in the correlation calculation. To form the (X,Y) pairs for a specific subject: If there exists more than one observation from the same subject in the parent node, then an (X,Y) pair is created. Also, because there is not any specific "X" or "Y" designation, any pair (X_1, Y_1) also creates another pair (Y_1, X_1). For example: There are 3 observations (call them A, B, and C) from subject 8 in the parent node. Thus we create 6 data points to be used in the correlation calculation: (A,B), (B,A), (A,C), (C,A), (B,C), and (C,B).

The values themselves: For each observation we calculated an "h" value, which is found by taking the difference of each observation's response value and the average response value of the daughter node in which it is located. For example, an observation with a response value of 1 is in the left daughter node, which has an average response of 0.56. Then $h = 1 - .56 = 0.44$

To calculate the correlation: Pairs are formed from every possible subject in the parent node (subjects with less than two observations do not get included), and then the correlation is calculated by using the pairs from all of these subjects together.

A negative correlation estimate was occasionally obtained, which we classified as "no correlation", since our conceptual idea is to only examine within-subject correlation. Negative correlation may mean that there is correlation from some other source involved.

Obtaining the classification tree

For each approach (independent and dependent), we performed a CART analysis with the rpart() function in program R (Therneau and Atkinson 2010). In both approaches, our dependent variables were tooth and diet (factors with 28 and 8 levels respectively). We utilized the "user splits" option, which allows the creation and use of non-standard splitting functions and criteria. The proposed split with the largest (magnitude) test statistic was chosen to be used in the tree.

3.2.2 Methods for Birds Example–dependence PLUS detection/occupancy

3.2.2.1 Detection and Occupancy

In presence/absence studies, a common problem is the [potential] issue of false-negative data (recording the site as "unoccupied" when in fact it is occupied). This issue, also referred to as imperfect detection, can result in biased estimates of occupancy probability.

Within the modeling framework, some examples of how detection could be estimated from those multiple observations are by using maximum likelihood, a variety of adjusted binomial models, logistic regression, hierarchical modeling, etc. There may even be many different detection parameters depending on how the model is specified (for example, a multiple-species model, as in Bailey et al 2009).

For this example with CART, we incorporate imperfect detection into the split from a parent node to two daughter nodes using a method based on assigning a multinomial likelihood to the patches, with categories for 'm' detections out of 'k' survey occasions. At each node, π_{occ} and π_d represent the occupancy and detection probabilities. Where clarification is needed, we will use π_P, π_L, and π_R to denote occupancy probabilities for the parent, left daughter, and right daughter nodes respectively, rather than a general π_{occ}. This method is explained in more detail in Chapter 3. For this example, the likelihood for a node in the situation with 2 survey occasions per patch is

$$L(\pi_d, \pi_{occ}|n_0, n_1, n_2) \propto [(1-\pi_{occ}) + \pi_{occ}(1-\pi_d)^2]^{n_0} * [2\pi_{occ}\pi_d(1-\pi_d)]^{n_1} * [\pi_{occ}{\pi_d}^2]^{n_2} \quad (3.7)$$

3.2.2.2 Dependence

Conceptually, under the null hypothesis, the linear model (3.4) now applies to (π_{ij}), the probability of occupancy for patch j on site i, except that it is computed on the logistic scale in our optimization methods to ensure that estimates will fall in the parameter space. For this analysis, we again assume equal (i.e. exchangeable) correlation between all patches on the same site.

Equal-variances patchup factor This is the same as the equal-variance patchup factor in the Rats example.

Unequal-variances patchup factor This is the same patchup factor as in the unequal variances section of the Rats example, BUT

We need something quick for estimating the variance of the occupancy probability of each patch when constructing the covariance matrix. With unequal variances, the variance of a patch would depend on whether that patch was sent to the left or right daughter node. Following on the heels of the Rats example, we adopt the Bernoulli variance as an approximation, using the estimated node occupancies as parameters. Therefore, the patch variances are estimated as $\hat{\pi}_L(1-\hat{\pi}_L)$ or $\hat{\pi}_R(1-\hat{\pi}_R)$. Note that in this example we use the node estimates $\hat{\pi}_L$ and $\hat{\pi}_R$ found from maximum likelihood optimization rather than a sample mean. After estimating the within-site correlation, we are then able to construct the covariance matrix from the patch variances and the correlation matrix.

3.2.2.3 Birds Implementation

The original CART model was based on the use of five covariates: elevation (in meters), slope (percent slope), aspect(degrees,0-360), stand edge (indicator for whether the plot center is within 50 m from the nearest seral stage edge), and patch edge (indicator for whether the plot center is within 50 m from the nearest patch edge). We estimated occupancy and detection parameters using maximum likelihood on (3.7) with the optim() function in R. In the case of the two daughter nodes, the search for $\hat{\pi}_{occ}$ was restricted based on the parent node's $\hat{\pi}_d$. We modeled the parameters using a logistic transformation. We estimated the naïve variance $Var(\pi_L - \pi_R)$ as previously shown in equation (3.3), except that V_0 (i.e. Var($\hat{\pi}_{occ}$) for the parent node) was found via maximum likelihood estimation, using the negative inverse Hessian matrix followed by a Delta method transformation. We reduced computation time by limiting how often we computed variance estimates (and Hessian matrices) of any $\hat{\pi}_{occ}$ values to once per parent node, rather than twice for every potential split!

Correlation estimation: The correlation estimate is similar to that described in the Rats example implementation, except that we have to deal with imperfect detection. We are specifying the correlation to be through the occupancy of the sites, so we need to use a measure of occupancy. We cannot use the estimated $\hat{\pi}_{occ}$ for the parent node, or all patches will have the

same value. Similarly, we cannot use the estimated $\hat{\pi}_L$ and $\hat{\pi}_R$ (only two different values would result in a correlation of 1). Thus, we used a conditional estimate of occupancy, based on the data observed at each patch. Calculating P(occ | not seen) results in two different values (one for patches in the left daughter node, and one for patches in the right daughter node), and P(occ | seen) is taken to be equal to 1 (assuming a "closed" patch, i.e. not movement through, into, or out of each patch). Each of these patch values is then centered on the average daughter node value to form an "h" value, as described in the Rats example implementation section, and the "h" values are then used in the correlation calculation.

3.3 Results

3.3.1 Results for Rats Example

The Clustered Approach vs. the Independence Approach

If we account for the dependence in the data, there is a potential to cause changes in the tree structure, and therefore to also cause changes in estimates or predictions for individuals in the tree. Figure 3.1 shows the usual CART tree produced assuming independence and the CART tree that accounts for the clustering of observations within subject. We display only a portion of each tree below (Figure 3.1, Table 3.1, and Table 3.2). We see that the clustered approach has *not* changed the first split in the tree, but it *does* change the second. For the clustered approach, the second split is based on "teeth" 2 and 3 (i.e. the 2nd and 3rd occlusal surfaces in each rat), while the independent approach splits on Diet 8. Furthermore, *all* of the splits in the tree accounting for dependence of observations within rat (even those not shown) are based on the "tooth" factor, while the tree assuming independence contains splits based on both "tooth" and diet.

We do not display the trees resulting from the unequal variances method, as they are identical to the trees displayed for equal variances. In the independence case, this is by design; when there is no correlation, the patchup factor is equal to 1. Since there are no other differences between the two methods, the results are identical. In the clustered examples, the equality between the equal and unequal variance methods is situational only, and does not hold in other

examples. We previously showed some small examples which demonstrate that the patchup factors *can* be different depending on the assumption of equal or unequal variances, which in turn allows for the possibility of different splitting choices.

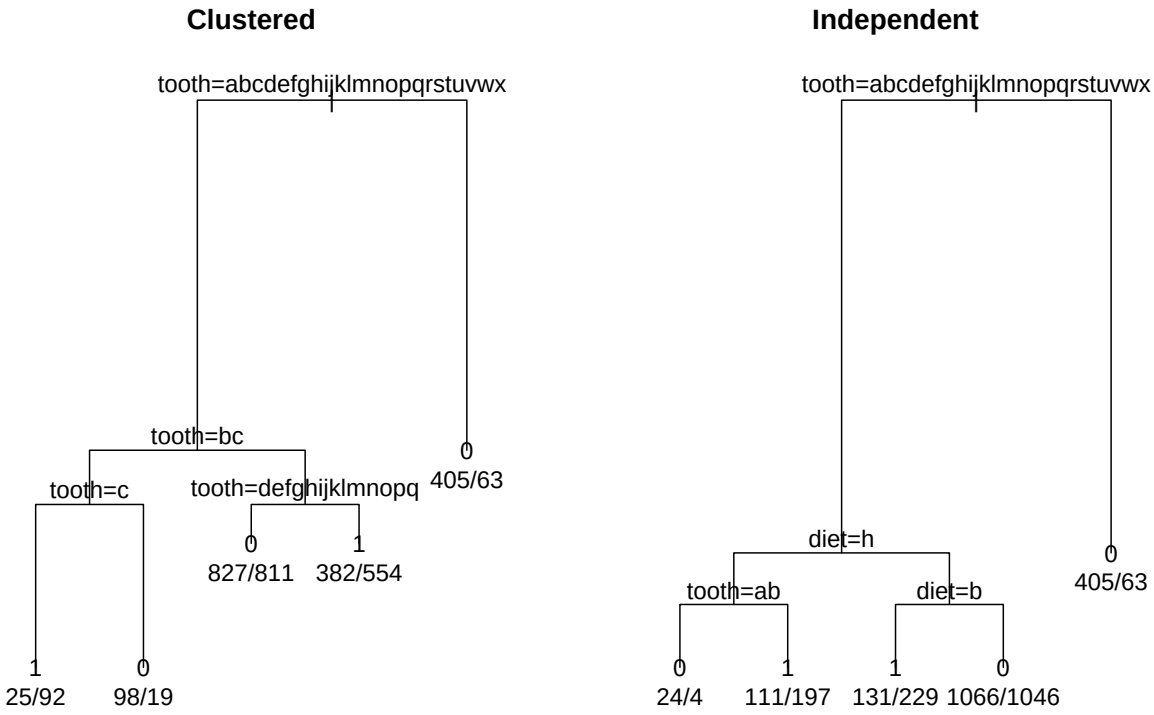

Figure 3.1 The tree structures produced by rpart() for the clustered and independent approaches of the Rats example, using cp=0.01. Some of the leaves in the diagram have been trimmed for ease of viewing. The 0 or 1 label on each terminal node is a damage prediction based on a $\hat{\pi} = 0.5$ cutoff for estimated damage probability. The X/Y ratio describes observations where the species was "(0 or 1) not damaged / (2 or 3) damaged". The differences in tree structure begin at Node 2 (1st left-daughter node).

Table 3.1 The node estimates of damage resulting from the equal variances method allowing for clustered data, using 2 covariates (tooth and diet) with cp=0.01. A (*) represents a terminal node of the tree. Nodes 10 and 11 have been trimmed for ease of viewing, and now appear as terminal nodes.

tree node	Split	damaged/n	0 $\hat{\text{damage}}$	1 $\hat{\text{damage}}$
1) root	1539/3276	.5302	.4698	
2)	tooth.f=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24	1476/2808	.4744	.5256
4)	tooth.f=2,3	111/234	.5256	.4744
8)*	tooth.f=3	92/117	.2137	.7863
9)*	tooth.f=2	19/117	.8376	.1624
5)	tooth.f=1,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24	1365/2574	.4697	.5303
10)*	tooth.f=4,5,6,7,8,9,10,11,12,13,14,15,16,17	811/1638	.5049	.4951
11)*	tooth.f=1,18,19,20,21,22,23,24	554/936	.4081	.5919
3)*	tooth.f=25,26,27,28	63/468	.8654	.1346

Table 3.2 The node estimates damage resulting from the EQUAL variances method under independence, using 2 covariates (tooth and diet) with cp=0.01. A (*) represents a terminal node of the tree. Nodes 9, 10, and 11 have been trimmed for ease of viewing, and now appear as terminal nodes.

tree node	Split	seen/n	0 $\hat{\text{damage}}$	1 $\hat{\text{damage}}$
1)	root	1539/3276	.5302	.4698
2)	tooth.f=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24	1476/2808	.4744	.5256
4)	diet.f=8	101/336	.4018	.5982
8)*	tooth.f=1,2	4/28	.8571	.1429
9)*	tooth.f=3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24	197/308	.3604	.6396
5)	diet.f=1,2,3,4,5,6,7	1275/2472	.4842	.5158
10)*	diet.f=2	229/360	.3639	.6361
11)*	diet.f=1,3,4,5,6,7	1046/2112	.5047	.4953
3)*	tooth.f=25,26,27,28	63/468	.8654	.1346

3.3.2 Results for Birds Example

The Clustered Approach vs. the Independence Approach Under the assumption of equal variances, the trees and estimates for both approaches were identical to the independence approach with unequal variances (see Figure 3.2 and Table 3.4). As previously explained above Figure 3.1, we expect the two independence situations to be identical. The clustered approach for equal variances *also* results in the same tree and estimates, although this outcome is situational and is similar in nature to the outcome of the Rats example.

Unequal variances

Once again, when we account for the dependence in the data, the result is a tree with different splitting choices (Figure 3.2, Table 3.3). This can be seen starting with the first left daughter node: the clustered approach splits the group using an aspect value of 332.5, while the independent tree uses an aspect value of 17.5. At that point, the classification tree differs from the tree assuming independence. Note that while final classification (yes/no for occupancy in this example) could still be the same, estimates are likely distinct between nodes and individual patches therein, and tree size is clearly different. When allowing for a clustering effect between patches on the same site, the tree (based on the same data!) was much larger (more splits and more nodes, which have been trimmed from the diagram and estimates for ease of viewing).

