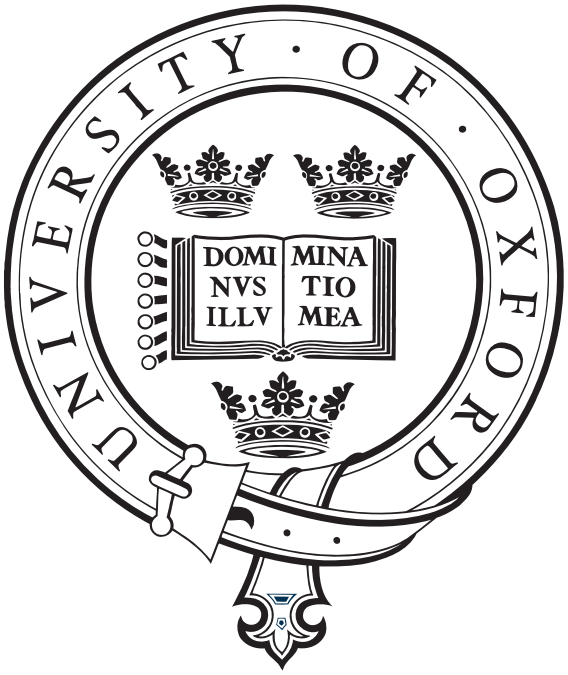
Fast Growing and Interpretable Oblique Trees via Logistic Regression Models

Alfred Kar Yin Truong

Christ Church College

University of Oxford



A Thesis submitted for the degree of Doctor of Philosophy

Trinity Term 2009

Acknowledgments

There are many people I would like to thank who have contributed to my completing this thesis.

My Supervisor:

Brian, you took me as your student and gave me the freedom to pursue my interests. I have grown both as a researcher and as a person in these past 4 years for which I cannot thank you enough.

My Friends:

I would like to thank my friends from both Oxford and Cambridge for the fun times that we shared and for the encouragement that you have given me along the way.

The UK:

The UK has given me the opportunity to pursue my studies at two great universities with some very smart people for which I am very grateful. In addition to this, my doctoral studies have been generously funded by EPSRC without which I could not have written this thesis.

My Family:

And finally I would like to take this opportunity to show my appreciation to my family.

阿妈: I still remember the time you taught me basic arithmetic and times-tables! The most important lesson you taught me however was to persevere and so I have done.

阿爸: You have been kind and honourable throughout your life for which I am proud. I have followed in your footsteps and will continue to do so.

阿哥: You are the best brother anyone could ask for. You are strong, kind and caring and have always been a role model for me.

贝贝: You were always supportive and encouraging and were always at my side. I grew up with you.

I dedicate this thesis to you all.

Fast Growing and Interpretable Oblique Trees via Logistic Regression Models

Alfred Kar Yin Truong, Christ Church College

DPhil thesis, Department of Statistics

Trinity Term 2009

The classification tree is an attractive method for classification as the predictions it makes are more transparent than most other classifiers. The most widely accepted approaches to tree-growth use axis-parallel splits to partition continuous attributes. Since the interpretability of a tree diminishes as it grows larger, researchers have sought ways of growing trees with oblique splits as they are better able to partition observations. The focus of this thesis is to grow oblique trees in a fast and deterministic manner and to propose ways of making them more interpretable.

Finding good oblique splits is a computationally difficult task. Various authors have proposed ways of doing this by either performing stochastic searches or by solving problems that effectively produce oblique splits at each stage of tree-growth. A new approach to finding such splits is proposed that restricts attention to a small but comprehensive set of splits. Empirical evidence shows that good oblique splits are found in most cases. When observations come from a small number of classes, empirical evidence shows that oblique trees can be grown in a matter of seconds.

As interpretability is the main strength of classification trees, it is important for oblique trees that are grown to be interpretable. As the proposed approach to finding oblique splits makes use of logistic regression, well-founded variable selection techniques are introduced to classification trees. This allows concise oblique splits to be found at each stage of tree-growth so that oblique trees that are more interpretable can be directly grown. In addition to this, cost-complexity pruning ideas which were developed for axis-parallel trees have been adapted to make oblique trees more interpretable.

A major and practical component of this thesis is in providing the *oblique.tree* package in R that allows casual users to experiment with oblique trees in a way that was not possible before.

Declaration

I declare that this thesis is wholly my own work unless otherwise stated. No part of this thesis has been accepted or is currently being submitted for any degree or diploma or certificate or other qualification in this university or elsewhere.

Candidate: Alfred Kar Yin TRUONG

Signed:

Date:

Contents

Chapter 1

Introduction

1.1 Recent Trends in Statistics

The emergence of modern computers has transformed the field of statistics in many ways. More questions are posed by the ever larger and more complex datasets that have since been amassed as a result of advances in automated data collection and cheaper data-storage. Datasets are both larger and more complex in that they have,

Many cases: This is perhaps the most common situation.

- Oxford's library catalog has 5 million items.
- Census data ($\approx$ 60m individuals in the UK, $\approx$ 300m in the US).

Many pieces of information per case:

- Genome-wide screens collect potentially tens of thousands of pieces of information from a small number of subjects (perhaps only 100).
- fMRI[1] brain scans produce gigabytes of data from each individual.

[1] functional Magnetic Resonance Imaging.

Many cases with many pieces of information per case: Though less common, examples do exist. Some consortiums have CRM[2] databases that gather information from consumers about their shopping habits (with thousands of products from millions of customers) which they use to target advertisements.

Classification datasets in particular can also become more complex in a further dimension; observations can come from a larger number of classes.

Many classes: Some OCR[3] problems require data to be gathered from instances of characters from some alphabet upon which a classifier is trained. There are 10 classes for digits, 26 for upper (or lower) case characters from the Latin alphabet and some 3,000 for Chinese characters.

Most classification problems have observations that come from a small number of classes, e.g. the presence or lack thereof of a disease, the response to a questionnaire with "yes", "no" or "maybe" answers and so on. Figure 1.1 shows a histogram of the number of classes in classification problems gathered from the UCI Machine Learning Repository [Asuncion and Newman, 2007]. One sees that a large proportion of problems have observations that come from 2-5 classes (most have less than 10).

Other than allowing more interesting questions to be posed, modern computers have also become an indispensable tool for statisticians. The computational power of computers has increased to the point where calculations that would have taken years by hand can now be performed at the blink of an eye by a computer.

Though much progress has been made in fields that range from bioinformatics to OCR, it is often difficult for users to apply cutting-edge statistical methods to real-

[2]Client Relationship Management.
[3]Optical Character Recognition.

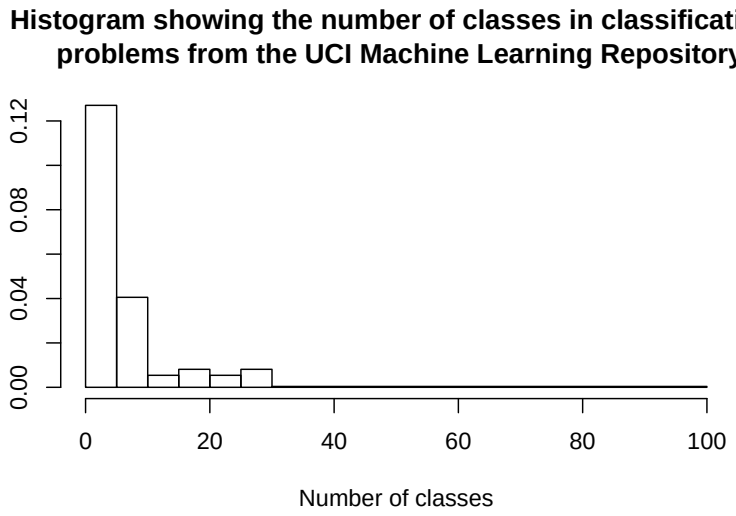


Figure 1.1: Classification datasets often have observations that come from a small number of classes. The above histogram confirms this for classification problems gathered in the UCI Machine Learning Repository [Asuncion and Newman, 2007].

world datasets (academic work is often poorly packaged and proprietary software too expensive for casual users). There has however been movement in recent years towards providing widely accessible open-source software[4] to address these issues. With vast improvements in computational power and a growing collection of easily extensible statistical software it is worthwhile revisiting problems that were once considered too difficult.

1.2 Aim of this Thesis

The *classification tree* is a classifier whose main attraction is the simplicity by which it makes predictions. A widely used form of this idea is described in Breiman et al.'s 1984 book. It produces binary partitioning trees by considering *axis-parallel splits*

[4]R [R Development Core Team, 2009] is a free software environment for statistical computing that gathers open-source implementations of an extensive collection of statistical and graphical techniques. It has become a defacto standard for developers of statistical software due to its highly extensible nature and ease of use.

over continuous attributes (tests of the form $X_i < c$ v.s. $X_i \geq c$). We refer to such trees as *CART*.

Though CART is generally considered to be interpretable, their interpretability diminishes as they become *larger* (uses more tests). Researchers have sought ways of growing[5] trees with tests of the form $\sum_{i=1}^{q} a_i X_i < c$ v.s. $\sum_{i=1}^{q} a_i X_i \geq c$ over continuous attributes (assuming continuous attributes lie in the first q positions of feature vectors) in hope of producing more interpretable trees. We call these tests *oblique splits*[6]. Existing research allows *oblique trees* (trees grown by considering oblique splits over continuous attributes) to be grown though such trees are not always interpretable. Unfortunately none of these attempts have caught on; *axis-parallel trees* (those that consider axis-parallel splits) are still widely used.