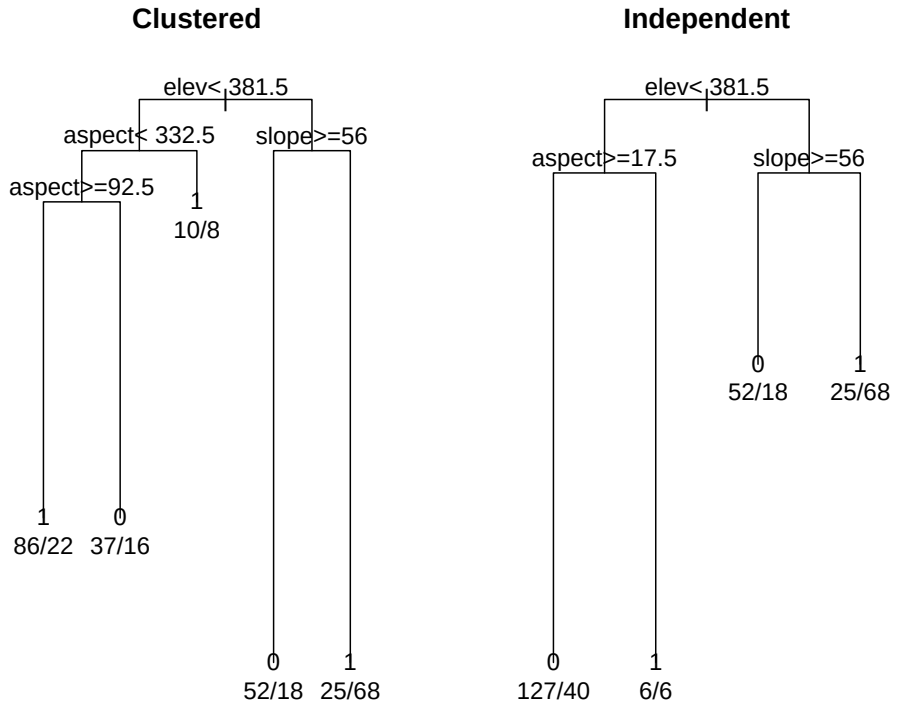

Figure 3.2 The tree structures produced by rpart() for the clustered and independent approaches of the Birds example, assuming unequal variances, using cp=0.01. The left-most leaf in the Clustered diagram has been trimmed for ease of viewing. The 0 or 1 label at each terminal node is an occupancy prediction based on a 0.5 cutoff for estimated occupancy probability. The X/Y ratio gives the number of patches where the species was "not seen/seen". Changes in tree structure begin at Node 2 (1st left-daughter node).

Table 3.3 The node estimates of occupancy and detection resulting from the unequal variances, clustered method, using 5 covariates (slope, elev, aspect, s.edge, p.edge) with cp=0.01. A (*) represents a terminal node of the tree. Node (8) was trimmed for ease of viewing, and now appears as a terminal node. The estimates shown here are computed as if each node were a parent node.

tree node	Split	seen/n	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$
1)	root	132/342	.5731	.4286
2)	elev < 381.5	46/179	.5091	.2963
4)	aspect < 332.5	38/161	.5742	.2326
8)*	aspect >= 92.5	22/108	.9962	.1070
9)*	aspect < 92.5	16/53	.4717	.3999
5)*	aspect >= 332.5	8/18	.5602	.5454
3)	elev >= 381.5	86/163	.7118	.4910
6)*	slope >= 56	18/70	.3429	.5000
7)*	slope < 56	68/93	.9895	.4890

Table 3.4 The node estimates of occupancy and detection resulting from the unequal variances, independent method, using 5 covariates (slope, elev, aspect, s.edge, p.edge) with cp=0.01. A (*) represents a terminal node of the tree. The estimates shown here are computed as if each node were a parent node.

tree node	Split	seen/n	$\hat{\pi}_{occ}$	$\hat{\pi}_{det}$
1)	root	132/342	0.5731	0.4286
2)	elev < 381.5	46/179	0.5091	0.2963
4)*	aspect >= 17.5	40/167	0.4724	0.2979
5)*	aspect < 17.5	6/12	0.9934	0.2936
3)	elev >= 381.5	86/163	0.7118	0.4910
6)*	slope >= 56	18/70	0.3429	0.5000
7)*	slope < 56	68/93	0.9895	0.4890

3.3.3 Simulation Evaluation of the Patchup Factor

Using either patchup factor method (equal or unequal variances) seems to provide a boost to the accuracy of Type I error rates. We ran several simulations to explore the error rates at the 5% level. Each simulation included 1000 realizations of the test statistic, which were compared to the tails of a Normal distribution. The general steps of the simulation are as follows:

Correlation was induced on site 'i' through a simulated epsilon value as

$$\epsilon_i = rnorm(0, sigma)$$

Patches were sent to each daughter node, using either a Mixed allotment (approximately half of all patches on a site were sent to the Left daughter node, while the other half went to the Right daughter node) or a Pure allotment (for a given site, ALL patches on the site stayed together and were sent to the same daughter node). In the Pure case, we sent half of the sites to the Left daughter node and the other half to the Right daughter node. The probabilities of occupancy in each daughter node were set equal to each other, referred to below as "side.effect". That node probability was then applied to all patches within that node. We then computed patch occupancy probabilities for every patch j on site i as

$$\pi_{ij} < -1/(1 + exp(-(side.effect + \epsilon_i)))$$

Next we simulated patch occupancy status (0 or 1) as

$$occ.true_{ij} < -rbinom(1, 1, \pi_{ij})$$

followed by simulating observations (either 1 or 2 occasions) for each patch as

$$obs.data_{ij} < -rbinom(\#occasions, 1, occ.true_{ij} * \pi det)$$

where πdet is the assigned probability of detection (if #occasions is one, the $\pi_{det} = 1$).

Using estimation methods described in Examples 1 (Rats) or 2 (Birds), we estimated π_{occ} for each daughter node, as well as estimating the naïve variance of $Var(\pi_L - \pi_R)$. We then applied three different patchup approaches (independence (Indpt), equal variances (EV), and

unequal variances (UV)) to come up with test statistics and the corresponding Type 1 error rates for each simulation.

These error rates are shown in Table 3.5 below. We used $\pi_{occ} = 0.5$ for all scenarios unless otherwise noted. In addition to type of splitting (Mixed vs. Pure) and π_{occ}, we varied π_{det}, the number of sites, and the number of patches per site (pps).

While there is occasionally an "over-correction" (e.g. see the second scenario in the table), the clustered approaches generally produce a more accurate Type I error rate than the independence approach. The exception to this comes when using smaller sample sizes (in both number of sites *and* number of patches per site) where it is more difficult to get an accurate estimate of the within-site correlation. We also note the often-identical EV and UV error rates. This is not unusual for two reasons: 1) Even during other simulations, there was shown to be no difference in patchup factor between EV and UV in pure site-splitting situations, and more importantly 2) When we are under the null hypothesis and $\pi_L = \pi_R$ (the simulations are a close approximation to this), then the UV method becomes the EV method (or a very close approximation). We also note that at smaller sample sizes, we see the first deviations betwe

Table 3.5 Type I error rates for the Independent (Indpt), equal variances (EV), and unequal variances (UV) methods, under various simulated scenarios. Each scenario uses 1000 trials, 100 sites and $\pi_{occ} = 0.5$ unless otherwise noted. The notation 'pps' stands for 'patches per site'.

Description	**Indpt**	**EV**	**UV**
Mixed, $\pi_{det} = 0.4$, 3 pps	.025	.043	.043
Mixed, $\pi_{det} = 0.8$, 3 pps	.018	.055	.055
Mixed, $\pi_{det} = 0.4$, 15 pps	.027	.044	.044
Mixed, $\pi_{det} = 0.8$, 15 pps	.007	.036	.036
Mixed, $\pi_{det} = 0.8$, 3 pps, 4 sites	.038	.064	.100
Mixed, $\pi_{det} = 0.8$, 3 pps, 10 sites	.024	.052	.048
Mixed, $\pi_{det} = 0.8$, 30 pps, 4 sites	.025	.075	.059
Mixed, $\pi_{det} = 0.8$, 30 pps, 10 sites	.009	.056	.055
Mixed, $\pi_{det} = 1$, $\pi_{occ} = 0.3$, 3 pps	.022	.044	.044
Mixed, $\pi_{det} = 1$, 3 pps	.019	.059	.059
Mixed, $\pi_{det} = 1$, $\pi_{occ} = 0.9$, 3 pps	.014	.052	.052
Pure, $\pi_{det} = 0.4$, 3 pps	.103	.038	.038
Pure, $\pi_{det} = 0.8$, 3 pps	.173	.049	.049
Pure, $\pi_{det} = 0.4$, 15 pps	.328	.038	.038
Pure, $\pi_{det} = 0.8$, 15 pps	.471	.045	.045
Pure, $\pi_{det} = 0.8$, 3 pps, 4 sites	.194	.141	.203
Pure, $\pi_{det} = 0.8$, 3 pps, 10 sites	.186	.102	.104
Pure, $\pi_{det} = 0.8$, 30 pps, 4 sites	.675	.259	.266
Pure, $\pi_{det} = 0.8$, 30 pps, 10 sites	.642	.091	.091
Pure, $\pi_{det} = 1$, $\pi_{occ} = 0.3$, 3 pps	.180	.052	.052
Pure, $\pi_{det} = 1$, 3 pps	.175	.046	.046
Pure, $\pi_{det} = 1$, $\pi_{occ} = 0.9$, 3 pps	.176	.053	.053

3.4 Discussion

This is a general method that applies to CART for binary data. For those attempting to change or extend the current examples, there are a several options. Of course the results may differ if the covariates are different. Related to the Birds example, we could easily extend this work to more than two occasions. The likelihood in equation 3.7 would simply be extended to account for each data scenario. We could extend this method to more than two levels. Currently we are only accounting for within-subject dependence, but there could be another level of dependence (e.g. site-within-region). The complexity factor (cp) is currently set at .01 and could be adjusted (cp plays a role in the formation and pruning of the tree). The working correlation matrix could be adjusted. We decided to use an exchangeable correlation structure (the same correlation between any two patches on the same site, but no correlation between patches on different sites). There are several other options, but the choice of structure does not prevent consistent estimators–it only changes efficiency. Due to a small number of patches-per-site in test data sets, we did not attempt to estimate separate correlations for each site. In our work, the exchangeable correlation is estimated as a pooled (common) correlation using all pairs of patches in site 1, all pairs of patches in site 2, etc. (based on whichever patches are available in the current parent node). However, this correlation only gets applied to those pairs of patches in the same site. For example, $Cor(X_{site1i}, X_{site1j}) = Cor(X_{site2i}, X_{site2j}) = \hat{p}$, but $Cor(X_{site1i}, X_{site2j}) = 0$. This method, like the imperfect detection method described in Chapter 2, could be easily extended to random forests (Breiman 2001).

The extensions and modifications to CART are already numerous. It is the hope of the authors that our proposed methods will lend CART to areas of study that have perhaps not thought about using such a method in their analysis.

3.5 Literature Cited

Andrews, D.F., and Herzberg, A.M. 1985. Data: A Collection of Problems from Many Fields for the Student and Research Worker. Springer-Verlag, New York.

Bahjat, Q. and Liang, K.Y. 1992. Marginal Models for Correlated Binary Responses with Multiple Classes and Multiple Levels of Nesting. *Biometrics*. Vol. 48, No. 3, pp. 939-950.

Bailey, Larissa L., Janice A. Reid, Eric D. Forsman, James D. Nichols. 2009. Modeling co-occurrence of northern spotted and barred owls: Accounting for detection probability differences. *Biological Conservation*. Vol. 142, Issue 12, p. 2983-2989.

Bel, L., D. Allard, J.M. Laurent, R. Cheddadi, A. Bar-Hen. 2009. CART algorithm for spatial data: Application to environmental and ecological data. *Computational Statistics and Data Analysis*. Vol. 53, p. 3082-3093

Bourg, Norman A., William J. McShea, Douglas E. Gill. 2005. Putting a CART Before the Search: Successful Habitat Prediction for a Rare Forest Herb. *Ecology*. Vol. 86, No. 10, p. 2793-2804.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. 1984. Classification and Regression Trees. Wadsworth International Group, Belmont, CA.

Castellón, Traci D., and Sieving, Kathryn E. 2006. Landscape History, Fragmentation, and Patch Occupancy: Models for a Forest Bird with Limited Dispersal. *Ecological Applications*. Vol. 16, No. 6, pp. 2223-2234.

Clark, L.A. and D. Pregibon. 1992. Tree-based Models. Chapter 9. *In*: J.M. Chambers and T.J. Hastie (eds.). Statistical Models in S. Wadsworth and Brooks, Pacific Grove, CA.

De'ath, Glenn, and Fabricius, Katharina E. 2000. Classification and Regression Trees: A Powerful Yet Simple Technique for Ecological Data Analysis. *Ecology*. Vol. 81, No. 11, pp. 3178-3192.

Dormann, C.F., McPherson, J.M., Arau jo, M.B., Bivand, R., Bolliger, J., Carl, G., Davies, R.G., Hirzel, A., Jetz, W., Kissling, W.D., Kuhn, I., Ohlemuller, R., Peres-Neto, P.R., Reineking, B., Schroder, B., Schurr, F.M. and Wilson, R. 2007. Methods to account for spatial autocorrelation in the analysis of species distributional data: a review. *Ecography* Vol. 30, p. 609628.

Le Cessie, S. and Van Houwelingen, J.C. 1994. Logistic Regression for Correlated Binary Data. *Applied Statistics*. Vol. 43, No. 1, pp. 95-108.

Li, Xiang, and Claramunt, Christophe. 2006. A Spatial Entropy-Based Decision Tree for Classification of Geographical Information. *Transactions in GIS* Vol. 10, No. 3, p. 451-467.

Liang, K.Y., Zeger, S.L. and Qaqish, B. 1992. Multivariate regression analyses for categorical data (with discussion). *Journal of the Royal Statistical Society B*. Vol. 54, pp. 3-40.

McGarigal, Kevin and McComb, William C. 1995. Relationships Between Landscape Structure and Breeding Birds in the Oregon Coast Range. *Ecological Monographs*. Vol. 65, No. 3, pp. 235-260.

Molinaro, Annette M., Sandrine Dudoit, Mark J. van der Laan. 2004. Tree-based multivariate regression and density estimation with right-censored data. *Journal of Multivariate Analysis*. Vol. 90, p. 154177.

Murray, J.V., S. Low Choy, C.A. McAlpine, H.P. Possingham, A.W. Goldizen. 2008. The importance of ecological scale for wildlife conservation in naturally fragmented environments: A case study of the brush-tailed rock-wallaby (Petrogale penicillata). *Biological Conservation*. Vol 141, p. 7-22.

Neuhaus, J. M., Kalbfleisch, J. D. and Hauck, W. W. 1991. A comparison of cluster-specific and population-averaged approaches for analyzing correlated binary data. *International Statistical Review*. Vol. 59,, No. 1, pp. 25-35.

Prentice, R.L. 1988. Correlated binary regression with covariates specific to each binary observation. *Biometrics*. Vol. 44 No. 4, 1033-1048.

Segal, Mark R. 1992. Tree-Structured Methods for Longitudinal Data. *Journal of the American Statistical Association*. Vol. 87, No. 418 (June), p. 407-418.

Sela, Rebecca J. and Simonoff, Jeffrey S. 2012. RE-EM trees: A data mining approach for longitudinal and clustered data. *Machine Learning*. Vol. 86, p. 169-207.

Therneau, Terry and Atkinson, Elizabeth. R port by Brian Ripley. 2010.
rpart: Recursive Partitioning. R package version 3.1.-46.
http://CRAN.R-project.org/package=rpart

3.6 Extra material: R code for Unequal Variances method

3.6.1 User splits

```
# Attempting to use the "user splits" option in R
#  Requires 3 pieces:  Init, Eval, and Splits
#  Optional 4th part called 'parms' to pass in other information
#  *NOTE: parms MUST be part of the call to rpart().
#         It will not work from the global environment.

options(warn = 1)   #prints warnings as they occur,
                    #rather than waiting until the end
options(digits=7)   # controls number of digits/decimals (default is 7)
        #  if digits is set too low, numbers may go to scientific notation
library(rpart)
set.seed(7)

###################################################################
# The 'evaluation' function.  Called once per node.
#  Produce a label (1 or more elements long) for labeling each node,
#  and a deviance.  The latter is
#       - of length 1
#       - equal to 0 if the node is "pure" in some sense (unsplittable)
#       - does not need to be a deviance: any measure that gets larger
#            as the node is less acceptable is fine.
#       - the measure underlies cost-complexity pruning, however

###############
###############  Mark's eval() code
temp1 <- function(y, wt, parms) {
# print("***** START: Evaluating *****")
         Ns <- timesseen(y, parms);
     # *NOTE: Ns[1] = n0 = never seen, Ns[2] = n1 = seen once,
     #        Ns[3] = n2 = seen twice

  # If using orig.parent, I'd like to report the same prob.det being used
  # in the split function, as well as the corresponding prob.occ
   if(parms$mygoodness==1){ param.parent <- optim(c(1), lnl.t.fixed,
       gr=NULL, method="BFGS", control=list(fnscale=-1),
       prob.det = parms$prob.det, Ns = Ns )$par
         param.parent <- backt(param.parent)
         prob.occ <- param.parent[1]
         prob.det <- parms$prob.det
                                  }
    # Anything else, I will report the node-specific prob.det and prob.occ
    # Note that for each.parent and p.v.2d methods, this does not reflect
```

```
    #  the values being used in the split() function
         else{
         param.parent <- optim(c(-1, -1), lnl.t, gr=NULL, method="BFGS",
                              control=list(fnscale=-1), Ns=Ns )$par
         param.parent <- backt(param.parent)
         prob.occ <- param.parent[1]
         prob.det <- param.parent[2]
                }

    labels <- matrix(nrow=1, ncol=5)
    # labels[1] is the fitted y category (i.e. the prediction of occ status)
    # labels[2] is sum(y == 0)   i.e. the "unseen" locations
    # labels[3] is sum(y >= 1)   i.e. the "seen" locations
    # labels[4] is prob.occ
    # labels[5] is prob.det
labels[1] <- ifelse(prob.occ >= parms$cutoff, 1, 0)
      labels[2] <- Ns[1]
      labels[3] <- sum(Ns[-1])
      labels[4] <- prob.occ
      labels[5] <- prob.det
# print(labels)
    dev <- ifelse(prob.occ >= parms$cutoff, Ns[1], sum(Ns[-1]))

    ret <- list(label=labels, deviance=dev)
# print("***** END: Evaluating *****")
    ret
    }

################ end Mark's eval() code
################

# The split function, where most of the work occurs.
#   Called once per split variable per node.
# If continuous=T
#   The actual x variable is ordered
#   y is supplied in the sort order of x, with no missings,
#   return two vectors of length (n-1):
#      goodness = goodness of the split, larger numbers are better.
#                 0 = couldn't find any worthwhile split
#        the ith value of goodness evaluates splitting obs 1:i vs (i+1):n
#      direction= -1 = send "y< cutpoint" to the left side of the tree
#                  1 = send "y< cutpoint" to the right
#         this is not a big deal, but making larger "mean y's" move towards
#         the right of the tree, as we do here, seems to make it easier to
#         read
# If continuos=F, x is a set of values defining the groups for an
#   unordered predictor.  In this case:
```

```
#       direction = a vector of length m= "# groups".
#       direction actually displays the names/labels for each group.
#        It asserts that the best split can be found by
#          lining the groups up in this order and going from left to right,
#           so that only m-1 splits need to be evaluated rather than 2^(m-1)
#       goodness = m-1 values here.
#
# The reason for returning a vector of goodness is that the C routine
#   enforces the "minbucket" constraint. It selects the best return value
#   that is not too close to an edge.

###############
###############  Mark's split() code

temp2 <- function(y, wt, x, parms, continuous) {

#####################################################################
#####################################################################
#####################################################################
#print("***** START: Splitting *****")
  if(parms$mygoodness==1){mygoodness=LRT.orig.parent}
  if(parms$mygoodness==2){mygoodness=LRT.each.parent}
  if(parms$mygoodness==3){mygoodness=LRT.parent.v.2daughters}
  if(parms$mygoodness==4){mygoodness=LRT.parent.v.daughter.v.daughter}

   idx <- order(x);  x <- x[idx];   y <- y[idx,];
        #Just in case ordering is not already done elsewhere
   y <- cbind(y)      # If y is a vector, this allows me to only calculate n
    #     using one method (cbind instead of length)
   n <- nrow(y)
   parent <- y       # often refer to the node being split as the parent


################
################
#  Run optim on the parent node, get the hessian,
# pass the variances into "goodness".
#  I will be able to get Ns.Left and Ns.Right from inside the "goodness",
#       e.g. each.parent

       Ns.parent <- timesseen(parent, parms)

       outcome.parent <- optim(c(-1, -1), lnl.t, gr=NULL, method="BFGS",
        control=list(fnscale=-1), hessian=TRUE, Ns = Ns.parent )
         param.parent <- backt(outcome.parent$par)
         prob.occ.P <- param.parent[1]
         prob.det.P <- param.parent[2]
```

```
          hessian.occ <- outcome.parent$hessian[1,1]
          varcov.matrix <- solve(-hessian.occ)
               # Observed Info. matrix IS the negative Hessian     #
               # Using only the .occ piece so it is conditional on pi.det,
               #  just like the daughter nodes would be
               #  (except the parent is funnily conditional on its
               #   own estimate of pi.det)
               #   (do this through [1,1] just below
         #print("Using Delta Method to change var-cov OUT of logistic scale")
          pi.occ.log.P <- outcome.parent$par[1]
         actual.var.occ.P <- varcov.matrix[1,1]*((exp(-pi.occ.log.P)/
                   ( (1+exp(-pi.occ.log.P))^2 ) )^2 )

#################
#################

    if (continuous) {     # continuous x variable

    # Get the goodness
            ## MAKE SURE IT IS A VECTOR!!
            ## Because rpart does the minimum node size elsewhere,
            ## I just have to compute n-1 deviances here.
      possibles <- rep(0,n-1)
      direction <- rep(-1, n-1)
      prob.occ.L = prob.occ.R <- rep(0, n-1)

      for (i in 1:(n-1)) {
         left <- matrix(parent[1:i,], ncol=ncol(parent))
         right <- matrix(parent[(i+1):n,], ncol=ncol(parent))
         if(x[i]==x[i+1]) {next}
            ### NOT allowed to split up observations with the same x value

         info <- mygoodness(parent, left, right,
                               orig.prob.det=parms$prob.det,
           actual.var.occ.P, prob.occ.P, prob.det.P)
         possibles[i] <- info[1]     # the test statistic
         prob.occ.L[i] <- info[2]
         prob.occ.R[i] <- info[3]

    # Get the direction   ALSO A VECTOR!!
      if(prob.occ.L[i] > prob.occ.R[i]){ direction[i] <- 1}
     # Compares occupancy probabilities, sends the higher one to the right
                           } # end 'for' loop


    goodness <- possibles
```

```
    ret <- list(goodness=goodness, direction=direction)
#print("***** END: Splitting *****")
    ret
    }

   else {
# Categorical X variable
     #  we can order the categories by their means
     #  (i.e. estimated prob.occ values)
      #  then use the same code as for a non-categorical
ux <- sort(unique(x))
  # Sort does smallest to larget (either numerical or alphabetical)
      m <- length(ux)

      occs <- 0

      for(i in 1:m){
         group <- matrix(y[x==ux[i], ], ncol=ncol(y))
    # Needed in the extreme case that either 'right' or 'left' is only 1 row
    #  For some reason it loses its matrix designation,
    # and nrow() won't work otherwise
         Ns <- timesseen(group, parms);
         param <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
                                    control=list(fnscale=-1), Ns=Ns )$par
         param <- backt(param)
         prob.occ <- param[1]
         occs[i] <- prob.occ
                     }  # end 'i' loop

ord <- order(occs)   #tells where each number belongs in order
  #  e.g. 2 1 4 3  means: first number is the second-lowest,
  #                       2nd number is the smallest

    # Get the goodness
            ## MAKE SURE IT IS A VECTOR!!
            ## Because rpart does the minimum node size elsewhere,
            ## I just have to compute m-1 deviances here.
      possibles <- rep(0,m-1)
      prob.occ.L = prob.occ.R <- rep(0, m-1)

      for (i in 1:(m-1)) {
         left <- matrix(parent[ x<=ux[i], ], ncol=ncol(y))
    # Needed in the extreme case that either 'right' or 'left' is only 1 row
         right <- matrix(parent[ x>ux[i], ], ncol=ncol(y))
      #  For some reason it loses its matrix designation,
      #     and nrow() won't work otherwise
```

```
          info <- mygoodness(parent, left, right,
                               orig.prob.det=parms$prob.det,
            actual.var.occ.P, prob.occ.P, prob.det.P)
          possibles[i] <- info[1]
          prob.occ.L[i] <- info[2]
          prob.occ.R[i] <- info[3]    }

    # Get the direction   ALSO A VECTOR!!
       direction <- ux[ord]

    goodness <- possibles
    ret <- list(goodness=goodness, direction=direction)
#print("***** END: Splitting *****")
    ret
}
   }

###############  end Mark's split() code
###############

# The init function:
#   fix up y to deal with offsets
#   return a parms list--this can be passed in from the call to rpart(),
#      but it MUST be reproduced (or changed) in init()
#      parms includes cutoff (for predictions/labeling),
#                  occasions (# sampling times/observations per member),
#                  prob.det (if specified by the user), and
#                  goodness (which imperfect detection method should be used)
#   numresp is the number of values produced by the eval routine's "label"
#   numy is the number of columns for y
#   summary is a function used to print one line in summary.rpart
#      yval is the matrix "yval2" in tree$frame
#      each row contains predicted value, deviance, n, prob.occ, prob.det
#   text is a function used to put text on the plot in text.rpart
# *NOTE: The split information printed is NOT controlled by the text
#             function in init()
#         Only the terminal node information comes from this text function
# In general, this function would also check for bad data,
#      see rpart.poisson as example

###############
############### begin Mark's init() code

temp3 <- function(y, offset, parms, wt) {
#    print("***** START: Init *****")
    if (!is.null(offset)) y <- y-offset
    # IF method is orig.parent and prob.det is not specified:
```

```
    #   which.goodness <- (deparse(substitute(mygoodness)))
    # print(which.goodness); print(which.goodness=="LRT.orig.parent")
    if(parms$mygoodness==1 & is.null(parms$prob.det)==TRUE) {
    # Calculate orig.prob.det from the very first parent node
    #  (i.e. all of the data)
     Ns <- timesseen(y, parms)
      param.parent <- optim(c(0.5, 0.5), lnl.t, gr=NULL, method="BFGS",
                                control=list(fnscale=-1), Ns = Ns)$par
      param.parent <- backt(param.parent)
      #orig.prob.occ <- param.parent[1]
      orig.prob.det <- param.parent[2]
      parms$prob.det <- orig.prob.det
          }  # end 'if' statement

ret <- list(y=y, parms=parms, numresp=5, numy=parms$occasions+2,
              #the "+2" is so that I can pass in SITE and PATCH labels
    summary= function(yval, dev, wt, ylevel, digits ) {
      paste("predicted value=", yval[,1], "deviance=", dev, "prob.occ=",
           round(yval[,4],digits), "prob.det=",
           round(yval[,5],digits) )
                       }, #end summary

    text= function(yval, dev, wt, ylevel, digits, n, use.n ) {
          nclass <- (ncol(yval) - 1)/2
          group <- yval[, 1]
          counts <- yval[, 1 + (1:nclass)]
          if (!is.null(ylevel)) {group <- ylevel[group] }
          temp1 <- format(counts, digits)
          if (nclass > 1) { temp1 <- apply(matrix(temp1,
                          ncol = nclass), 1,paste, collapse = "/") }
          if (use.n) { out <- paste(format(group, justify = "left"),
           "\n", temp1, sep = "") }
          else {out <- format(group, justify = "left") }
          return(out)
                          },     #end text

    print= function(yval, ylevel, digits){
        if (is.null(ylevel)) {temp <- as.character(yval[, 1]) }
          else {temp <- ylevel[yval[, 1]] }
        nclass <- (ncol(yval) - 1)/2
        if (nclass < 5) {yprob <- format(yval[, 1 + nclass + 1:nclass],
         digits = digits, nsmall = digits)}
      else {yprob <- formatg(yval[, 1 + nclass + 1:nclass], digits = 2)}
if (is.null(dim(yprob))) {yprob <- matrix(yprob, ncol = length(yprob)) }
      temp <- paste(temp, " (", yprob[, 1], sep = "")
     for (i in 2:ncol(yprob)) temp <- paste(temp, yprob[, i], sep = " ")
        temp <- paste(temp, ")", sep = "")
```

```
            temp
                              }  #end print

          )  # end ret
#    print("***** END: Init *****")
    ret
    }
```