This thesis aims to identify and address possible reasons for this so that oblique trees can be better made use of. After reviewing existing work, a directed approach to finding oblique splits with low values of impurity is proposed that restricts attention to a small but comprehensive set of splits. They are found by solving a family of two-class classification problems at each node. As these problems are based on partitioning observations of different classes from each other such splits should automatically have low values of impurity. By solving these problems with logistic regression we effectively introduce well-founded variable selection techniques to classification trees. This link is then exploited so that more interpretable oblique trees can be found by using oblique splits that depend on a smaller number of continuous attributes.

[5]Trees are said to be *grown* as well as trained to data.

[6]Oblique splits have been known by many names as different researchers have tried to incorporate them into tree-growth. Other names include Linear-Combination Tests, Multivariate Tests, Perceptrons and Linear Machines.

1.3 Organization of this Thesis

This thesis is organized as follows. Chapter 1 gives an overview of recent developments in statistics and follows with a description of the aim of this thesis.

Chapter 2 attempts to identify possible reasons as to why existing approaches to growing oblique trees have not caught on. Motivational ideas behind how classification trees are grown are covered and a discussion about why using oblique splits might produce more interpretable trees follows.

Chapter 3 surveys existing approaches to growing oblique trees. Existing approaches are categorized into two types as specified by the way in which they find oblique splits and are presented in the same manner. Doing so provides a clearer overview of its progression and allows for a better understanding of the pros and cons of each approach.

Chapter 4 presents a new approach to finding oblique splits as well as to growing oblique trees. This approach is demonstrated on various datasets to gauge how quickly it grows oblique trees. A heuristic argument that explains why such an approach works follows.

Building on Chapter 4, Chapter 5 proposes ways of producing more interpretable oblique trees. Approaches come in two forms, those applied during tree-growth and those applied post-tree-growth. Well-founded variable selection ideas are used in each case.

Having proposed a new approach to growing oblique trees as well as various ways of making them more interpretable, Chapter 6 analyses some of the finer details of its implementation. Situations where such an approach might fail to perform well are considered as well as the computational limits of such an approach.

Chapter 7 concludes with final remarks.

Chapter 2

Background

This chapter covers motivational ideas behind tree-structured classifiers and reviews issues involved when training CART to data. By covering all aspects of tree-growth, likely reasons why existing approaches to growing oblique trees have not caught on are identified. Section 2.1 revisits the basic idea of a classification tree and Section 2.2 follows with a discussion of issues that affect its interpretability. A list of plausible reasons why existing approaches to growing oblique trees have not caught on are given in Section 2.3.

2.1 Classification Trees

2.1.1 Natural Origins

Tree-structured classifiers appear naturally when considering populations with hierarchical structure. Biologists and botanists use "taxonomic keys" to identify organisms and doctors use "decision trees" or "flow-charts" to assist with medical diagnoses: they all amount to the same idea. Figure 2.1(a) shows an example of a taxonomic key in botany. Armed with such a key anybody could easily follow it to

1a. Plants with shoots ..... Go to 2
1b. Plants without shoots ..... Moss

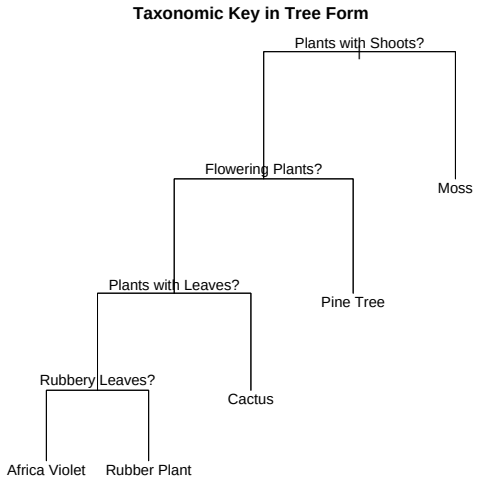
2a. Flowering Plants ..... Go to 3
2b. Non-Flowering Plants ..... Pine Tree

3a. Plants with leaves ..... Go to 4
3b. Plants without leaves ..... Cactus

4a. Rubbery leaves ..... Rubber Plant
4b. Fuzzy Leaves ..... Africa Violet

Generated by General Biology I students: Amani Salem, Vinny Motiram, Suraiya Roy, and Nelum Shahzadi at Kingsborough Community College.

(a) Taxonomic key for various plant species.



(b) Graphical representation of taxonomic key.

Figure 2.1: Tree-structured classifiers are a natural way of explaining hierarchical populations and have long been used by practitioners from various fields. A dichotomous taxonomic key is a realization of this idea that poses a series of questions with *Yes* or *No* answers as shown in Figure 2.1(a). It is easy to follow with each path leading to a prediction. This information can also be represented graphically as seen in Figure 2.1(b) where a left branch by convention denotes a positive answer.

try identifying the species of a plant, e.g. if a plant has shoots but does not flower we guess it is a pine tree, if it shoots and flowers but does not have leaves we guess it is a cactus tree. This information can also be represented graphically as shown in Figure 2.1(b).

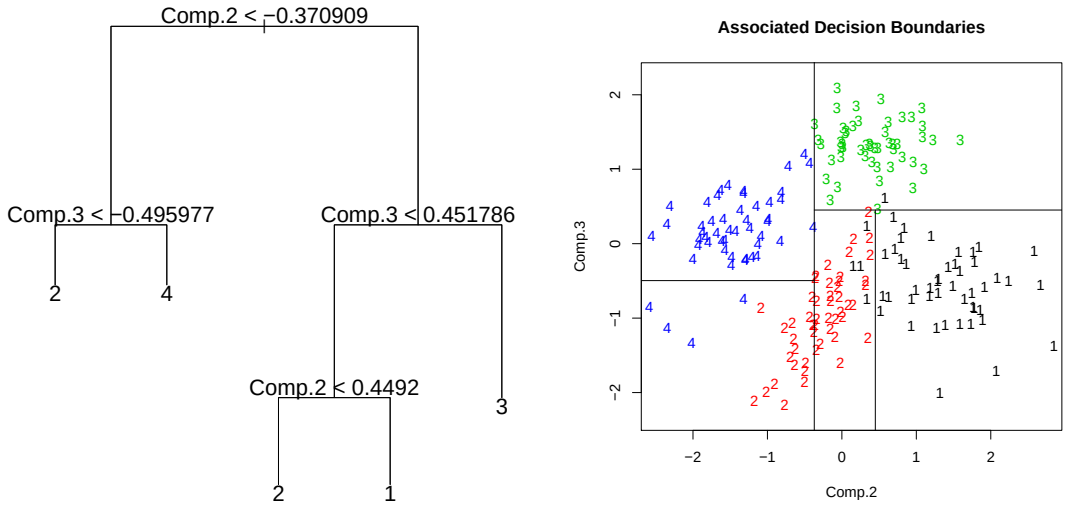
Though a great deal of thought is often needed to create such classifiers, researchers have sought ways of automating this process. According to Breiman et al. [1984, pg. 6–7],

> "an important criterion for a good classification procedure is that it not only produce accurate classifiers ... but that it also provide insight and understanding into the predictive structure of the data."

These goals are not mutually exclusive and it is anecdotally known that they are often of equal importance to practitioners. Work by researchers from computer science [Quinlan, 1979, 1986, 1988, 1990], engineering [Henrichon and Fu, 1969, Sethi and Sarvarayudu, 1982] and statistics [Morgan and Messenger, 1973, Kass, 1980] resulted in similar approaches to tackling this problem. With greater maturity and cross-fertilization of ideas, Breiman et al.'s [1984] CART has come to be widely accepted. An overview of these ideas follow.

2.1.2 Tree-Growth

Given training data of the form $\mathcal{T} = \left(C_i, X_1^i, \ldots, X_p^i\right)_{i=1}^N = (C_i, \mathbf{X}^i)_{i=1}^N$ where $C_i \in \mathcal{C} = \{C_1, \ldots, C_k\}$ denotes the class of each observation, a classification tree is grown to $\mathcal{T}$ by recursively partitioning observations into smaller subsets. This process is naturally representable as a tree with



(a) Example of a binary partitioning axis-parallel tree.

(b) The associated decision boundaries that are defined by the tree on the feature space.

Figure 2.2: The above tree is grown to a training set where observations have measurements from two continuous attributes ($p = q = 2$) and where observations come from four classes ($k = 4$). Four binary partitioning tests are applied to produce five leaves. As $q = 2$, it is possible to view the decision boundaries defined by this tree. The feature space is partitioned into rectangular blocks as axis-parallel splits are used.

- internal nodes where tests[1] partition observations,

- branches from nodes which represent the outcomes of these tests and

- leaves at the bottom[2] of a tree.

When predicting with classification trees, observations are passed through a tree and classified by the class associated to the leaf where it falls. Figure 2.2 gives an example of a typical tree. Four tests partition training set observations and good separation is achieved between observations of each class.

[1]Tests partition training set observations into smaller subsets and so are also called *splits*. We use these terms interchangeably.

[2]Trees are conventionally drawn with its root at the top of a page growing downwards.

Though the idea of a classification tree allows arbitrarily complex tests to be applied, CART restricts attention to the following types of binary partitioning tests,

X_i continuous: $X_i < c$ v.s. $X_i \geq c$ for some constant $c \in \mathbb{R}$

X_i categorical: $X_i \in A$ v.s. $X_i \notin A$ for some subset $A \subset \{a_1, \ldots, a_{|X_i|}\}$ of the factors of X_i

Seeking to "produce purer child nodes" at each stage of tree-growth, CART applies the test that "best partitions observations of each class" [Breiman et al., 1984, pg. 23–24]. This is done by use of an *impurity measure* that somehow quantifies this idea (the role of an impurity measure in tree-growth is touched upon here, an in-depth discussion follows in Section 4.1).

Definition 2.1.1 (Best Split)**.** *Given an impurity measure $j()$ over a set of splits, the* best split *is the one that results in the lowest value of $j()$ over that set of splits (where ties are broken randomly).*

CART applies the best split at each stage of tree-growth by comparing $j()$ over all splits permitted at a node.

Having understood tree-growth, we move on to consider two techniques that CART employs to avoid overfitting[3]. The simpler of the two works by forcing recursive partitioning to terminate before every training set observation becomes perfectly classified. The logic is straightforward,

1. Were recursive partitioning to proceed until all training set observations are perfectly classified, it is likely the tree would have focused more on explain-

[3]Highly parameterized models are able to fit training data perfectly but often fail to predict well to previously unobserved observations, that is to say, overfit classifiers are undesirable. Hastie et al. [2001, Chapter 7] and Geurts [2002] give good accounts of this phenomenon better known as the bias-variance tradeoff.

ing idiosyncrasies of the training set rather than the general trends in the population.

2. Recursive partitioning at a node should therefore stop before all observations are of the same class.

To achieve this, a node is no longer partitioned if any of the following *stopping criteria* are satisfied at this node,

Pure Nodes: all observations are of the same class.

Size Threshold: there are fewer than some prespecified number of observations, say 10.

Impurity Threshold: though it differs among implementations, an example would be if the impurity at a node is below some prespecified proportion of that of the root node, say 1%.

Child Node Threshold: each split produces a child node with fewer than some prespecified number of observations, say half the size threshold.

Unfortunately these stopping criteria were found to be insufficient in ensuring trees grown will generalize[4]. Breiman et al. proposed a complementary technique.

2.1.3 Cost-Complexity Pruning

Growing trees that generalize can be thought of as tackling an optimal stopping problem for tree-growth; trees should be grown to data to learn from it but they should not be overgrown. When exactly should tree-growth stop?

[4]A classifier generalizes if that is not overfit, i.e. it makes good predictions to previously unobserved observations.

Cost-complexity pruning sidesteps this problem by simply growing a *maximal tree* (a tree that is grown until the stopping criteria is satisfied at each leaf). Rooted subtrees of this tree are then considered to find trees that do generalize. Rather than considering all possible rooted subtrees[5], cost-complexity pruning focuses on a particular subset of them found as follows. Let

$R(T)$ denote a measure of the goodness of fit of T over training set observations found by adding contributions over its leaves t (the number of misclassified observations say or their node impurities) and

$size(T)$ denote a measure of the complexity of T (the number of leaves say).

Consider the following measure for comparing rooted subtrees of a maximal tree

$$R_k(T) = R(T) + k\ size(T).$$

For each $k \in \mathbb{R}$, $R_k(T)$ weights the ability of rooted subtrees to predict training set observations well against its complexity cost to the tree as measured by $size(T)$.

Those rooted subtrees that are further considered are those that minimize $R_k(T)$ as k varies over $\mathbb{R}$. Breiman et al. [1984, pg. 284–293] show[6] that such a *tree sequence* $\{T_i\}_{i=0}^{K}$ is nested and that each T_i is optimal over the range $k \in [k_i, k_{i+1})$ where

$$-\infty = k_0 < k_1 < \ldots < \infty.$$

[5]The number of rooted subtrees of a maximal tree is potentially vast. For binary trees with l leaves, Breiman et al. [1984, pg. 284] state an upper bound of $\lfloor 1.5028369^l \rfloor$ with equality in the case of binary saturated trees (binary trees with n levels of internal nodes and a full complement of 2^n leaves). There are $26 = \lfloor (1.5028369)^{2^3} \rfloor$ rooted subtrees of a 3-leveled saturated tree and $677 = \lfloor (1.5028369)^{2^4} \rfloor$ for a 4-leveled one. To see how this growth in the number of rooted subtrees arises, all rooted subtrees of a 3-leveled saturated tree are shown in Figure 2.3.

[6]Concise version of these proofs are found in Ripley [1996, pg. 221–224].

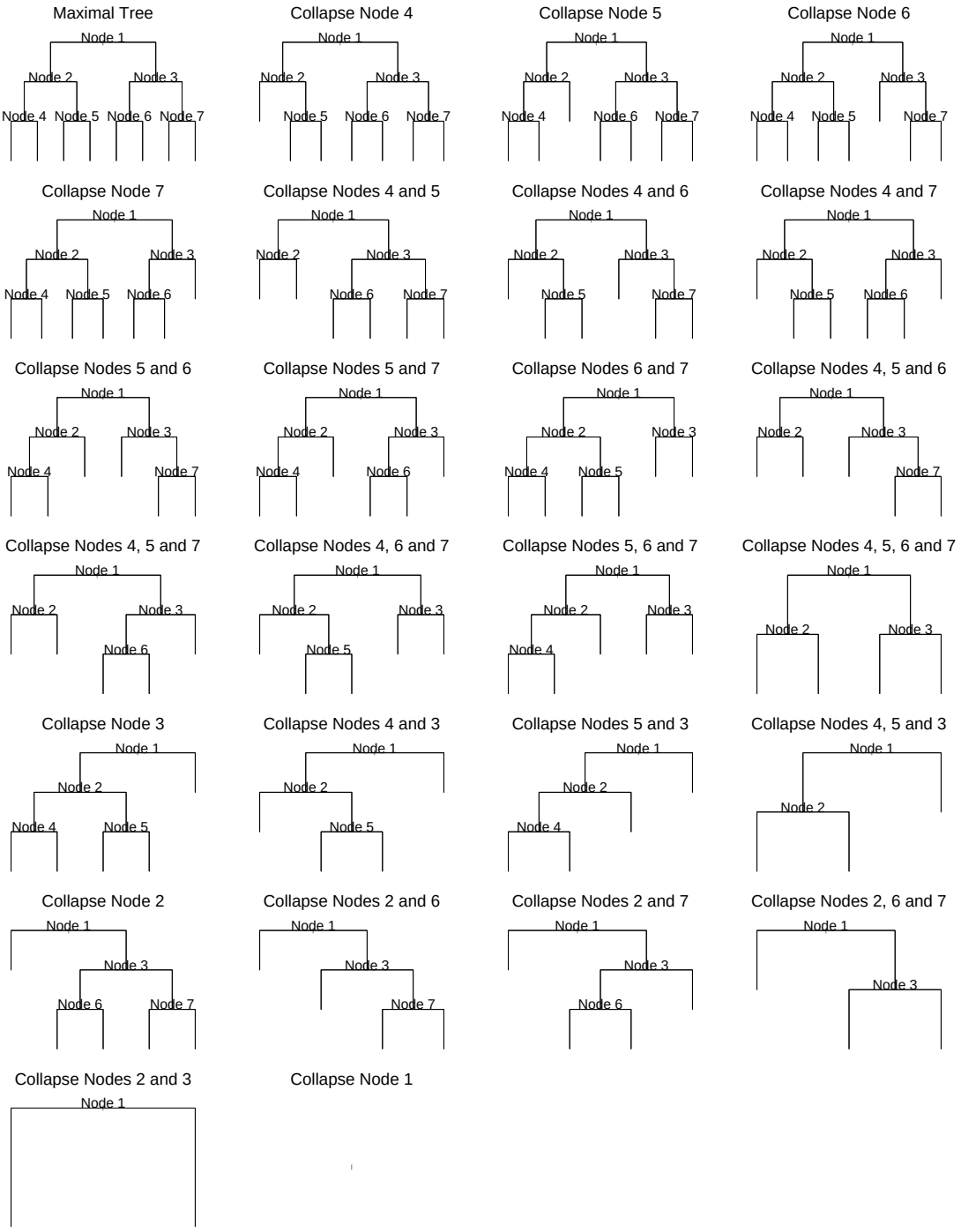


Figure 2.3: A 3-leveled saturated tree is shown in the top-left subplot. The other subplots show rooted subtrees of it. Rows 1, 2, 3 and 4 shows those where neither nodes 2 and 3 are absent, row 5 shows those where node 3 is absent, row 6 shows those where node 2 is absent and the rest follow on the last row. It is easy to see the multiplicative effect each additional internal node has on the total number of rooted subtrees.

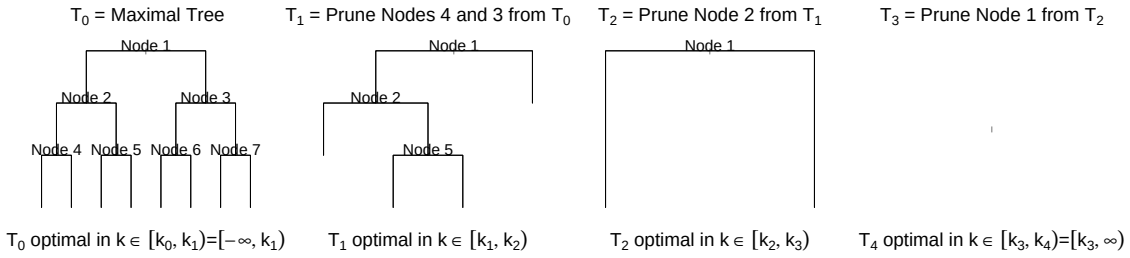


Figure 2.4: Cost-complexity pruning allows trees that better generalize to be found. Starting with a maximal tree, a tree sequence $\{T_i\}_{i=0}^{K}$ is extracted from it and are used as candidates for the final classifier. Though *training error* (prediction error on the training set) increases with i, *test error* (prediction error on previously unobserved data) needs not; the tree with the lowest test error is often used as the final classifier, T_1 in this example say.