3.6.2 Companion functions

```
# Calculates the total number of times the species was detected at each site
timesseen <- function(y, parms){
   timesseen <- 0
   Ns <- rep(0, (parms$occasions+1)  )
      for (i in 1:nrow(y)){
           timesseen[i] <- sum(y[i,1:parms$occasions ]) }
      for (j in 1:length(Ns)){
           Ns[j] <- sum(timesseen==(j-1)) }
      # NOTE: Ns[1] = n0 = never seen, Ns[2] = n1 = seen once,
      #       Ns[3] = n2 = seen twice
              return(Ns)
         }    #end timesseen

# backtransforming parameters when using logistic representation
backt <- function(ln.param) {
           1/(1+exp(-ln.param))
              } #end backt
############
# lnl using logistic parameterization
#    Used for any node when estimating both prob.occ and prob.det
lnl.t <- function(param, Ns){
             ln.Likelihood(backt(param), Ns)
                   } #end lnl.t

ln.Likelihood <- function(x, Ns){
      prob.occ <- x[1]
      prob.det <- x[2]
   #print(c(prob.occ, prob.det))
      k <- length(Ns)-1               # k = number of sampling occasions
      ln.like <- Ns[1]*log((1-prob.occ) + prob.occ*(1-prob.det)^k)
                      # not occupied or occupied and seen 0 times
      for(i in 1:k){   # occupied, seen 'i' times out of 'k' possible
  ln.like <- ln.like + Ns[i+1]*log( choose(k,i) * prob.occ * (prob.det^i) *
           ((1-prob.det)^(k-i)) )
                     } # end 'i' loop
       return(ln.like) }  #end ln.Likelihood
```

```
############
#  Need for nodes when I'm fixing prob.det while optimizing pi.occ
#  this occurs in orig.parent and each.parent
lnl.t.fixed <- function(param, prob.det, Ns){
            ln.Likelihood.fixed(backt(param), prob.det, Ns)
                  } #end lnl.t.fixed

ln.Likelihood.fixed <- function(x, prob.det, Ns){
      prob.occ <- x[1]
      k <- length(Ns)-1                # k = number of sampling occasions
      ln.like <- Ns[1]*log((1-prob.occ) + prob.occ*(1-prob.det)^k)
        # not occupied or occupied and seen 0 times
      for(i in 1:k){   # occupied, seen 'i' times out of 'k' possible
  ln.like <- ln.like + Ns[i+1]*log( choose(k,i) * prob.occ * (prob.det^i) *
          ((1-prob.det)^(k-i)) )
                    } # end 'i' loop
       return(ln.like) }  #end ln.Likelihood


############
### New GEE function--estimating correlation within a site.

rho <- function(left, right){
   left <- as.data.frame(left); right <- as.data.frame(right);
      left$patch.occ <- left$patch.occ - mean(left$patch.occ)
      right$patch.occ <- right$patch.occ - mean(right$patch.occ)
        c.left <- left; c.right <- right
        data <- rbind(c.left, c.right);
        w <- ncol(data)-3    #the 3 is to exclude SITE, PATCH, and patch.occ
        data2 <- data[,1:w];
    # Allows me to calculate parms$occasions without having to
    #    pass in another parameter
        SITE <- data[,w+1]; PATCH <- data[,w+2]; patch.occ <- data[,w+3]
        data <- data.frame(data2, SITE, PATCH, patch.occ);

  # Uses all possible pairs (of patches) at each site involved in the node
     #     (done one site at a time)
      p.trueBi <- 0
        X <- NA; Y <- NA;
         for(j in unique(data$SITE)){
           temp.X <- NA; temp.Y <- NA;
           count <- 1
           values <- data$patch.occ[which(data$SITE==j)];
           if( length(values) < 2 ){next}
            for(r in 1:(length(values)-1)){
            for(s in (r+1):length(values)){
              temp.X[count] <- values[r]    ;
```

```
              temp.Y[count] <- values[s]
              count <- count+1 }}  #end r, s loops
              X <- c(X, temp.X)
              Y <- c(Y, temp.Y)
                             } #end 'j' loop
          save <- X
          X <- c(X,Y); X <- X[!is.na(X)]
          Y <- c(Y, save); Y <- Y[!is.na(Y)]
p.trueBi <- ifelse( length(X)> 1 & (sum(X==X[1]) < length(X)), cor(X,Y), 0)
            # divide by anything?

       return(p.trueBi) }  # end 'rho' function

###########
##############  The following two functions are needed to label
#  my print output properly

print.rpart <- function(x, minlength=0, spaces=2, cp,
                digits=getOption("digits"), ...) {
    if(!inherits(x, "rpart")) stop("Not legitimate rpart object")
    if (!is.null(x$frame$splits)) x <- rpconvert(x)  #help for old objects

    if (!missing(cp)) x <- prune.rpart(x, cp=cp)
    frame <- x$frame
    ylevel <- attr(x, "ylevels")
    node <- as.numeric(row.names(frame))
    depth <- tree.depth(node)
    indent <- paste(rep(" ", spaces * 32), collapse = "")
    #32 is the maximal depth
    if(length(node) > 1) {
        indent <- substring(indent, 1, spaces * seq(depth))
        indent <- paste(c("", indent[depth]), format(node), ")", sep = "")
    }
    else indent <- paste(format(node), ")", sep = "")

    tfun <- (x$functions)$print
    if (!is.null(tfun)) {
if (is.null(frame$yval2))
yval <- tfun(frame$yval,  ylevel, digits)
else    yval <- tfun(frame$yval2,  ylevel, digits)
}
    else yval <- format(signif(frame$yval, digits = digits))
    term <- rep(" ", length(depth))
    term[frame$var == "<leaf>"] <- "*"
    z <- labels(x, digits=digits, minlength=minlength, ...)
    n <- frame$n
    z <- paste(indent, z, n, format(signif(frame$dev, digits = digits)),
```

```
              yval, term)

    omit <- x$na.action
    if (length(omit))
    cat("n=", n[1], " (", naprint(omit), ")\n\n", sep="")
    else cat("n=", n[1], "\n\n")

    #This is stolen, unabashedly, from print.tree
    if (x$method=="class")
         cat("node), split, n, loss, yval, (yprob)\n")
   # NEW PART!!!
   if (x$method=="user"){ cat("node), split, n, deviance, yval, prob.occ,
          prob.det\n") }
   #####
    else cat("node), split, n, deviance, yval\n")
    cat("      * denotes terminal node\n\n")

    cat(z, sep = "\n")
    return(invisible(x))
    #end of the theft
    }

          # This one is located in treemisc.R
tree.depth <- function(nodes)
{
    depth <- floor(log(nodes, base = 2) + 1e-7)
    as.vector(depth - min(depth))
}
```

3.6.3 Proposed method each.parent

```
# Even though it is not actually a LRT this time, the names never got changed
LRT.each.parent <- function(parent, left, right, orig.prob.det,
          actual.var.occ.P, prob.occ.P, prob.det.P){
      prob.det <- prob.det.P

      # LEFT
      Ns.left <- timesseen(left, parms)

      # RIGHT
      Ns.right <- timesseen(right, parms)

       outcome.left <- optim(c(1), lnl.t.fixed, gr=NULL, method="BFGS",
          control=list(fnscale=-1), hessian=FALSE,
          prob.det = prob.det.P, Ns=Ns.left)
         param.left <- backt(outcome.left$par)
         prob.occ.L <- param.left[1]
```

```
      outcome.right <- optim(c(1), lnl.t.fixed, gr=NULL, method="BFGS",
        control=list(fnscale=-1), hessian=FALSE,
        prob.det = prob.det.P, Ns=Ns.right)
        param.right <- backt(outcome.right$par)
        prob.occ.R <- param.right[1]


      ## DONE:  Generalize to 'k' occasions instead of just k=2
      #Estimate Occupancy prob/status for each patch
        k <- parms$occasions
        Eocc.given.seen <- 1
        Eocc.given.notseen.L <- (prob.occ.L*(1-prob.det)^k)/
         ( (prob.occ.L*(1-prob.det)^k) + (1-prob.occ.L) )
        Eocc.given.notseen.R <- (prob.occ.R*(1-prob.det)^k)/
         ( (prob.occ.R*(1-prob.det)^k) + (1-prob.occ.R) )
        patch.occ <- 0;
        for(i in 1:nrow(left)){
           patch.occ[i] <- ifelse( sum(left[i,1:parms$occasions])==0 ,
            Eocc.given.notseen.L, Eocc.given.seen) }
        left <- cbind(left, patch.occ)
  #colnames(left) <- c("X2002.1", "X2002.2", "SITE", "PATCH", "patch.occ")
        patch.occ <- 0;
        for(j in 1:nrow(right)){
           patch.occ[j] <- ifelse( sum(right[j,1:parms$occasions])==0 ,
            Eocc.given.notseen.R, Eocc.given.seen) }
        right <- cbind(right, patch.occ)
 #colnames(right) <- c("X2002.1", "X2002.2", "SITE", "PATCH", "patch.occ")

      #Estimate correlation between patches (rho or p)
      dummy <- rho((left), (right))
  #At one point I had problems unless left and right were in matrix format,
             #  so I used as.matrix(left)
    p.hat <-  max(0, dummy[1]);
      # some of the correlations could be negative.
      #  This is fine mathematically, but not in our conceptual framework,
      #        where we think of positive correlation within site.

      #Calculate the "patch-up" ratio
SITE <- parms$occasions+1;
PATCH <- parms$occasions+2;
patch.occ <- parms$occasions+3
C <- matrix(c(1,-1), nrow=1)
   # For testing the hypothesis that pi.occ.L = pi.occ.R
V.left <- prob.occ.L*(1-prob.occ.L); V.right <- prob.occ.R*(1-prob.occ.R)
V.each.side <- matrix(c(V.left, V.right), nrow=2)

        max.patches.per.site <- max(parent[,PATCH])
```

```
  X <- matrix(0, nrow=(max.patches.per.site*max(parent[,SITE])), ncol=2)
      # 2 columns b/c X-matrix tells if the site goes L or R in the split
   i = 1
   for(j in unique(parent[,SITE])){
   temp.left <- sum(left[,SITE]==j)
   temp.right <- sum(right[,SITE]==j)
   if(temp.left==0 & temp.right==0){i = i+max.patches.per.site; next }
   if(temp.left == 0){ X[(i:(i+temp.right-1)), 2] <- 1 ;
                       i=i+max.patches.per.site; next}
   if(temp.right == 0){ X[(i:(i+temp.left-1)), 1] <- 1 ;
                       i=i+max.patches.per.site; next}

   X[(i:(i+temp.left-1)), 1] <- 1 ;
   X[(i+temp.left):(i+temp.left+temp.right-1),2] <- 1 ;
            i = i+max.patches.per.site
                } # end 'j' loop

   V.each.patch <- X%*%V.each.side
W = c(1/V.each.patch)
         W[which(W == Inf)] <- 0
         W <- diag(W)

  D = diag(c(V.each.patch))

  SD.matrix <- diag(c(sqrt(V.each.patch)))

    start.matrix <- diag(rep(1, nrow(X)/max.patches.per.site))
    Corr.block <- matrix(p.hat, nrow=max.patches.per.site,
                                ncol=max.patches.per.site)
     diag(Corr.block) <- 1
      Corr.matrix <- kronecker(start.matrix, Corr.block)

  # VC.matrix <- SD.matrix%*%Corr.matrix%*%SD.matrix
    SD.vector <- c(sqrt(V.each.patch))  # No. 2.  (also needed for No.3)
    matrix1 <- SD.vector * Corr.matrix  # No. 2
    VC.matrix <- SD.vector * t(matrix1)  # No. 2

   # VC.matrix <- Corr.matrix * tcrossprod(SD.vector)  # No. 3

    # Using WLS estimates at the moment
   Var.correct <- try(C %*% solve(t(X)%*%W%*%X) %*%
     (t(X) %*% W %*% VC.matrix %*% W %*% X)
     %*% solve(t(X)%*%W%*%X) %*% t(C) ,
      silent=TRUE)   #this turns off the errors printing
      if(class(Var.correct)=="try-error"){Var.correct <- 0; }
   Var.naive <- try(C %*% solve(t(X)%*%W%*%X) %*% t(C) , silent=TRUE)
      if(class(Var.naive)=="try-error"){Var.naive <- 1; }
```

```
    patchup <- as.numeric(Var.correct/Var.naive)
             #"Correct" var over "Naive" var

       #Wald Statistic
     if(prob.occ.L == prob.occ.R){ test.stat <- 0}
      # prevents 0/0, especially if both estimates are on the boundary
naive.var.of.diff <- actual.var.occ.P*(nrow(parent))*(1/nrow(left) +
                                                        1/nrow(right))
     actual.var.of.diff <- naive.var.of.diff*patchup
       ifelse(actual.var.of.diff > 0,
       test.stat <- abs(prob.occ.L - prob.occ.R) / sqrt(actual.var.of.diff),
            test.stat <- 0)
     out <- c(test.stat, prob.occ.L, prob.occ.R, prob.det.P, prob.occ.P)
      return(out)  }
```