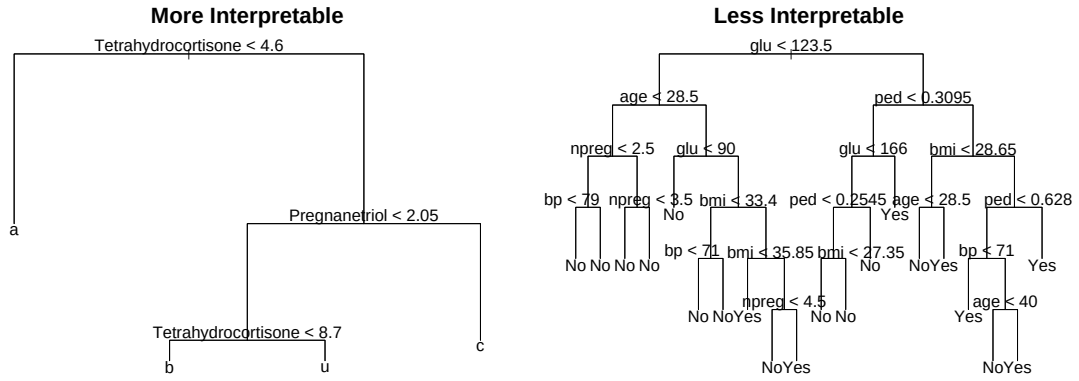
They also present an algorithm that finds this tree sequence. The rooted subtree that best generalizes (with lowest test error) is easy to identify and is often used as the final classifier as demonstrated in Figure 2.4.

2.2 Towards More Interpretable Trees

The reason why researchers have tried to grow trees with oblique splits is to produce more interpretable trees. Unfortunately oblique splits are difficult to implement and there are unintended side-effects of their use which we now consider.

2.2.1 Smaller Trees with Oblique Splits

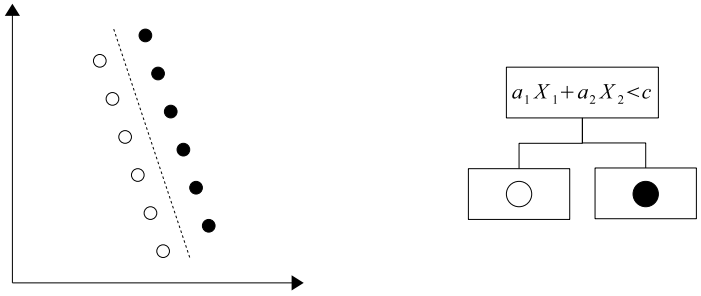
Some trees are larger than others simply because of the interplay between how observations of different classes are distributed over the feature space and the tests used to partition them. Smaller trees are more interpretable as seen in Figure 2.5. As oblique splits are better able to partition observations (Figure 2.6) oblique trees naturally use fewer tests, leads one to believe that they are also more interpretable which unfortunately is not always the case.



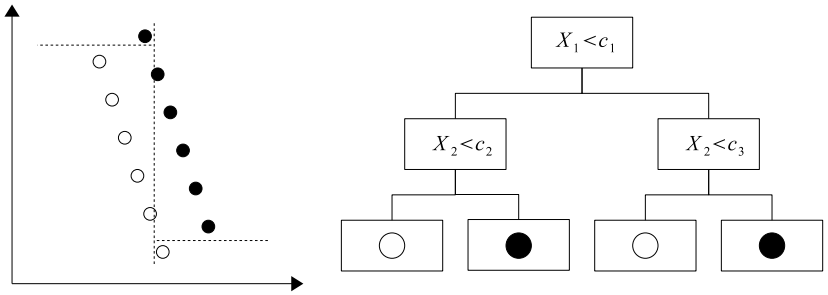
(a) Axis-parallel tree grown on the Cushings dataset [Aitchison and Dunsmore, 1975].

(b) Axis-parallel tree grown on the Pima Indians dataset [Smith et al., 1988].

Figure 2.5: The interpretability of a tree depends on the interplay between two factors, the interpretability of each split in isolation and its cumulative effect over the entire tree. Though axis-parallel splits are interpretable in isolation, the interpretability of entire axis-parallel trees is another matter altogether.



(a) A single oblique split is able to perfectly partition these observations.



(b) An axis-parallel tree however requires more than one level of splits to achieve the same partitioning of observations.

Figure 2.6: Observations from this artificial two-class classification problem are easily partitioned with a single oblique split, axis-parallel splits however cannot produce the same partition with a single split, i.e. oblique splits are better able to partition observations so oblique trees are naturally smaller.

Definition 2.2.1 (Full and Concise Oblique Splits)**.** *Where there are q continuous attributes in a dataset, an oblique split based on*

- *all q attributes is called a* full oblique split*,*

- *fewer than q attributes is called a* concise oblique split*.*

Concise oblique splits are more interpretable than full oblique splits, e.g. $X_1 > 0.25X_2$ v.s. $X_1 > 0.25X_2 + 0.5X_3 + X_4$. Though full oblique splits are better able to partition observations, *full oblique trees* (trees grown with full oblique splits) can paradoxically be less interpretable than *concise oblique trees* (those grown with concise oblique splits); too much information is compressed in each split (the problem becoming more acute with larger q). To obtain more interpretable oblique trees, concise oblique splits should be used "whenever possible". With so much subjectivity, obtaining more interpretable oblique trees is a non-trivial task.

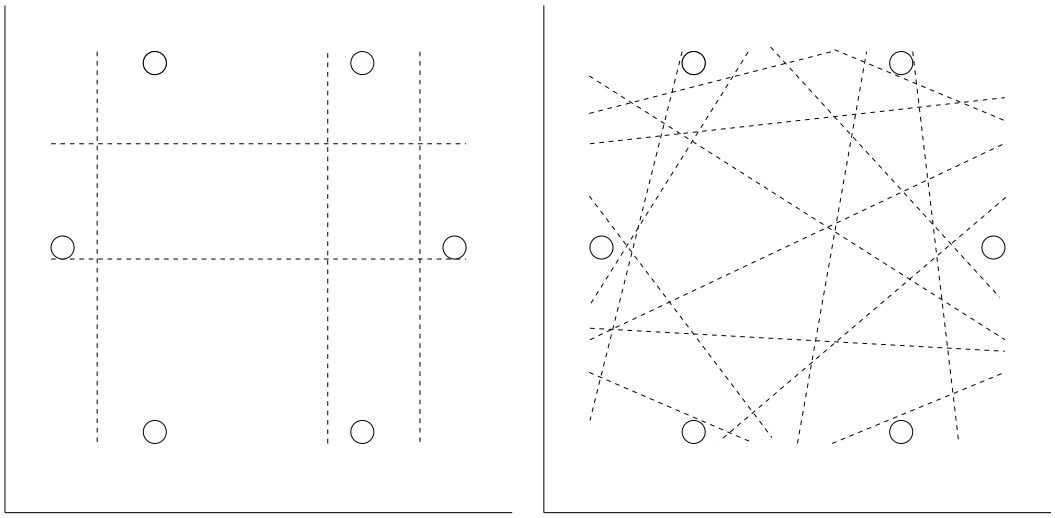
2.2.2 Finding Oblique Splits

Putting aside the issue of how one finds concise oblique splits for the moment, consider the simpler problem of using full oblique splits.

One ideally applies the best oblique split at each stage of tree-growth which implies that *every* oblique split needs to be considered. How difficult a problem is this in comparison to that of finding the best axis-parallel split?

Definition 2.2.2 (Outcome of a Test)**.** *The* outcome of a test *is the allocation of observations to child nodes that results when such a test is applied at a node.*

Definition 2.2.3 (Split Dictionary)**.** *A* split dictionary *(for a family of splits) is a set of splits of minimal size that produces all possible outcomes (permitted by that family of splits) over the training set.*



(a) Example of an axis-parallel split dictionary.

(b) Example of an oblique split dictionary.

Figure 2.7: Though split dictionaries are not necessarily unique (perturbing a split in a split dictionary might produce new split dictionaries) its size remains constant over a given dataset. Figure 2.7(a) shows a possible axis-parallel split dictionary for a small 2-dimensional dataset and is of size 6. No outcomes other than those generated by these splits can result when axis-parallel splits are applied to $\mathcal{T}$. Figure 2.7(b) shows a possible oblique split dictionary for the same data and is of size 15 (it is larger as oblique splits are better able to partition observations).

Figure 2.7 illustrates these ideas by showing examples of axis-parallel and oblique split dictionaries over an artificial dataset. It is easy to see that many split dictionaries may exist for a given family of splits on any dataset. Though this may be the case, their size remain constant. With this in mind we can directly compare the difficulty of finding the best axis-parallel split with that of finding the best oblique split by comparing the size of each split dictionary over a generic training set $\mathcal{T}$ (with N observations and q continuous attributes) as we do now.