3.6.4 Run code

```
############################
# To get observed GEE testing data FROM MCGARIGAL & MCCOMB

library(rpart);
birds27 <- read.csv("C://Documents and Settings/Owner/My Documents/
                  Research Part II/GEE approach/bird.visit.csv", header=T)
 # Columns 1-11 are identifiers (basin, site, patch, occasion)
 #     and survey-day variables
 # Columns 12-89 are bird species and counts of how many seen
hab.data <- read.csv("C://Documents and Settings/Owner/
         My Documents/Research Part II/GEE approach/hab.sta.csv", header=T)
    # Columns 1-3 are identifiers (basin, site, patch)
    # Columns 4-24 are station (patch)-level covariates.
    #   ***Columns 9,10,11 are all factors/categorical,
    #         even though only 10 and 11 are listed as factors.
    #        may need to change 9 (s.id) to as.factor(col9).
    #        **BUT, don't use s.id in CART...

#  To get data in the right format for user splits:

# Arbitrarily choose a basin
  step1 <- birds27[birds27$basin=="D",-c(1,5,6,7,8,9,10,11)]

# To get sites (sub-basins) as numbers from 1 to Max.number.of.sites
  step2 <- step1
  step2$sub <- as.numeric(step2$sub)
  count <- 1;
   for(i in unique(step2$sub)){
      step2$sub[which(step2$sub==i)] <- count;
      count <- count+1 ;
                       }
```

```
# To get the four observations into one row
  #  e.g. NOT ARBITRARILY I shall choose the species GCKI,
  #     the Golden-crowned Kinglet
  column <- step2$GCKI
  column[which(column>=2)] <- 1
       # THIS CHANGES ANY 2+ "number seen" count into a "1" for just 'seen'
  ydata <- matrix(NA, nrow=(nrow(step2)/4), ncol=4)
    for(i in 1:(nrow(step2)/4)){
       ydata[i, ] <- column[(4*i-3) : (4*i)] }


# To get sites and sub-basins from 4 repeats to only one
   site <- step2$sub[seq(1, nrow(step2), 4)]
   patch <- step2$sta[seq(1, nrow(step2), 4)]

# Final (response) data frame for entry into CART
#  bird.data <- data.frame(ydata, site, patch)

# Final (covariate) data for entry into CART
  print("NOTE THE CHOICE OF BASIN AGAIN")
  covariates <- hab.data[hab.data$basin=="D", 4:ncol(hab.data)]
    # BUT DON'T FORGET TO NOT USE s.id (as a covariate)
  covariates <- covariates[,-which(colnames(covariates)=="s.id")]

  total.data <- data.frame(ydata, site, patch, covariates)

  # Try using GEE.4c for EV (OLS)
  # source("C:/Documents and Settings/Owner/My Documents/Research Part II/
   GEE approach/MY attempt at user splits GEE 4c.R")

  # Currently using GEE.5 for UV (WLS) !!!
  #Clustered:
  #source("C:/Documents and Settings/Owner/My Documents/Research Part II/
   GEE approach/MY attempt at user splits GEE 5.R")
  #Indpt:  (makes p.hat = 0 in all cases)
  #source("C:/Documents and Settings/Owner/My Documents/Research Part II/
   GEE approach/MY attempt at user splits GEE indpt.R")

#source("E:/Research Part II/GEE approach/
#                 MY attempt at user splits GEE 5.R")
  alist <- list(eval=temp1, split=temp2, init=temp3)
  parms <- list(cutoff=0.5, occasions=2, prob.det=NULL, mygoodness=2)
    # mygoodness:  1=orig.parent, 2=each.parent, 3=p.v.2d, 4=p.v.d.v.d.
  date()
  fit.user <- rpart( cbind(X1, X2, site, patch) ~  elev +  slope + aspect +
   s.edge + p.edge, data=total.data, method=alist, parms=parms, cp=0.01 )
  date()
```

```
   # THERE ARE 2 COLUMNS NAMED p.edge!! (in hab.data)
   # My code is getting the categorical one (1st one to appear).
   #     The quantitative one is renamed "p.edge.1"        #
fit.user
 #fit.user2 <- prp(fit.user, snip=TRUE)$obj
 #fit.user.indpt2 <- prp(fit.user.indpt, snip=TRUE)$obj
par(mfrow=c(1,2), xpd=NA)
plot(fit.user, main="Clustered")
text(fit.user, use.n=TRUE)
plot(fit.user.indpt, main="Independent")
text(fit.user.indpt, use.n=TRUE)
```

CHAPTER 4. Pruning Classification and Regression Trees with Modifications for Occupancy Modeling

Modifications to methods of growing trees (i.e. in how nodes are split) in Classification and Regression Trees (CART) lead to potentially different trees. Similarly, those modifications mean that the methods of pruning the trees may need their own changes. We previously proposed methodology to incorporate imperfect detection and correlated data into the splitting mechanisms of CART, which led to our re-evaluation of pruning criteria. Here we discuss 5 pruning criteria that could be used with our CART methodology. Simulated examples for each criteria [run separately] result in error rates that are used to assess the performance of the pruning criteria.

4.1 Introduction

In Chapter 2, we introduced methods for including detection probability in the process of growing the tree. Each method estimates detection and occupancy probability slightly differently, but all of the methods involve the use of a Likelihood function to estimate the parameters at each node in the tree. The methods address fundamental idea that you cannot ignore imperfect detection while modeling without suffering a loss of accuracy in the resulting estimates. Estimates of occupancy probability are consistently lower when using the naïve tree than those computed using detection-adjusted trees. Accounting for imperfect detection is especially crucial when occupancy is modeled using a CART tree, due to the potential for compounding mistakes made early on in the tree process.

In Chapter 3, we turn to another common issue that can affect CART: observations that are clustered (either spatially or otherwise related). The characteristics of clustered individuals are

likely to be correlated in some way. Therefore, any model (e.g. CART) examining data where independence is in doubt should account for that [possible] correlation. The traditional CART model assumes independence, so once again there arises the potential for error in the CART process, this time in the presence of data that is not known to be independent. Clustered data affects the variance of parameter estimates in a non-constant manner based on the splitting situation and the relationship of clustered patches within the tree. We proposed an approach based on generalized estimating equations (GEEs) that involved "patching up" the naïvely-calculated variance estimates prior to their use in a statistical test.

When growing a CART tree using the methodology presented in the previous chapters, there are many potential rules for stopping and/or pruning the tree. Atkinson and Therneau (2011) implement some of these into *rpart()*. One of the most basic statistical constraints is that of node size. This constraint specifies the minimum parent node size required to even consider splitting, as well as setting the minimum daughter node size required to accept a proposed split.

Another pruning mechanism involves a choice of a value for the complexity parameter (cp). The cp value can be used both during (as a stopping rule) and after (as a pruning criteria) the formation of the tree. During the creation of the tree, an estimated complexity value (cp*) is calculated for each potential split. While the specific details of this calculation are hidden within the software, if the estimated complexity cp* is not as large as the specified level (cp), then the split being considered is not made (during the tree creation) or else the branch in question is pruned off (after the tree creation).

A more-advanced option requires the user to implement the "user splits" option of rpart(). User splits is a set of three functions which allow for the use of non-traditional node evaluation and splitting methods. In the *eval* function, there is a calculation for node deviance. When using the "user splits" option for a classification tree, the default measure of deviance at a node in *rpart()* is a simple misclassification count. Deviance (D), along with the complexity parameter, are then used together in a cost-complexity pruning algorithm. Cross-validation is also affected by the choice of a node impurity measure and the complexity parameter (Therneau and Atkinson 2011).

In situations with imperfect survey detection or correlated observations, such as were discussed in the previous chapters, some of the methods for stopping and/or pruning the tree have a potential for error. The misclassification measure used to measure node deviance is clearly suspect when the correct status of an individual is unknown. The estimated complexity value of cp* may also be influenced by the presence of imperfect detection, in addition to being affected by spatial dependence among the individuals. Therefore, using these two measures as a decision tool for when to stop growing the tree, or for how far to prune the tree back after it has been already grown, is not a wise idea.

Instead, we present a different method for pruning that is based on the same method used to grow the tree. This method utilizes some of the existing features of *prune.rpart()*, but with the twist that pruning is no longer based on a chosen cp value. Rather, a set of pruning criteria are proposed to be used with a fully-grown tree (no internal pruning during the growth of the tree). Internal pruning can be turned off completely by specifying cp=0 (in the call to *rpart()*) and node deviance (impurity) as $(nodesize)^2$ (in the *eval()* function of user splits). We apply our methodology to two simulated data sets, which will demonstrate the potential ability of the pruning algorithm to discover and choose the sub-tree with the least predictive error.

4.2 Methods

We assume a scenario such as that described in the "Birds" example of Chapter 3, wherein we are potentially dealing with imperfect detection and correlated observations in the same data set. The confluence of these two problems allows us a wide range of potential statistics to use in a pruning algorithm. This paper addresses a few of the more accessible and easier-to-understand statistics.

Recall from Chapter 3 that a proposed solution for dealing with a situation involving both imperfect detection and correlated observations was based upon the ideas of generalized estimating equations, likelihood functions, and a Wald test. Under the assumption of independence, a likelihood function estimates the occupancy and detection parameters, as well as their variances, at the three nodes of the tree involved in a given split. We then "correct" the naïve variance estimates by using the correlation found in the sample data set. The result is a mod-

ified Wald test statistic (Equation 3.2) that indicates the relative strength of a proposed split from a parent node (when compared with all other proposed splits). For more details on the splitting methodology, please refer back to Chapter 3.

The set of potential pruning criteria chosen were:

Node Size: Using the minimum daughter-node size as it is used in rpart(). Somehow, setting $cp = 0$ and the node deviance $= (nodesize)^2$ causes the minimum daughter-node size constraint to no longer be referenced during tree creation (it is an override of sorts). Thus, we cause the original node size pruning criterion to be applied during our prune function.

Criterion A: Prune back leaves based on the test statistic that was used in the split creation. Similar to the idea of comparing cp* to a specified value of cp, we compare the estimated test statistic with a specified level, and prune those leaves if the estimated value fails to exceed the specified level. However, the calculation of the estimated test statistic is a known quantity (see Chapter 3 and 3.2), while the estimated cp* depends on a variety of factors which are not easily identifiable.

Criterion B: Prune based on the measure of impurity at each node. The default node impurity in *rpart()* is a misclassification measure. In trees, the desire is to decrease the overall impurity at each step in the trees' growth. Thus, we set a "cutoff point" which specifies how much the impurity must decrease from the parent node to the two daughter nodes. Unfortunately, we note that in an imperfect detection setting, it is possible to have a calculation result in *increased* impurity, so the "base" level of misclassification (B=0) can now result in pruning when applied to a fully grown tree.

Criteria C and D: The methodology from Chapter 3 was developed with an occupancy/detection framework in mind, so those concepts have also been applied to the pruning process. Once again, there are specified "cutoff" points for the estimated occupancy and detection probabilities, beyond which the leaves are pruned. It is important to note that C and D work together and not separately. For example, there may be nothing wrong with a node that specifies occupancy probability as 0.99, but there is likely an issue if that some node also estimates the detection probability as 0.01.

4.2.1 Implementation

The pruning function assumes that one has first used the *user splits()* option of *rpart()* to create a fully-grown tree. The pruning function requires as inputs the tree object and a set of original data (contains the data you passed into user splits in the "Y" part of the model (e.g. $Y \sim X1 + X2$). For most data sets, this original data will contain the multiple-occasion survey results (one occasion per column), site number and patch number (if clustering is applicable)). Three companion functions are also required:

Goodness: A measure of the strength of a proposed split. You must use the same function that was used in the creation of the tree (which may be referred to as a "drop in deviance"). It is necessary to call this function again, because the information created during the tree growth is not easily accessible.

Node deviance: A measure of node impurity. The growth of the original tree [using this method] requires specification of node deviance = $(nodesize)^2$. However, for pruning, this measure can be defined by the user, and has a default of naïve misclassification.

Decision: Using criteria A, B, C, D as described above, this function returns a simple Yes/No on whether the current leaves in question should be pruned. The user can easily add their own criterion or modify the current ones.

Through a "rollup" process, our algorithm isolates node "triads" (a parent and two terminal daughter nodes), re-computes the original splitting information, and then applies the Decision function to that triad. If the decision is to prune, then the *snip.rpart()* function is used to update the tree, resulting in the parent node now becoming a terminal node. If the decision is not to prune, then the nodes of that triad, as well as all ancestor nodes of that triad, are kept in the final tree. This rollup continues until all potential triads have been checked for pruning, and then returns the newly pruned tree.

4.2.2 Simulated Scenarios

We performed simulations to test the usefulness of our pruning algorithm in situations with designed clustered observations with imperfect detections. Forty iterations of each scenario

are averaged to produce final statistics. Each of the simulations involves the use of 10 sites and a total of 342 patches distributed fairly evenly [30 to 37 patches per site] across the sites. This corresponds to the original situation in the Birds example from Chapter 3 (taken from McGarigal and McComb (1995)).

The original CART model from Chapter 3 used five of the covariates described by McGarigal and McComb (1995): elevation (in meters), slope (percent slope), aspect(degrees,0-360), stand edge (indicator for whether the plot center is within 50 m from the nearest seral stage edge), and patch edge (indicator for whether the plot center is within 50 m from the nearest patch edge). To keep as close as possible to the previous model, each of our simulations will involve the same five covariates (in name), although their values will change for each of the 40 iterations of the simulation.

We introduce dependence into the simulation by clustering patches within each site. Occupancy probabilities are assigned to each patch, and imperfect detection is included through a fixed probability of detection for all patches ($\pi_{det} = 0.8$). Then the observed values (detected or undetected) for two occasions are generated for each of the 342 patches. These observed values are then used to create a classification and regression tree using *rpart() and the methodology introduced in Chapter 3*. Validation data sets are created to test the pruning algorithm on each tree.

First, dependence is induced through the clustering of patches on each of the 10 sites. We simulated coordinates for the "center" of each of the 10 sites using two uniform distributions (one each for X and Y, for a total of 10 pairs of simulated coordinates). To represent the patch (within-site) variation, we used normal distributions nested with each site. In other words, for site i and patch j,

$$Y_i \sim \text{Uniform}(0,10)$$

$$X_i \sim \text{Uniform}(0,5)$$

$$Y_{ij} \sim Y_i + \text{N}(0,\ 0.3)$$

$$X_{ij} \sim X_i + \text{N}(0,\ 0.15)$$

The final coordinates for each of the 342 patches are used to assign a probability of occupancy (based on their location in Designs A or B (Figure 4.1). Simulated X and Y values that fall below zero are treated as zeros and values above 10 (for Y) or 5 (for X) are treated as 10 or 5 respectively.

The patch occupancy probabilities are then used to simulate two observed values (detected or undetected) under a probability of detection (π_{det}) of 0.8.

To create a classification and regression tree using *rpart()*, the two simulated observed values at each patch were treated as the response variable, and were modeled using five covariates: three "original" variables from the Birds model (elevation, stand edge, and patch edge) plus the simulated X and Y values that were used to assign occupancy probabilities.

Validation sets were created by bootstrapping the 5 covariates (X, Y, elevation, stand edge, and patch edge) to create values for a new set of 342 patches. The new covariate information for X and Y is then used to re-simulate the patch occupancies and observed values (still using Figure 4.1, the dependence structure, and $\pi_{det} = 0.8$). We then applied the proposed pruning algorithm to the validation data and the tree created from the "original" data. The pruning algorithm is applied to both designs in each of the following situations: Criterion A only, Criterion B only, Criterion A with Criterion B set equal to zero, and several simulations involving various levels of Criteria C and D. Every simulation included the node size pruning constraints, which were kept constant (minimum parent node size was 21, minimum daughter node size was 7). For each criterion, we tested a variety of specified "cutoff" levels, which can be seen in the tables shown in the Results section below.