Starting with axis-parallel splits, though there are uncountably many splits over each X_i ($X_i < c$ with $c \in \mathbb{R}$), only a finite number of outcomes ever occur over $\mathcal{T}$. With L_i unique values of X_i, there are exactly $L_i - 1$ unique outcomes. The

size of the axis-parallel split dictionary over all attributes is therefore $\sum_{i=1}^{q}(L_i - 1)$. As $L_i \leq N$ the axis-parallel split dictionary grows at worst linearly in both N and q.

Though it is difficult to directly compute the size of oblique split dictionaries, its size can be shown to be exactly $\left\{2\sum_{k=0}^{q}\binom{L_{gen}-1}{k} - 2\right\}/2$ where L_{gen} is the number of generally positioned[7] q-dimensional points in $\mathcal{T}$ and is a consequence of a result[8] from combinatorial geometry. Assuming $L_{gen} \approx N$ (it will only be significantly less in the unlikely event that all q-dimensional points lie in a space of much lower dimensionality) shows that oblique split dictionaries grow extremely quickly in N and q.

As noted in Section 1.1 N and q are often large in real-life datasets. It is therefore not feasible in practice to find the best oblique split at each stage of tree-growth (unlike for axis-parallel splits) and so researchers do not try.

Definition 2.2.4 (Good Split)**.** *For a given subset of splits from a split dictionary,* good splits *are those that have low values of impurity relative to other splits in this subset.*

Rather than finding the best oblique split at each stage of tree-growth, good oblique splits are instead sought. This avoids the computational explosion in the number of splits that need to be considered. As will be seen in Chapter 3, this is done by either

[7] A set of q-dimensional vectors are generally positioned if every subset of size q or less is linearly independent.

[8] Given L_{gen} generally positioned points in q-dimensional space, there are exactly $2\sum_{k=0}^{q-1}\binom{L_{gen}-1}{k}$ ways of partitioning them into two bins as specified by which side of a hyperplane $\{x : w \cdot x = 0\}$ they fall. Cover [1965] extended this result to enumerate the number of ways there are to partition points across hyperplanes of the form $\{x : w \cdot f(x) = 0\}$ for more general functions f. Choosing $f(x) = (1, x)$ in particular shows there are exactly $2\sum_{k=0}^{q}\binom{L_{gen}-1}{k}$ ways of binning points across hyperplanes with arbitrary intercept, it includes the trivial cases where all points are assigned to the same bin while also double counting across bins.

- searching for the best oblique split with the intention of stopping early or

- restricting attention to a subset of the oblique split dictionary.

2.3 Low Uptake of Oblique Trees

In light of the issues mentioned in this chapter, plausible reasons why existing approaches to growing oblique trees have not caught on are identified.

Intellectual Appeal of the Approach: In order to avoid searching over the entire oblique split dictionary, existing approaches find good oblique splits in haphazard ways. This does not give confidence to users in the suitability of such classifiers.

Multiplicity of Trees: Some approaches for finding good oblique splits perform stochastic searches and so produce a different tree at each attempt. Though not necessarily a problem, no well-defined approach is given as to how one obtains a "final classifier" from the countless that can be grown. This again does not foster confidence.

Interpretability of Trees: Most researchers are content with simply growing full oblique trees and so the trees that are grown are not always interpretable, i.e. there is no advantage in using them over other classifiers.

Easily Accessible Implementations: It is difficult to obtain and apply implementations of these approaches to data, perhaps the greatest deterrent of potential users.

This thesis seeks to address these issues by proposing an intellectually appealing approach to growing more interpretable oblique trees while providing an easily

accessible implementation for others to use.

Chapter 3

Existing Approaches to Growing Oblique Trees

This chapter reviews existing approaches to growing oblique trees. As it is not feasible to find the best oblique split at each stage of tree-growth, good oblique splits are instead sought and done so with a view to finding them as quickly as possible. Existing work can be categorized into two types,

Direct Search Approaches: search for good oblique splits *over the entire oblique split dictionary* with the intention of stopping the search early.

Indirect Search Approaches: search for good oblique splits *over some prespecified subset of the split dictionary.*

This chapter is organized as follows. Existing work from each category is presented separately and chronologically with direct searches in Section 3.1 and indirect searches in Section 3.2. This provides a clear overview of their progression and highlights inherent strengths and weaknesses of each approach as summarized in Section 3.3.

3.1 Direct Search Approaches

The approach taken by researchers who propose direct searches can be summarized as follows,

1. a full oblique split is simply a q-dimensional hyperplane $\sum_{i=1}^{q} a_i X_i = c$,
2. this can be thought of in terms of its coefficients $(a_1, \ldots, a_q, c) \in \mathbb{R}^{q+1}$,
3. by defining a goodness (badness) function for oblique splits $j()$ over all points in $\mathbb{R}^{q+1}$ finding the best oblique split is equivalent to finding the global maxima (minima) over $j()$.

Though this problem is still difficult, it is hoped that stopping the search early produces good oblique splits. Notable direct search approaches are covered in detail.

3.1.1 CART-LC: CART with Linear Combinations

Breiman et al. [1984, Section 5.2] proposed one of the earliest direct search approaches as an aside to their work on growing axis-parallel trees. Though they did not name their approach it has come to be known as "CART with Linear Combinations" (CART-LC). It can be summarized as follows,

1. starting from the best axis-parallel split (Algorithm 1),
2. look for a full oblique split with a high goodness value $j()$ by perturbing hyperplane coefficients (Algorithm 3),
3. "insignificant attributes" are removed in a heuristic manner from the full oblique split so that a concise oblique split[1] can be applied (Algorithm 2),.

A pseudocode description of CART-LC is given in Algorithms 1-3.

Algorithm 1 CART-LC

#Find an interpretable oblique split

Require: (a) goodness measure $j()$

#Scale continuous attributes
Centre each continuous attribute at its median and divide through by its interquartile range

#Find the best axis-parallel split
$H^{axis} \leftarrow$ best axis-parallel split

#Find a concise oblique split by considering perturbations from H^{axis}
$H^{concise} \leftarrow$ **Algorithm 2** CART-LC-Concise(H^{axis})

return the better of H^{axis} and $H^{concise}$ w.r.t. $j()$

Though CART-LC finds concise oblique splits, it does so in a highly heuristic manner and in effect only considers oblique splits that are close to the best axis-parallel split. As there is no reason why the global minima must reside there, oblique splits with lower values of $j()$ can surely be found.

3.1.2 SADT: Simulated-Annealing Decision Trees

Heath et al.'s [1993] "Simulated-Annealing Decision Trees" (SADT) improves upon CART-LC in that it performs a less prescribed search for the best oblique split. Their approach can be summarized as follows,

1. starting from some prespecified oblique split[2],

2. hyperplane coefficients are perturbed with $Unif[-0.5, 0.5)$ random variables under a simulated-annealing regime [Kirkpatrick et al., 1983, Cerny, 1985] whereby perturbed hyperplanes that increase $j()$ are always accepted while

[1]Breiman et al. [1984, pg. 133] were aware of the need to improve the interpretability of full oblique trees and are one of a small group of researchers who have tried to address this issue.

[2]Heath et al. consider the choice of the initial hyperplane to be irrelevant. They specify it to simply be $\sum_{i=1}^{q} X_i = 1$ as it is an oblique split that passes through all points $+1$ away from the origin on each coordinate axis, e.g. $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ when $q = 3$.

Algorithm 2 CART-LC-Concise(H^{axis})

#Find a good oblique split by considering small perturbations from H^{axis} (via Algorithm 3) then make it more interpretable by removing insignificant attributes

Require: (a) goodness measure $j()$ (b) tolerance variable β

#Find a full oblique split with a goodness value $j()$ by perturbing H^{axis}
$M \leftarrow \{1, \ldots, q\}$ – attributes over which perturbations are considered
$\{\sum_M a_i X_i = c\} \leftarrow$ **Algorithm 3** CART-LC-Oblique(H^{axis}, M)

#Improve interpretability of the oblique split found
repeat
  #Calculate $j()$ for the current oblique split
  $\Delta^\star \leftarrow j(\sum_M a_i X_i = c)$

  #Consider the effect of removing each attribute singly
  for each $m \in M$ **do**
    #Find c_m that maximizes $j()$ for hyperplanes of the form $\sum_{M \setminus m} a_i X_i = c_m$
    $\Delta_m \leftarrow \max_{c_m} j(\sum_{M \setminus m} a_i X_i = c_m)$
  end for

  #If within tolerance, remove the least significant attribute
  if $\Delta^\star - \max_m \Delta_m < \beta(\Delta^\star - \min_m \Delta_m)$ **then**
    #Identify least significant attribute
    $m^\star \leftarrow \arg\min_m \Delta_m$

    #Remove it and update the split
    $a_{m^\star} \leftarrow 0$
    $c \leftarrow c_{m^\star}$

    #Stop considering perturbations over $X_{m^\star}$
    $M \leftarrow M \setminus m^\star$
  else
    #Accept current oblique split as the best concise oblique split
    exit
  end if

until exit

#Having settled with a concise oblique split, update it to further increase $j()$
$H^{concise} \leftarrow$ **Algorithm 3** CART-LC-Oblique($\sum_M a_i X_i = c, M$)

return $H^{concise}$

Algorithm 3 CART-LC-Oblique

#Given an oblique split as a starting point, perturb it to improve $j()$

Require: (a) oblique split $\sum_{i=1}^{q} a_i X_i = c$, (b) a set $M \subset \{1, \ldots, q\}$ denoting which attributes to consider perturbations over (c) goodness measure $j()$ (d) tolerance variable ϵ

#Perturb current hyperplane until increases in $j()$ are sufficiently small
$n \leftarrow 1$
Label current hyperplane H_n
repeat
  #For each attribute $m \in M$, consider perturbations over a_m and c in H_n
  for each $m \in M$ **do**
    Let $\nu = \sum_{i=1}^{q} a_i X_i$, $\gamma \in \{-0.25, 0, 0.25\}$ and $\delta \in \mathbb{R}$
    Find $(\gamma^\star, \delta^\star)$ that maximizes $j()$ for the hyperplane $\nu - \delta(X_m + \gamma) = c$

    #Update the split
    $a_m \leftarrow a_m - \delta^\star$
    $c \leftarrow c + \delta^\star \gamma^\star$
  end for

  #Keeping attribute coefficients fixed, perturb c to further improve $j()$
  Find $c^\star$ to maximize $j()$ while keeping ν fixed

  #Prepare for next iteration
  $n \leftarrow n + 1$
  Label current hyperplane H_n

until $|j(H_{n-1}) - j(H_n)| < \epsilon$

return H_n

those that do not are accepted with decreasing probability,

3. simulated-annealing terminates when $j()$ remains unchanged for some pre-specified number of iterations[3].

Though SADT is able to find good oblique splits, there are still criticisms of it.

- Simulated-annealing is stochastic in nature so a different tree is grown at each attempt. Though it can be seen as an attractive property that allows different parts of the feature space to be investigated it leaves unanswered the question of which tree from the multitude that can be grown should be used.

- Full oblique trees are grown and no attempt is made to make them more interpretable.

- Each search for the best oblique split is dictated by the cooling rate of the simulated-annealing regime. This propagates throughout tree-growth.

3.1.3 OC1: Oblique Classifier 1

Murthy et al.'s [1993, 1994, 1995, 1998] "Oblique Classifier 1" (OC1) improves upon SADT in that it speeds up the search for each oblique split. It does so by replacing the simulated-annealing regime with a non-stochastic search, all else remains the same. For their search, hyperplanes that have converged are perturbed in a randomly chosen direction a preset number of times and multiple random starts[4] are applied to avoid local minima. A pseudocode description of OC1 is given in Algorithms 4-6.

[3]Heath et al. experiment with 3000 to 30000 iterations.

[4]Murthy et al. experiment with perturbing converged hyperplanes 5 times while applying 20 to 50 random starts.

Algorithm 4 OC1

#Find an oblique split with a low value of $j()$

Require: (a) badness measure $j()$ (b) the number of random starts $R \in \mathbb{N}$

#Find the best axis-parallel split
$H^{axis} \leftarrow$ best axis-parallel split

#Find R full oblique splits with low values of $j()$
for r in $1, \ldots, R$ **do**
  $H_r^{oblique} \leftarrow$ **Algorithm 5** OC1-Oblique
end for

#Identify the best oblique split that is found
$r^\star \leftarrow \arg\min_r j(H_r^{oblique})$

return the better of H^{axis} and $H_{r^\star}^{oblique}$ w.r.t. $j()$

Algorithm 5 OC1-Oblique

#Find an oblique split with low value of $j()$ from a random initial hyperplane

Require: (a) badness measure $j()$ (b) the number of escape attempts from local minima $S \in \mathbb{N}$ (c) a rule for indexing shyperplane coefficients Π

#Initialize a randomly chosen hyperplane
$H^{oblique} \leftarrow$ randomly initialized $(q+1)$-dimensional hyperplane

#Search for global minima of $j()$ while attempting to escape from local minima S times
for s in $1, \ldots, S$ **do**
  #Minimize $j()$ by perturbing hyperplane coefficients as indexed by Π
  repeat
    $H^{oblique} \leftarrow$ **Algorithm 6** OC1-Minimize($H^{oblique}, a_m$)
  until $j()$ converges at local minima

  #Attempt to escape minima by perturbing hyperplane in randomly chosen direction
  $R \leftarrow$ arbitrary $(q+1)$-dimensional hyperplane
  Choose $\alpha^\star$ to minimize $j()$ for hyperplanes $H_R^{oblique}(\alpha) = H^{oblique} + \alpha R$

  #Accept bumped hyperplane if $j()$ reduced
  if $j(H_R^{oblique}(\alpha^\star)) < j(H^{oblique})$ **then**
    $H^{oblique} \leftarrow H_R^{oblique}(\alpha^\star)$
  end if
end for

return $H^{oblique}$

Algorithm 6 OC1-Minimize

#Minimize $j()$ by varying a_m

Require: (a) Current hyperplane H (b) the index of coefficient to optimize over m

#Project training set onto X_m
for each observation in the training set **do**
  $h_k \leftarrow$ value of kth observation when evaluated on H
  $u_k \leftarrow \frac{a_m X_{k,m} - h_k}{X_{k,m}}$ where $X_{k,m}$ is the mth variable of the kth observation
end for

#Optimize $j()$ over univariate space
$a_m^\star \leftarrow$ best univariate split over u_k's w.r.t. $j()$

#Update current hyperplane
$H^\star \leftarrow \sum_{i \neq m} a_i X_i + a_m^\star X_m = c$

#Accept $H^\star$ under a regime that is similar to simulated-annealing
if $j(H^\star) < j(H)$ **then**
  return $H^\star$
  $P_{move} = P_{stagnation}$
else if $j(H^\star) \geq j(H)$ **then**
  return $H^\star$ with probability P_{move} and H with probability $1 - P_{move}$
  $P_{move} = P_{move} - 0.1 P_{stagnation}$
end if

The core of OC1 is encapsulated in Algorithm 6, it effectively approximates a $\mathbb{R}^{q+1}$ minimization problem with a series of 1-dimensional problems which are solved deterministically with ideas found in Breiman et al.'s CART book [1984, pg. 171–173]. Doing so removes the overhead associated with simulated-annealing and allows good oblique splits to be found more quickly.

As the only difference between OC1 and SADT is the way it finds local maxima (minima), OC1 still has the same weaknesses as SADT, i.e. a multiplicity of full oblique trees that are not always interpretable.

3.1.4 Similar Work

Others have followed in Murthy et al.'s footsteps to implement alternative minimum searching algorithms. Ideas similar to genetic algorithms [Holland, 1992] are used by Cantu-Paz and Kamath [2000, 2003] and Tan and Dowe [2004a,b] which result in algorithms that work in essentially the same way; "mutations" of splits (randomly perturbing hyperplanes) are compared with $j()$, "fit" splits (with lower values of impurity) are always accepted and "less fit" splits (that do not reduce impurity) are accepted with decreasing probability. As with Murthy et al. however, no attempt is made to address the other weaknesses of SADT.

3.2 Indirect Search Approaches

The approach taken by researchers who propose indirect searches can be summarized as follows,

1. oblique splits are simply linear decision boundaries,

2. they can therefore be found by solving classification problems with linear classifiers,

3. by specifying a set of classification problems at each stage of tree-growth (in effect a subset of the oblique split dictionary), tree-growth proceeds as usual by applying the best oblique split.

It is important to note that researchers who proceed in this way are not all seeking to grow oblique trees with binary partitioning splits and so different types of classification trees are produced. Though this may be the case, it is still interesting to consider the ideas that are applied. We begin with two definitions.

Definition 3.2.1 (Residual Observations)**.** Residual observations *at a node are those training set observations that fall to a node of interest during tree-growth.*

Definition 3.2.2 (Residual Classes)**.** Residual classes *at a node are the classes of residual observations that fall to a node of interest during tree-growth.*

3.2.1 FACT: Fast Algorithm for Classification Trees

One of the earliest approaches that apply this idea is Loh and Vanichsetakul's [1988] "Fast Algorithm for Classification Trees" (FACT). FACT trains LDA classifiers to residual observations at each node and so produces multi-way[5] oblique trees. As residual classes in latter stages of tree-growth become sparse, covariance matrices of residual observations needed to perform LDA become singular. Though solutions still exist, Loh and Vanichsetakul choose to apply LDA to the projection of residual observations to those principal components whose eigenvalues are greater than 0.05.

[5]Tests at internal nodes result in two or more child nodes.

3.2.2 Perceptron Trees and Linear Machine Decision Trees

Utgoff's initial work with "Perceptron Trees" [1989] was an attempt to extend perceptron learning ideas [Rosenblatt, 1958] to non-linearly separable two-class classification problems[6] and so only apply to two-class classification problems where observations appear in an online manner. They are grown by recursive use of *perceptrons*[7] as follows,

1. a perceptron is trained to observations as they arrive and can be thought of as a tree with two child nodes whose initial split is continually updated as observations arrive,

2. the leaves of the perceptron tree themselves become perceptrons when the dataset at its parent node is (heuristically) deemed to be non-linearly separable,

3. all perceptrons thoughout the tree are updated as observations arrive and leaves are continually partitioned as required.

A perceptron tree can either be seen as

1. a method for perceptron learning on non-linearly separable data or

2. an online approach to tree-growth that produces binary partitioning oblique trees for two-class classification problems.

Utgoff later changes focus to the latter of these by proposing "Linear Machine Decision Trees" (LMDT) [Utgoff and Brodley, 1991, Brodley and Utgoff, 1992a,b, Draper et al., 1994] that grows multi-way oblique trees in the normal setting (where

[6] A two-class classification problem is linearly separable if a hyperplane exists that partitions all observations of one class from the other.