The recorded response for each iteration of the simulation is the mean absolute error (MAE) between the predicted occupancy probabilities (from CART) and the "true" occupancy probabilities (from Figure 4.1). Results were then averaged over the 40 simulations for each "cutoff" level being examined.

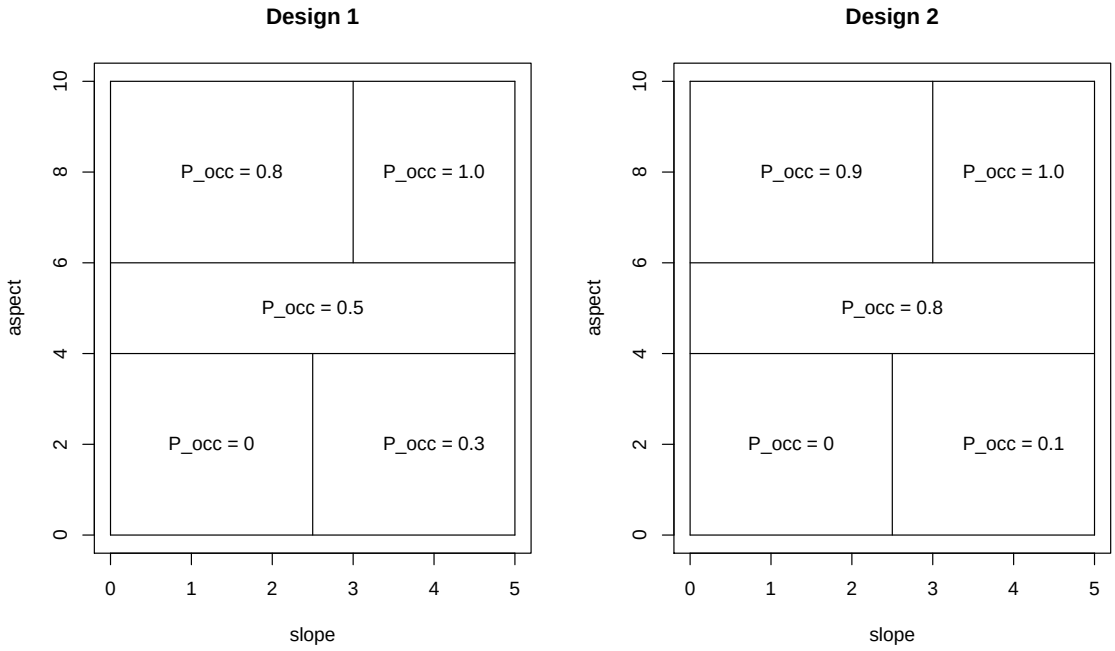

Figure 4.1 Visualization of the two designs which assign occupancy probabilities to each patch during the two simulations. The X and Y axes correspond to the "slope" and "aspect" variables, respectively. Design 1 is used in the first simulation for each criterion and Design 2 is used in the second simulation.

4.3 Results

In each set of results, there is a "base" level of pruning which should not result in any pruning other than what is caused by the node size constraint. The node size constraint is applied in every simulation. We note that in general, the trees produced contain between 25 and 30 potential pruning points (e.g. starting from the full tree and working back to a tree with a single node, we could remove pairs of terminal nodes approximately 25 to 30 times before arriving back at the original starting node).

For the simulations involving levels of Criterion A only (Figure 4.2), the simulation based on Design 1 results in a "base" (i.e. A=0) mean absolute error of 16%, then exhibits a slight upward tick before gradually improves until we hit a low of 9.5% when the test statistic cutoff is set to 4, and then we begin to steadily lose accuracy due to over-pruning. Design 2 yields similar results, except that we see nearly identical values for the "base" error rate as for the

best error rate (9.6% at A=4.5) than occurred with Design 1. We also see a steep increase in error rates as the cutoff value increases. The slightly curved patterns seen in the mean absolute errors are consistent with patterns of error estimates mentioned by Breiman et al (1984). The following is direct from Segal (1995): *"Finally, in CART (i.e. Breiman et al 1984) Section 3.4.3, the authors report their experience that 'honest' error estimates exhibit a characteristic pattern as a function of tree size. The estimates have a rapid initial decrease followed by a long flat valley and then a gradual increase as tree size increases. The minimum error or cost occurs in the valley region, but its position within this region is unstable. So, an heuristic for choosing tree size is to look at plots of error versus tree size and identify the tree size where there is a change in slope after the initial decrease."*

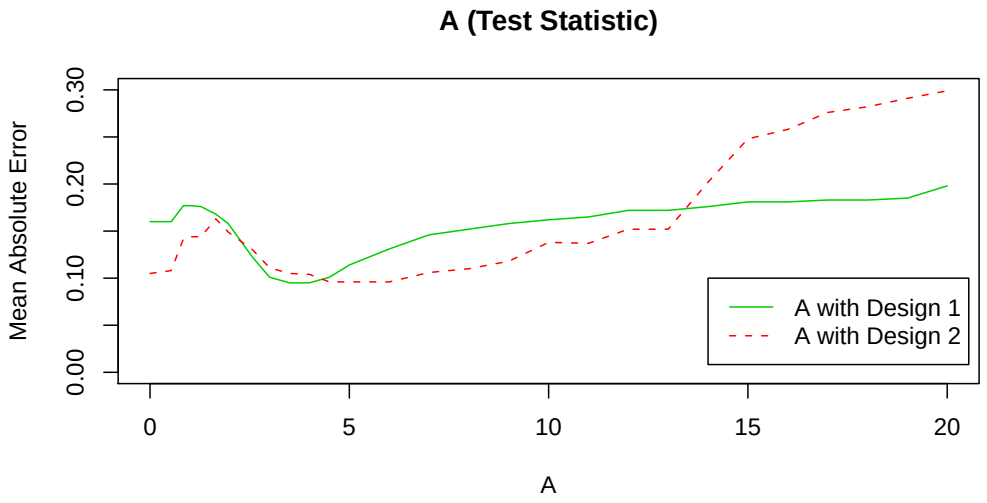

Figure 4.2 The averaged results from simulations using Design 1 and Design 2 (Figure 4.1) with multiple levels of criterion A. "A" is the specified level of the test statistic, which is compared to the calculated test statistic (Equation 3.2). If the calculated test statistic for a given split (from a parent to two terminal nodes) fails to exceed the specified level, then that split in the tree is pruned back. Error estimates for each of the specified levels of the test statistic are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

For the simulations involving levels of Criterion A with Criterion B set equal to zero, the pruning rates have increased, but only at the lowest levels of Criterion A (Figure 4.3). As the levels of Criterion A increase, pruning is dominated by criterion A rather than the single level

of criterion B. Unfortunately, incorporating B=0 as an additional cutoff point for pruning has actually made the error rates *higher* overall when compared to the Criterion A-only, Design 1 situation (Figure 4.4)! It seems to have a very slight (yet inconsistent) under Design 2.

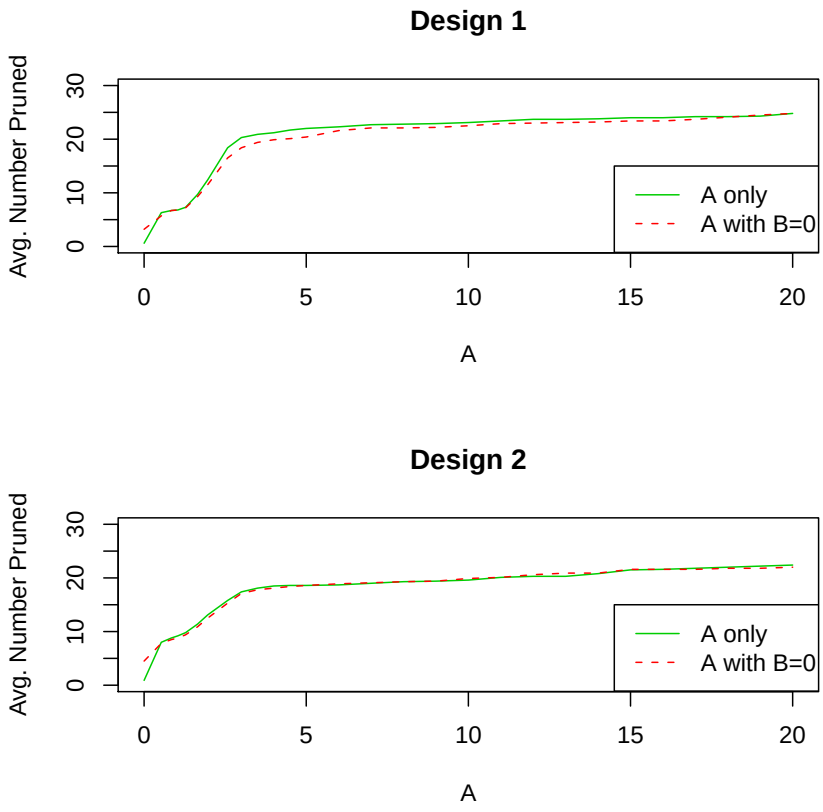

Figure 4.3 Comparison of the average number of pruning decisions resulting from using only Criterion A versus using Criterion A with Criterion B equal to zero. A "pruning decision" reduces two "sister" terminal nodes to their parent node. The top panel shows the results from Design 1, while the bottom panel contains results from Design 2.

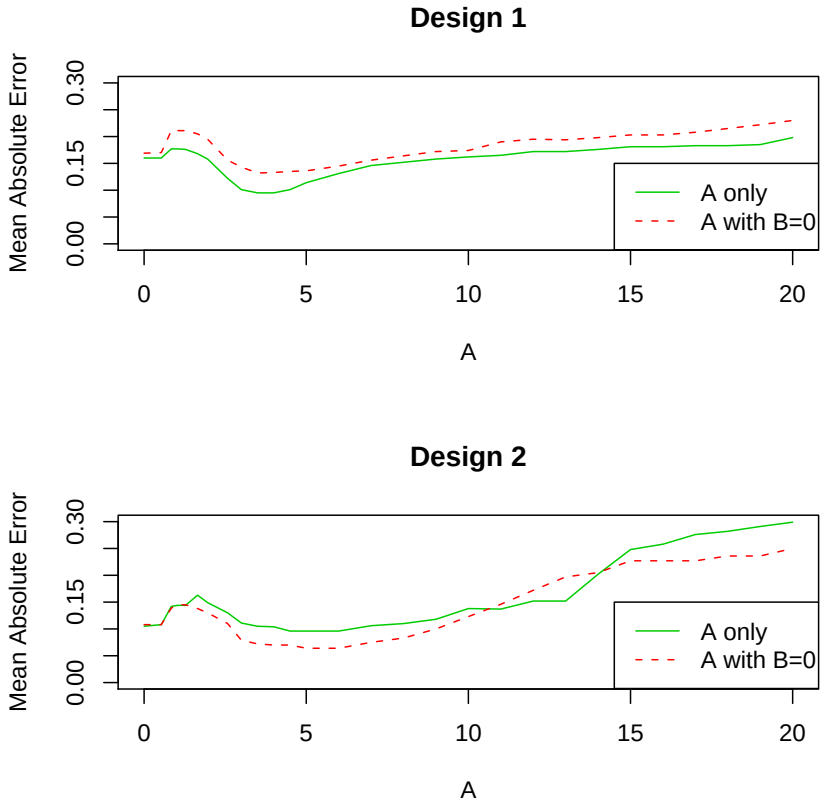

Figure 4.4 Comparison of the mean absolute errors resulting from pruning using only Criterion A versus pruning using Criterion A with Criterion B equal to zero. The top panel shows the results from Design 1, while the bottom panel contains results from Design 2.

For the simulations involving levels of Criterion B (Figure 4.5), the "curvature" that was on display in the previous results has noticeably diminished. In both Design 1 and 2, the difference between the error rates at the lowest ("base") levels of B and those at the middle levels of Criterion B (with the lowest absolute errors) is around 3%, whereas the simulations which varied the level of Criterion A generally displayed larger ranges (Figure 4.2). For Criterion B on its own (Figure 4.5), the minimum error rates for both Design 1 and Design 2 are 11.5% and 6.6% respectively. After the large jump in number of times a pair of terminal nodes were pruned (from B=0 to B=1), both Designs 1 and 2 exhibit a long, relatively flat period in mean absolute error rates that very gradually increases with the larger values of Criterion B. Pruning with Criterion B displays a much smaller error rate when used with Design 2 that when used

with Design 1. Note that the "base" level of Criterion B (B=0) still actually allows pruning to occur above and beyond that caused by the node size criterion.

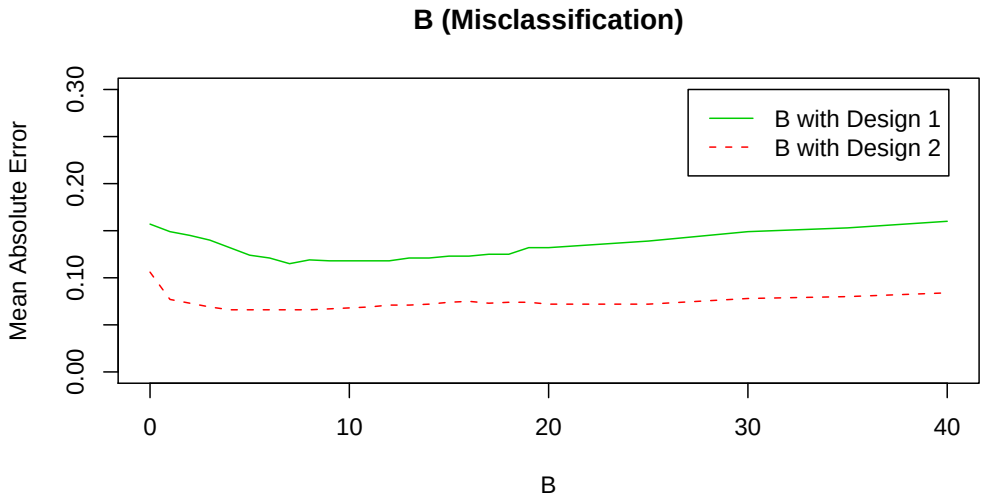

Figure 4.5 The averaged results from simulations using Design 1 and Design 2 (Figure 4.1) with multiple levels of criterion B. "B" is the specified level of the deviance for the proposed split using the "misclassification" criterion, which is compared to the calculated deviance of the split. If the deviance for a given split (from a parent to two terminal nodes) fails to exceed the specified level, then that split in the tree is pruned back. Error estimates for each of the specified levels of deviance are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

For the simulations involving levels of Criteria C and D, results based on Design 1 (Table 4.1) and those based on Design 2 (Table 4.2) were inconclusive. The changing levels did not result in much pruning, and therefore also did not result in much difference among the resulting error rates for the trials. We speculate first that the combined nature of the pruning criteria will in general lead to less pruning. In addition (and perhaps more relevant to this specific issue), consider that the occupancy probability simulations tend to result in fewer extreme values (in general), when using either Design 1 or Design 2. This outcome, combined with the assignation of a detection probability of 0.8 to all patches during the simulations, leads to very few nodes in the trees that have high occupancy and low detection estimates.

Table 4.1 Averaged results from a simulation using Design 1 (left-most picture in Figure 4.1) with multiple levels of criteria C and D. "C" is the specified level of the probability of occupancy criterion and "D" is the specified level of the probability of detection criterion, which are compared to the calculated measures. If a calculated probability of occupancy for a terminal node exceeds the specified level AND the calculated detection probability falls below the specified level, then that split in the tree is pruned back. Error estimates for each pair of the specified levels of occupancy/detection are shown.

C	D	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
1.00	0.00	0.8	1.114	0.164	0.035	0.006
0.95	0.01	0.8	1.114	0.164	0.035	0.006
0.95	0.05	1.4	1.614	0.186	0.063	0.010
0.95	0.10	1.4	1.599	0.186	0.063	0.010
0.95	0.50	1.5	1.601	0.186	0.063	0.010
0.70	0.01	3.4	2.967	0.164	0.035	0.006
0.70	0.05	4.2	3.256	0.206	0.087	0.014
0.70	0.10	4.3	3.234	0.206	0.087	0.014
0.70	0.50	4.3	3.215	0.205	0.087	0.014

Table 4.2 Averaged results from a simulation using Design 2 (right-most picture in Figure 4.1) with multiple levels of criteria C and D. "C" is the specified level of the probability of occupancy criterion and "D" is the specified level of the probability of detection criterion, which are compared to the calculated measures. If a calculated probability of occupancy for a terminal node exceeds the specified level AND the calculated detection probability falls below the specified level, then that split in the tree is pruned back. Error estimates for each pair of the specified levels of occupancy/detection are shown.

C	D	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
1.00	0.00	0.8	0.974	0.116	0.034	0.005
0.95	0.01	0.8	0.974	0.116	0.034	0.005
0.95	0.05	1.1	1.231	0.132	0.059	0.009
0.95	0.10	1.2	1.238	0.133	0.060	0.009
0.95	0.50	1.2	1.238	0.133	0.060	0.009
0.70	0.01	4.9	3.467	0.116	0.034	0.005
0.70	0.05	5.6	3.642	0.149	0.094	0.015
0.70	0.10	5.7	3.648	0.149	0.094	0.015
0.70	0.50	5.7	3.661	0.147	0.094	0.015

4.4 Discussion

Examining the Results collectively, we note that the simulations which use Design 2 to generate probabilities of occupancy generally result in lower mean absolute errors estimates than the corresponding simulations using Design 1. This may be due to the more distinct groups of occupancy probabilities (either very low or very high) used in Design 2, as opposed to the probabilities in Design 1 which are more varied across the parameter space. In general, this also leads to a smaller potential improvement in error rates when applying pruning, so we see more shallow "curvature" in the resulting error rates.

Recall the lack of interesting results based on pruning with Criteria C and D (Tables 4.1 and 4.2. It was previously mentioned that the occupancy probability simulations tend to result in fewer extreme values (in general) for both Design 1 or Design 2. This outcome, combined with

the assignation of a detection probability of 0.8 to all patches during the simulations, leads to very few nodes in the trees that have high occupancy and low detection estimates. This explains why the numbers of pruned terminal node pairs were so low; despite running the cutoff out to 0.5 for each Criterion, the detection probability criterion (D) likely needed to be increased even more to allow for the higher estimates of that parameter. In future simulations, we would treat the probability of detection similar to the probability of occupancy–as a simulated value for each patch rather than a constant. This would allow for detection probability to approach the edge of the parameter space, particularly near zero where we would be tempted to prune leaves from the tree.

I do note the relatively small number of trials (40) for each simulation. The computation time involved in running the simulation was quite long due to the number of times it was necessary to compute a CART tree, which is already one of the drawbacks of computing a single CART tree, let alone multiple trees. The number of trials could be easily increased with more computing power.

4.4.1 Recommendations

Node size: This is directly tied to the normal workings of *rpart()*, where the default minimum daughter-node size is 7 individuals. Do not change this unless you have an extremely small sample, or if your current tree is not separating known classified objects well enough.

Criterion A: It is possible that the use of the A pruning criterion could result in making the B and C/D rules obsolete (they may all indicate pruning, but only one is needed). Most trees will have a sort of "sweet spot" in the number of terminal nodes where prediction accuracy is the best. Because the behavior of A on the efficacy of all trees is unknown, we recommend choosing a cutoff value of 3.5 to begin with. This can be adjusted if you have a desired tree size in mind.

Criterion B: This is a naïve test based on having perfect detection. In an imperfect detection situation, its usefulness is limited because there is no "base" level of pruning (even negative cutoff values could still cause pruning!), and we recommend avoiding the use of this criterion. Related to this paper, it seems to have an adverse affect in situations such as Design 1, where

the actual probabilities of occupancy are quite varied. If there is reason to use this Criterion (i.e. known perfect detection or close to it), then we recommend pruning at a level of B=1 (i.e. any split that does not decrease misclassification by at least 1 should be discarded. Splits where this does not happen are likely not very good) and increasing that level if needed.

Criteria C and D: Despite the poor indications from the simulated results, the use of these two criteria can still be very important (albeit of more limited use). The optimization method for maximizing the node likelihood has a tendency to favor nodes with a high probability of occupancy and a low probability of detection when there are a lot of individuals with a "not seen" designation in their observed values. In many cases, this is unsettling if not completely unbelievable. Our recommendation is to set these levels at probability of occupancy > 0.90 and probability of detection < 0.10, although to be conservative, blind application of this rule might function better with 0.95 and 0.05 as the chosen cutoff values. We previously mentioned the reduction in pruning when two criteria are used at the same time. One way around this would be to use each criteria separately, although that is not recommended in this situation, as occupancy and detection probabilities are tied together in any imperfect detection scenario.

4.5 Conclusion

Of our proposed modifications to the pruning process, it appears that the use of the test statistic for pruning has the most potential. The test statistic incorporates all aspects of the statistical model (imperfect detection, correlated observations, etc.), and therefore is a good overall choice for a pruning criterion that also reflects the growth process. The user may need to adjust the level of pruning based on their specific situation, much as is currently done now in other CART programs and algorithms (e.g. *rpart()* and the complexity parameter). However, in using these specific methods of pruning, it is also essential to include the node size criterion, lest the user end up with terminal nodes of extremely small size. The occupancy and detection pruning criteria may be less useful in the case of correlated data than they are in a case involving imperfect detection, but they can add value to the pruning by focusing on the small but important details within the test statistic. Regardless of whether a researcher follows the methodologies laid out in this paper in Chapters 2 and 3, we strongly encourage anyone using

modified CART methodologies to also consider the ramifications of using the default pruning algorithms involved with CART and decide if they should also modify the pruning process to match the growth process.

4.6 Literature Cited

Breiman, L., Friedman, J., Olshen, R., and Stone, C. 1984. Classification and Regression Trees. Wadsworth International Group, Belmont, CA.

McGarigal, Kevin and McComb, William C. 1995. Relationships Between Landscape Structure and Breeding Birds in the Oregon Coast Range. *Ecological Monographs*. Vol. 65, No. 3, pp. 235-260.

Segal, Mark R. 1995. Extending the elements of tree-structured regression. *Statistical Methods in Medical Research*. Vol. 4, p. 219-236.

Therneau, Terry and Atkinson, Elizabeth. R port by Brian Ripley. 2010.
rpart: Recursive Partitioning. R package version 3.1.-46.
http://CRAN.R-project.org/package=rpart

Therneau Terry M, Atkinson, Elizabeth J. 2011. An Introduction to Recursive Partitioning Using the RPART Routines. Mayo Clinic, Rochester.
Available at http://r.789695.n4.nabble.com/attachment/3209029/0/zed.pdf

4.7 Extra material: Detailed simulation results for Criterion A, Criterion B, and Criterion A with Criterion B set equal to zero

For the simulations involving levels of Criterion A only, the simulation based on Design 1 (Table 4.3) results in what seems to be a more symmetrical "curvature" of the mean absolute error values. A "base" (i.e. A=0) mean absolute error of 16% moves irregularly at first, then gradually improves until we hit a low of 9.5% when the test statistic cutoff is set to 4, and then we begin to lose accuracy due to over-pruning up to an error of about 20%. Design 2 (Table 4.4) yields similar results, except that we see a much shallower decrease from the "base" error rate to the best error rate (9.6% at A=4.5) than occurred in Design 1. We also see a steep increase in error rates as the cutoff value increases. Interestingly, the "base" error rates in the Design 2 simulation are quite close to the best error rate.

Table 4.3 The averaged results from a simulation using Design 1 (left-most picture in Figure 4.1) with multiple levels of criterion A. "A" is the specified level of the test statistic, which is compared to the calculated test statistic (Equation 3.2). If the calculated test statistic for a given split (from a parent to two terminal nodes) fails to exceed the specified level, then that split in the tree is pruned back. Error estimates for each of the specified levels of the test statistic are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

A	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
0.000	0.6	0.675	0.160	0.044	0.007
0.530	6.3	3.510	0.160	0.047	0.008
0.840	6.7	3.517	0.177	0.080	0.013
1.040	6.8	3.463	0.177	0.080	0.013
1.280	7.3	3.300	0.176	0.079	0.013
1.650	9.7	3.611	0.168	0.074	0.012
1.960	12.4	3.471	0.158	0.076	0.012
2.576	18.4	4.413	0.122	0.073	0.012
3.000	20.3	4.059	0.101	0.069	0.011
3.500	20.9	4.060	0.095	0.062	0.010
4.000	21.2	4.242	0.095	0.057	0.009
4.500	21.7	3.996	0.101	0.057	0.009
5.000	22.0	4.240	0.114	0.056	0.009
6.000	22.3	4.304	0.131	0.066	0.010
7.000	22.7	4.339	0.146	0.072	0.011
8.000	22.8	4.335	0.152	0.072	0.011
9.000	22.9	4.460	0.158	0.073	0.012
10.000	23.1	4.407	0.162	0.076	0.012
11.000	23.4	4.407	0.165	0.077	0.012
12.000	23.7	4.433	0.172	0.078	0.012
13.000	23.7	4.433	0.172	0.078	0.012
14.000	23.8	4.508	0.176	0.080	0.013
15.000	24.0	4.452	0.181	0.084	0.013
16.000	24.0	4.368	0.181	0.084	0.013
17.000	24.2	4.344	0.183	0.085	0.013
18.000	24.2	4.334	0.183	0.085	0.013
19.000	24.3	4.033	0.185	0.083	0.013
20.000	24.8	3.935	0.198	0.084	0.013

Table 4.4 The averaged results from a simulation using Design 2 (right-most picture in Figure 4.1) with multiple levels of criterion A. "A" is the specified level of the test statistic, which is compared to the calculated test statistic (Equation 3.2). If the calculated test statistic for a given split (from a parent to two terminal nodes) fails to exceed the specified level, then that split in the tree is pruned back. Error estimates for each of the specified levels of the test statistic are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

A	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
0.000	0.9	0.982	0.105	0.033	0.005
0.530	8.0	3.783	0.108	0.039	0.006
0.840	8.8	3.527	0.142	0.088	0.014
1.040	9.2	3.468	0.144	0.093	0.015
1.280	9.8	3.484	0.144	0.093	0.015
1.650	11.4	3.349	0.163	0.125	0.020
1.960	13.1	3.460	0.149	0.121	0.019
2.576	15.8	3.650	0.130	0.117	0.018
3.000	17.4	4.024	0.111	0.115	0.018
3.500	18.1	4.422	0.105	0.116	0.018
4.000	18.5	4.535	0.104	0.115	0.018
4.500	18.6	4.527	0.096	0.110	0.017
5.000	18.6	4.527	0.096	0.110	0.017
6.000	18.7	4.519	0.096	0.110	0.017
7.000	19.0	4.796	0.106	0.119	0.019
8.000	19.3	4.921	0.110	0.120	0.019
9.000	19.4	4.872	0.118	0.124	0.020
10.000	19.6	4.956	0.138	0.138	0.022
11.000	20.1	5.055	0.137	0.139	0.022
12.000	20.3	5.116	0.152	0.150	0.024
13.000	20.3	5.116	0.152	0.150	0.024
14.000	20.8	5.281	0.202	0.168	0.027
15.000	21.5	5.368	0.248	0.166	0.026
16.000	21.6	5.314	0.258	0.165	0.026
17.000	21.8	5.412	0.276	0.163	0.026
18.000	22.0	5.623	0.282	0.163	0.026
19.000	22.2	5.481	0.291	0.160	0.025
20.000	22.4	5.586	0.299	0.158	0.025

For the simulations involving levels of Criterion A with Criterion B set equal to zero, both the simulation based on Design 1 (Table 4.5) and the one based on Design 2 (Table 4.6) again display a noticeable "curvature" in the mean absolute error values for each trial. Also similar to the simulations involving Criterion A alone (Tables 4.3 and 4.4), the "curvature" from Design 2 is much shallower at the beginning, but with a steep increase in error rate at the end. In contrast, when comparing the best mean absolute error rates, Design 2 results in a better error rate than Design 1 (6.4% to 13.2%), and those rates appear in slightly different places (but not significantly so–the Design 2 best error rate occurs when A=5, but that error rate is only a shade better than the 7.2% seen when A=4. The base error rates were 16.9% for Design 1 and 10.8% for Design 2. For Design 2, we see a similar effect as we did for the simulation without criterion B (Table 4.4), where the base error rate is surprisingly good compared to the first few levels of Criterion A. Recall that even though B=0 is referred to as the "base" level of Criterion B (meaning it is supposed to cause no pruning), using B=0 can potentially cause extra pruning due to its potential inaccuracies when measuring node impurity in an imperfect detection setting.