[7] A perceptron is a function that computes a linear combination of the attributes and returns the sign of the result. It can be thought of as two-class classifier.

the data arrives in batch). It does so by training *thermal linear machines*[8] [Frean, 1992] to residual observations at each stage of tree-growth. Notably, Utgoff et al. tried to make thermal linear machines more interpretable during tree-growth though it is done heuristically.

3.2.3 Use of Linear Programming

Brown et al. [1996] propose solving linear programming problems similar to that of support vector machines to find oblique splits. Their approach grows binary partitioning oblique trees. With R residual classes $\mathcal{C} = \{C_1, \ldots, C_R\}$ at each node, the following problem is considered for each class C_i,

Minimize δ subject to:

$$X_i w - \delta \leq b \; \forall \; X_i \in C_i$$

$$X_i w + \delta \geq b \; \forall \; X_i \notin C_i$$

$$b + \sum w_k = const$$

The solution to this problem finds an oblique split that tries to partition observations of class C_i from those of all others. Having found R oblique splits, tree-growth continues as usual to apply the split with the lowest value of impurity. Bennett et al. [1993, 1997, 2000] also use linear programming to grow oblique trees.

3.2.4 P-BOT: Perceptron-Based Oblique Trees

The last approach we consider is misleadingly called "Perceptron-Based Oblique Trees" (P-BOT) even though it uses logistic regression to find oblique splits. Axelrod

[8] In brief summary, a *linear machine* [Utgoff and Brodley, 1991] is a multiclass classifier composed of several perceptrons. A *thermal linear machine* is a linear machine augmented to converge even with non-linearly separable data.

et al.'s [2005] P-BOT grows two-leveled tree-structured classifiers by concatenated use of logistic regression and does so as follows,

1. logistic regression is trained to a classification dataset to give predictions,
2. these predictions are used to partition training set observations into smaller subsets and so produces a multi-way tree,
3. by considering residual observations at each node as separate classification problems, logistic regression classifiers are trained at each node to give final predictions.

Axelrod et al.'s motivation for P-BOT was only to create two-leveled tree-structured classifiers with more flexible decision boundaries and so no attention was paid to making P-BOT more interpretable: all continuous and categorical attributes are simultaneously used as regressors for each logistic regression classifier. As a result the oblique splits that are found are very complicated (indicator functions are used to specify the factors of each categorical attribute).

3.3 Discussion

Direct and indirect searches are fundamentally different ways of finding oblique splits and should be seen as such; the former finds oblique splits with low values of impurity by changing hyperplane coefficients while the latter uses linear classifiers to produce oblique splits. As we wish to grow oblique trees with oblique splits that (1) have "low values of impurity" (2) are easy to find and (3) are concise, we compare each approach in this light.

Low Values of Impurity: The impurity of the final split is greatly influenced by the set of splits that one considers. Though direct searches in theory consider

the entire split dictionary, the impurity of the final split is highly dependent on how such a search is implemented. The same is true with indirect searches the only difference being the prespecification of the set of splits considered.

Computational Cost: The more splits considered the greater the computational cost. The computational cost of a direct search depends on how the stopping criteria is implemented while it is predetermined with indirect searches (the set of classification problems considered is prespecified).

More Interpretable Oblique Splits: Given the stochastic nature of direct searches, it is difficult to find concise oblique splits in ways other than heuristics. As for indirect searches, variable selection techniques are applicable (if probabilistic models are used to find oblique splits) though others have paid little attention to this.

Though the ability of any approach to find good oblique splits is highly dependent on its implementation, indirect searches offer greater control over computational costs while allowing well-founded variable selection techniques to be applied to find concise oblique splits. Chapter 4 presents a new indirect search approach that finds full oblique splits with low values of impurity and does so in a fast and intellectually appealing manner. The problem of obtaining more interpretable obliques trees is covered separately in Chapter 5 where these ideas are extended to make use of variable selection techniques.

Chapter 4

Oblique Splits via Logistic Regression

This chapter presents a new approach to growing full oblique trees. It is an indirect search approach that finds oblique splits with "low values of impurity" by using the information contained in the class of each observation in the training set. The question of how one obtains more interpretable oblique trees is deferred to Chapter 5.

This chapter is organized as follows. By revisiting original ideas behind tree-growth, Section 4.1 summarizes how impurity measures are often used to find good oblique splits. This sheds light on a family of two-class classification problems identified in Section 4.2 that forms the basis of the new approach. Details of how oblique splits are found from them are given in Section 4.3 followed by a summary of the approach in Section 4.4. Examples of oblique trees grown with these ideas are also given. Section 4.5 concludes with a heuristic argument that explains why this approach is so effective.

4.1 Re-examining Tree-Growth

Existing approaches seek to grow trees by applying the test that produces the "purest child nodes" at each stage of tree-growth. To achieve this,

1. an impurity measure $j()$ is defined that tries to quantify this idea,
2. $j()$ is evaluated over all permitted tests,
3. the test that is deemed to be best with respect to $j()$ is then applied.

We examine how $j()$ is constructed to quantify the concept of "purest child nodes".

Many approaches start by quantifying the impurity of a node. According to Breiman et al. [1984, pg. 24] such a measure should have the following properties,

Definition 4.1.1 (Node Impurity)**.** *Let p_i denote the proportion of training set observations of class C_i at a node for $i = 1, \ldots, k$.* Node impurity *is a function $\phi : (p_1, \ldots, p_k) \mapsto \mathbb{R}$ with $p_i \geq 0$ for $i = 1, \ldots, k$ with $\sum_i p_i = 1$ where*

1. *ϕ is maximal only at the point $(\frac{1}{k}, \ldots, \frac{1}{k})$,*
2. *ϕ is minimal only at the points $(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$,*
3. *ϕ is a symmetric function of the p_i's.*

That is to say, ϕ should be greatest when observed class probabilities are uniformly distributed over all classes and least when concentrated on a single class. Having defined node impurity, the impurity of a split is then defined follows,

Definition 4.1.2 (Split Impurity)**.** *Given a binary partitioning test S that partitions a proportion p_L of observations to the left child node L and a proportion p_R to the*

right child node R, the impurity of a split S *is given by*

$$j(S) = p_L\phi(L) + p_R\phi(R).$$

This definition seems reasonable as the impurity of a split is deemed to be high if the impurity of either child node is high. The following are commonly used measures of node (and therefore split) impurity,

$$\begin{aligned} \text{Gini Index:} \quad & \phi(p_1, \ldots, p_k) = \textstyle\sum_{i \neq j} p_i p_j = 1 - \sum_{i=1}^{k} p_i^2 \\ \text{Entropy:} \quad & \phi(p_1, \ldots, p_k) = -\textstyle\sum_{i=1}^{k} p_i \log p_i \end{aligned}$$

Regardless of which measure of split impurity is used, existing approaches rely heavily on $j()$ to decide which split to apply at each stage of tree-growth. A more directed approach to finding oblique splits that produce "purer child nodes" is proposed.

4.2 Ideal Outcomes and Ideal Oblique Splits

Why look for splits that produce "purer child nodes" by blindly comparing $j()$ over all splits when we already have information about where they lie?

Definition 4.2.1 (Ideal Outcomes)**.** *An* ideal outcome *(of a test) is one where all observations of some classes are partitioned from those of all other classes.*

Given the choice, each test of a tree would produce an ideal outcome. When there are R residual classes at a node, there are exactly $2^{R-1} - 1$ ideal outcomes[1].

Definition 4.2.2 (Ideal Oblique Splits)**.** *By considering ideal outcomes as two-class classification problems, linear decision boundaries are found when linear classifiers*

[1]There are exactly $2^{R-1} - 1$ ways of assigning R classes to two non-empty child nodes

are trained to them. Oblique splits found in this manner are called ideal oblique splits *(or simply* ideal splits*).*

These ideas are illustrated in Figure 4.1. With 4 residual classes, there are 7 ($= 2^{4-1} - 1$) ideal outcomes as shown by each subplot. The titles of each plot denotes the ideal outcome upon which it is based, e.g. {1}{2,3,4} denotes the case where all observations of class 1 are partitioned from those of all others. Ideal splits found with logistic regression are shown in each case. One sees that they generally replicate their associated ideal outcomes well, the ideal split based on {1}{2,3,4} in particular partitions most class 1 observations from those of all others as do ideal splits based on {3}{1,2,4}, {4}{1,2,3}, {1,3}{2,4} and {1,2}{3,4}. Some ideal outcomes however simply cannot be replicated well with oblique splits (with linear decision boundaries), i.e. ideal splits based on {2}{1,3,4} and {1,4}{2,3}.

Regardless of which measure of split impurity used, if $j()$ is able to quantify the idea of "purer child nodes", ideal splits that replicate their associated ideal outcomes well should automatically have low values of $j()$. Moreover as ideal splits are defined via ideal outcomes, they can be found directly.

4.3 Realizing Ideal Oblique Splits

As many classifiers are linear there is a choice as to how ideal splits are realized. Other than being linear, other qualities are also desirable of such a classifier, namely,

Inexpensive to train: as $2^{R-1}-1$ problems need to be solved at each node (where R is the number of residual classes).