Table 4.5 The averaged results from a simulation using Design 1 (left-most picture in Figure 4.1) with multiple levels of criterion A and a constant level (zero) of criterion B. "A" is the specified level of the test statistic, which is compared to the calculated test statistic (Equation 3.2). "B" is the specified level of the deviance for the proposed split using the "misclassification" criterion, which is compared to the calculated deviance of the split. If the calculated test statistic for a given split (from a parent to two terminal nodes) fails to exceed the specified level or if the calculated deviance for the split fails to exceed zero, then that split in the tree is pruned back. Error estimates for each of the specified levels of the test statistic are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

A	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
0.000	3.2	2.866	0.169	0.044	0.007
0.530	5.7	3.389	0.170	0.045	0.007
0.840	6.7	3.562	0.211	0.097	0.015
1.040	6.8	3.563	0.211	0.096	0.015
1.280	7.2	3.765	0.211	0.096	0.015
1.650	9.3	3.863	0.205	0.094	0.015
1.960	11.5	3.616	0.195	0.093	0.015
2.576	16.5	4.574	0.156	0.089	0.014
3.000	18.4	5.119	0.143	0.078	0.012
3.500	19.4	5.038	0.132	0.082	0.013
4.000	19.9	5.138	0.133	0.077	0.012
4.500	20.1	5.241	0.135	0.074	0.012
5.000	20.4	5.108	0.136	0.073	0.011
6.000	21.6	4.872	0.145	0.073	0.012
7.000	22.1	4.966	0.156	0.069	0.011
8.000	22.1	5.016	0.164	0.078	0.012
9.000	22.2	4.879	0.172	0.082	0.013
10.000	22.5	4.915	0.174	0.083	0.013
11.000	22.9	5.011	0.190	0.087	0.014
12.000	23.0	4.854	0.195	0.086	0.014
13.000	23.1	4.843	0.194	0.086	0.014
14.000	23.2	4.870	0.198	0.086	0.014
15.000	23.4	4.950	0.203	0.088	0.014
16.000	23.4	4.950	0.203	0.088	0.014
17.000	23.7	5.229	0.208	0.089	0.014
18.000	24.1	5.328	0.215	0.089	0.014
19.000	24.5	5.449	0.222	0.088	0.014
20.000	24.8	5.477	0.230	0.085	0.014

Table 4.6 The averaged results from a simulation using Design 2 (right-most picture in Figure 4.1) with multiple levels of criterion A and a constant level (zero) of criterion B. "A" is the specified level of the test statistic, which is compared to the calculated test statistic (Equation 3.2). "B" is the specified level of the deviance for the proposed split using the "misclassification" criterion, which is compared to the calculated deviance of the split. If the calculated test statistic for a given split (from a parent to two terminal nodes) fails to exceed the specified level or if the calculated deviance for the split fails to exceed zero, then that split in the tree is pruned back. Error estimates for each of the specified levels of the test statistic are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

A	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
0.000	4.5	3.071	0.108	0.028	0.004
0.530	7.8	3.851	0.108	0.028	0.004
0.840	8.5	3.850	0.139	0.074	0.012
1.040	8.8	3.827	0.145	0.086	0.014
1.280	9.4	3.947	0.145	0.086	0.014
1.650	10.9	4.104	0.138	0.085	0.013
1.960	12.5	4.495	0.130	0.087	0.014
2.576	15.3	5.205	0.110	0.081	0.013
3.000	17.1	5.528	0.079	0.054	0.009
3.500	17.8	5.565	0.072	0.037	0.006
4.000	18.1	5.463	0.070	0.038	0.006
4.500	18.4	5.443	0.070	0.038	0.006
5.000	18.6	5.610	0.064	0.017	0.003
6.000	18.9	5.550	0.064	0.018	0.003
7.000	19.1	5.745	0.075	0.056	0.009
8.000	19.3	5.811	0.083	0.074	0.012
9.000	19.4	5.853	0.100	0.101	0.016
10.000	19.9	5.688	0.123	0.128	0.020
11.000	20.1	5.618	0.146	0.144	0.023
12.000	20.6	5.995	0.172	0.159	0.025
13.000	20.9	6.312	0.197	0.164	0.026
14.000	20.9	6.269	0.205	0.165	0.026
15.000	21.6	6.156	0.227	0.168	0.027
16.000	21.6	6.156	0.227	0.168	0.027
17.000	21.6	6.156	0.227	0.168	0.027
18.000	21.8	6.324	0.236	0.169	0.027
19.000	21.8	6.324	0.236	0.169	0.027
20.000	22.0	6.383	0.250	0.169	0.027

For the simulations involving levels of Criterion B, the "curvature" that was on display in the previous results has noticeably diminished, although it is still present. In both Design 1 and 2 (Tables 4.7 and 4.8, respectively), the difference between the error rates at the lowest levels of B and those at the middle levels of Criterion B (with the lowest absolute errors) is around 3%, whereas the simulations which varied the level of Criterion A generally displayed larger ranges. The minimum error rates for both Design 1 and Design 2 are 11.5% and 6.6% respectively. After the large jump in number of times a pair of terminal nodes were pruned (from B=0 to B=1), both Designs 1 and 2 exhibit a long, relatively flat period in mean absolute error rates that very gradually increases with the larger values of Criterion B. Note that the "base" level of Criterion B (B=0) still actually allows pruning to occur above and beyond that caused by the node size criterion.

Table 4.7 The averaged results from a simulation using Design 1 (left-most picture in Figure 4.1) with multiple levels of criterion B. "B" is the specified level of the deviance for the proposed split using the "misclassification" criterion, which is compared to the calculated deviance of the split. If the deviance for a given split (from a parent to two terminal nodes) fails to exceed the specified level, then that split in the tree is pruned back. Error estimates for each of the specified levels of deviance are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

B	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
0	4.5	3.464	0.157	0.043	0.007
1	14.9	5.435	0.149	0.038	0.006
2	16.3	5.652	0.145	0.036	0.006
3	17.1	5.709	0.140	0.036	0.006
4	19.2	6.017	0.132	0.034	0.005
5	20.8	5.854	0.124	0.027	0.004
6	21.5	5.675	0.121	0.025	0.004
7	22.8	5.736	0.115	0.028	0.004
8	23.2	5.709	0.119	0.029	0.005
9	23.6	5.601	0.118	0.030	0.005
10	23.7	5.742	0.118	0.030	0.005
11	23.8	5.528	0.118	0.030	0.005
12	23.9	5.492	0.118	0.030	0.005
13	24.3	5.233	0.121	0.030	0.005
14	24.3	5.209	0.121	0.030	0.005
15	24.4	5.172	0.123	0.029	0.005
16	24.4	5.152	0.123	0.028	0.004
17	24.4	5.128	0.125	0.028	0.004
18	24.4	5.128	0.125	0.028	0.004
19	24.5	5.094	0.132	0.038	0.006
20	24.6	4.912	0.132	0.037	0.006
25	24.8	4.883	0.139	0.044	0.007
30	25.0	4.804	0.149	0.046	0.007
35	25.0	4.822	0.153	0.049	0.008
40	25.2	4.849	0.160	0.053	0.008

Table 4.8 The averaged results from a simulation using Design 2 (right-most picture in Figure 4.1) with multiple levels of criterion B. "B" is the specified level of the deviance for the proposed split using the "misclassification" criterion, which is compared to the calculated deviance of the split. If the deviance for a given split (from a parent to two terminal nodes) fails to exceed the specified level, then that split in the tree is pruned back. Error estimates for each of the specified levels of deviance are shown, revealing a slightly curved pattern, which indicates that the "best" tree can be found in the middle ranges of the specified levels that were tested.

B	Avg. number pruned	Std. deviation of 40 "number pruned" values	Mean absolute error	Std. deviation of 40 "absolute error" values	Std. Error of MAE
0	5.2	3.608	0.106	0.030	0.005
1	18.5	7.394	0.077	0.027	0.004
2	19.4	7.445	0.073	0.027	0.004
3	20.0	7.512	0.069	0.027	0.004
4	20.9	7.842	0.066	0.025	0.004
5	20.9	7.842	0.066	0.025	0.004
6	21.1	7.826	0.066	0.025	0.004
7	21.2	7.883	0.066	0.025	0.004
8	21.2	7.915	0.066	0.025	0.004
9	21.3	7.865	0.067	0.025	0.004
10	21.4	7.830	0.068	0.026	0.004
11	21.4	7.782	0.069	0.026	0.004
12	21.4	7.736	0.071	0.026	0.004
13	21.4	7.703	0.071	0.026	0.004
14	21.5	7.653	0.072	0.029	0.005
15	21.6	7.608	0.074	0.031	0.005
16	21.8	7.530	0.075	0.034	0.005
17	22.0	7.651	0.073	0.034	0.005
18	22.0	7.636	0.074	0.035	0.006
19	22.0	7.636	0.074	0.035	0.006
20	22.1	7.459	0.072	0.032	0.005
25	22.2	7.445	0.072	0.031	0.005
30	22.4	7.396	0.078	0.051	0.008
35	22.4	7.383	0.080	0.055	0.009
40	22.4	7.404	0.084	0.059	0.009

CHAPTER 5. SUMMARY

Classification and regression trees have proven their worth as an effective statistical modeling tool for many situations. While in some cases, CART may be lacking the ability to create accurate models, an advantage of CART is that it is flexible enough to adapt to new situations (with some help from the statistician, of course). We have examined two such situations: (1) Multiple surveys with imperfect detection and (2) Correlated binary data.

In the former situation, we considered studies in which the researcher is interested in determining the presence of an individual (or characteristic). To do so, the researcher takes multiple observations of whether that individual/characteristic is seen or not seen. The underlying issue of the observations is whether or not non-detection implies non-presence. Being able to account for detection probability in the pursuit of predicting presence/absence of a desired individual (or characteristic, etc.) allows CART to be more accurate in its predictions. We proposed four methods designed to improve model performance by incorporating imperfect detection into the model. The p.v.d.v.d. method called for separate detection and occupancy parameters (π_d and π_{occ}) at each node involved in a split. The orig.parent, each.parent, and parent.v.2daughters methods each worked on the assumption of a single detection probability during the split, and with separate occupancy probabilities at each node. The difference between the latter three methods is how the detection parameter is estimated in the model. The orig.parent method specifies that detection is constant throughout the entire tree (detection probability is then estimated from the root node of the tree). The each.parent and p.v.2d methods keep the detection probability constant only within each split, but each.parent specifies that the detection parameter value is estimated from the parent node, while it is jointly estimated from the two daughter nodes in p.v.2d. Results for the plover data strongly indicate that the orig.parent model performed the best for this example (but may not always be the best). From our dis-

cussions in Chapter 2, we expect that most methods incorporating imperfect detection will outperform a naïve model assuming perfect detection. Of the four alternative methods proposed here, p.v.d.v.d. has noted shortcomings; of the remaining three, the best-performing method may vary depending on the data being used. The methods that we presented enable CART to be extended to statistical analyses which would otherwise use a different modeling tool.

In the latter situation, knowledge of underlying correlation structure between data points, when used in conjunction with CART methodology, serves to enhance the splitting algorithms. For example, if CART has information suggesting that two (or more) data points are strongly correlated, then a split of the current parent node node under consideration may place a greater emphasis on what happens to those two (or more) points (i.e. if they stay together or become separated between the parent node and the daughter nodes). The naïve approach to clustered data would be to treat the observations as if they were independent. Accounting for clustering is shown to [potentially] cause changes to the splitting decisions, and even to the variable and split point used in each branch of the tree. These changes in turn could lead to changes in prediction or classification of the individuals in the model, which makes clustering an important, yet often overlooked, contributor to the CART model. More specifically, clustered data can affect the variance of parameter estimates, but unfortunately this effect (with respect to the naïve case assuming independence) is neither constant nor has a consistent direction (increase or decrease). This requires methodology that is adaptable to any possible situation involving correlated data in a CART tree. The drop in deviance of the potential splits of nodes in CART was based on the use of a Wald test statistic. Since clustering can affect the variance of the parameters, the test statistic was calculated using a generalized estimating equations-based approach, which attempts to "correct" the naïve test statistic. We modified the variance of the point estimate $(\pi_L - \pi_R)$ based on the amount of correlation present in the data set, the relationship of correlated data points to each other in the tree, and whether the variances of observations in the two daughter nodes were considered to be equal or allowed to be unequal. The improvement of the clustering technique over the naïve application of CART to a set of "independent" data was demonstrated through a simulation, the results of which were displayed

in Table 3.5. In most situations, accounting for clustering led to empirical Type I error rates for tests closer to nominal rates.

After creating a CART tree, a well-known practice is for the researcher to further prune the tree. This pruning uses the same methodology and calculations that were used to grow the tree. Since we have introduced new methodology for the growth process, we have also examined the need for corresponding changes when pruning. Current methods of pruning in *rpart()* include (but are not limited to) a node size criterion, a misclassification measure, and a complexity measure. We desired to implement a new pruning algorithm based on the new methodologies introduced in Chapters 2 and 3.

We considered five potential measures to decide whether to prune a particular branch:

1. The parameter of interest, occupancy probability
2. The intertwined parameter, detection probability
3. The test statistic (Equation 3.2)
4. The misclassification measure of node impurity
5. The node size criterion

Simulations were run to assess the performance of each pruning criterion separately. The exception to this was the node size criterion, which was applied in all simulations (the node size criterion was inadvertently ignored when the growth process was forced to completion without internal pruning). The results indicated that pruning based on the test statistic did a good job at reducing the error rate. Pruning based on misclassification is slightly erratic and untrustworthy (this statistic may be biased when used under conditions of imperfect detection). The simulations involving occupancy and detection probabilities were inconclusive. Please refer to Chapter 4 for more details on the simulations.

In conclusion, the methodologies presented in this paper have direct application to many studies in which researchers would otherwise be likely to choose an analysis tool other than CART, but where they might now reconsider the application of CART. Given CART's various strong features (e.g. Deal with missing values, allows for interaction, excellent visual display results), in addition to the extra methodologies we have presented, could provide the added

incentive to use CART. Although there are many other current and future extensions to CART, incorporating imperfect detection and correlated data into the CART process provides useful extensions to the library of possibilities for classification and regression trees, by providing opportunities in studies involving imperfect detection or correlated data..