Automated training: it should require minimal human input to train as we would

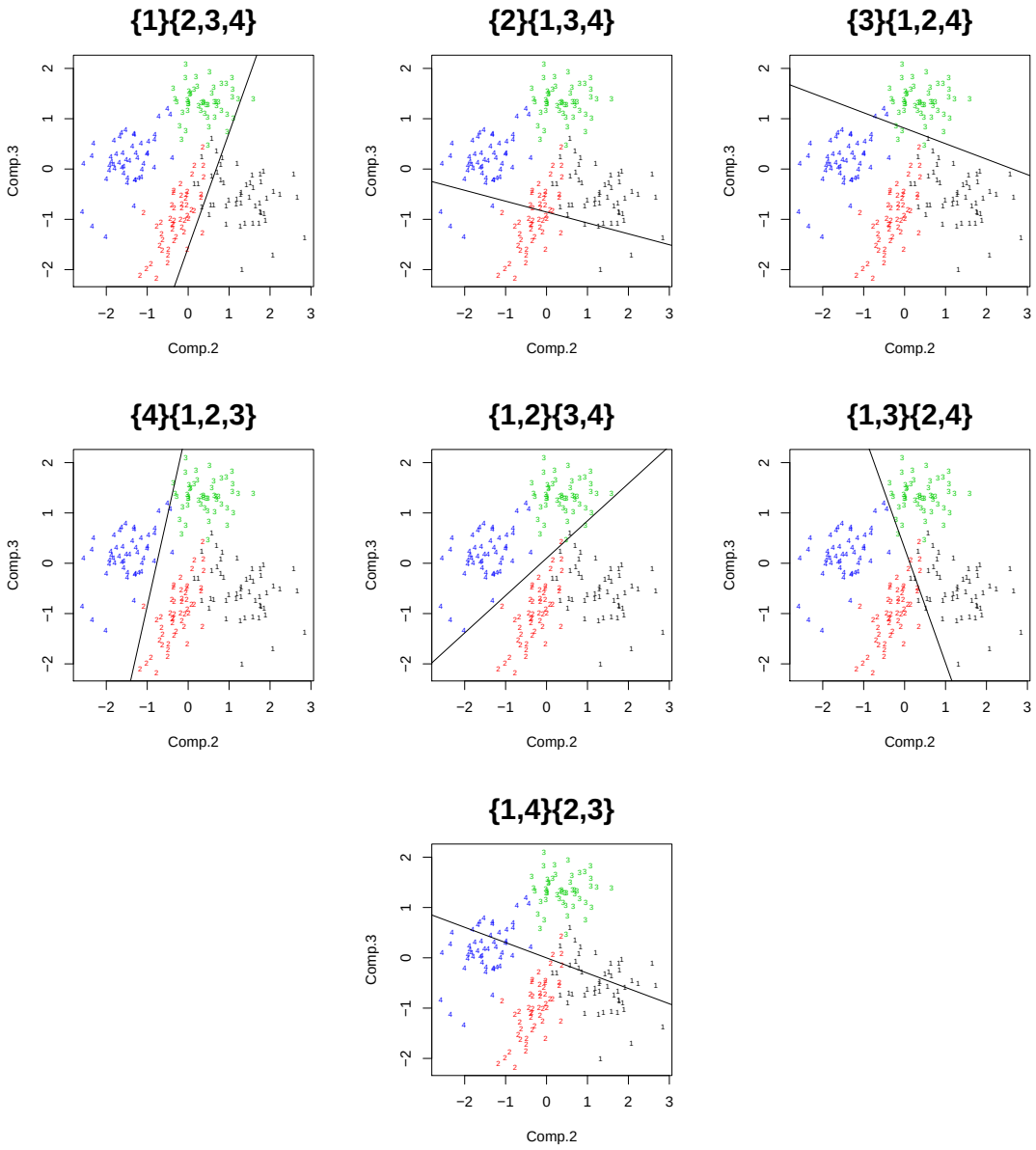


Figure 4.1: An ideal outcome of a test is one where all observations of some classes are partitioned from those of all others. With 4 residual classes, there are 7 ideal outcomes as shown in each subplot. By considering ideal outcomes as two-class classification problems, linear decision boundaries can be found by training linear classifiers to these problems, such splits are called ideal oblique splits. Realized ideal splits found using logistic regression are shown in each plot. Though ideal splits are not always able to perfectly replicate ideal outcomes, the outcomes produced are still very interesting.

like tree-growth to be fully automated.

Robust: it should not be overly influenced by outliers.

Separating hyperplanes: it should be able to find separating hyperplanes where they exist.

Applicable to well-founded variable selection ideas: though unrelated to the problem of realizing ideal splits, the ability to remove attributes without resorting to heuristics allows concise oblique splits to be found.

With these criteria in mind, we critique various linear classifiers to justify the preference of logistic regression [McCullagh and Nelder, 1989].

Perceptron Learners: perceptron learners are conceptually simple though have many drawbacks. They fail to converge in the presence of non-linearly separable data and even were they to converge its training time is highly variable.

Support Vector Machines: support vector machines that allow soft-margins [Cortes and Vapnik, 1995] overcome these weaknesses and are able to find "separating hyperplanes" even when data is not linearly separable. It does however require a quadratic programming problem to be solved which is computationally intensive.

Linear Regression: linear regression is easy to train and well-founded variable selection ideas are applicable. As it is not designed for classification problems however it may fail to find separating hyperplanes where they exist. It is also easily affected by outliers.

Linear Discriminant Analysis: LDA is easy to train. As it places strong assumptions on the structure of the data however it too can fail to find separating hyperplanes where they exist and is also easily affected by outliers.

Logistic Regression: Placing weak assumptions on the data and seeking to maximize a likelihood, logistic regression is a good choice for realizing ideal splits. It is robust to outliers, a separating hyperplane is always found where one exists and well-founded variable selection ideas are also applicable. Though there are situations where numerical maximization of the likelihood may fail, checks can be put in place to overcome these situations.

Accepting the use of logistic regression, attention is turned to how an expression for the realized ideal split can be extracted from fitted models. This turns out to be straightforward. Binomial logistic regression assumes that posterior log-odds of one class against the other (the 2nd to the 1st say) has the following form

$$\log \frac{P(G=2|X=x)}{P(G=1|X=x)} = a_0 + \sum_{i=1}^{q} a_i X_i. \tag{4.1}$$

As decision boundaries are located where posterior odds are equal it is simply where the left hand side of (4.1) is zero, i.e. an expression for the realized ideal split is given by the right hand side of (4.1) with its plug-in maximum likelihood estimates

$$0 = \hat{a}_0 + \sum_{i=1}^{q} \hat{a}_i X_i.$$

4.4 Growing Oblique Trees

With ideal splits in mind, oblique trees can be grown both quickly and deterministically in the following way. At each stage of tree-growth,

1. $2^{R-1}-1$ ideal splits are found (where R denotes the number of residual classes),

2. the impurity of each split is evaluated to identify the best ideal split,

3. tree-growth proceeds as usual by applying the best ideal split (with the same stopping criteria as before).

This approach to growing oblique trees has been implemented in an R package called *oblique.tree* [Truong, 2009] and is used throughout this thesis. Documentation for all exported functions are found in the Appendix.

A function called `oblique.tree()` in the R package *oblique.tree* allows various types of trees to be grown. They include,

1. Oblique Trees: that only consider oblique splits
   `oblique.tree(...,oblique.splits="only")`
2. Mixture Trees: that consider both axis-parallel and oblique splits during tree-growth (using whichever has the lowest value of impurity)
   `oblique.tree(...,oblique.splits="on")`
3. Axis-Parallel Trees: that only consider axis-parallel splits
   `oblique.tree(...,oblique.splits="off")`

An additional argument called `split.impurity` allows the measure of split impurity that is used during tree-growth to be toggled between `deviance` and `gini`.

Examples of maximal trees grown to various datasets with `oblique.tree()` follow (no attempt is made to improve their interpretability). The time taken to grow each tree has also been computed (averaged over five attempts) and were performed on a computer with an Intel Pentium 4 3.00Ghz CPU.

Type of Tree	Time (seconds)	
	Augmented Crabs	Pima Indians
Oblique Tree	0.546	0.092
Mixture Tree	2.678	5.126
Axis-Parallel Tree	2.240	5.334

Table 4.1: This table shows the time taken to grow various trees to the augmented crabs and Pima Indians datasets (averaged over 5 attempts). As `oblique.tree()` is written in R, the code is not optimized and so considers axis-parallel splits inefficiently. This however is not the case with oblique splits, the oblique tree is grown in half a second as there are only 4 classes in the augmented crabs dataset. It is quicker still on the Pima Indians dataset as there are only 2 classes.

4.4.1 Augmented Crabs Dataset

The original crabs dataset [Campbell and Mahon, 1974] consists of 5 morphological measurements of 200 crabs (50 crabs of two colours from both sexes) collected at Fremantle, W. Australia. Projecting this dataset onto its second and third principal components produces a simpler 2-dimensional dataset. This augmented crabs dataset is interesting to consider as decision boundaries of trees grown upon it can also be visualized. Various trees are grown to this dataset as shown in Figure 4.2. Computation times are tabulated in Table 4.1.

Looking at Figure 4.2, it is immediate that trees grown with oblique splits are smaller. It is also interesting to note that only 7 ($= 2^{4-1} - 1$) splits need to be considered when oblique splits are used as compared to the $995 \approx 199 \times 5$ odd for axis-parallel splits (the mixture tree considers everything), this is reflected in computation times in Table 4.1; the oblique tree is grown in 0.5 seconds while the mixture and axis-parallel trees take around 2.5. A final point to note is that as the mixture tree has more tests at its disposal, it continues partitioning where the oblique tree would otherwise stop.

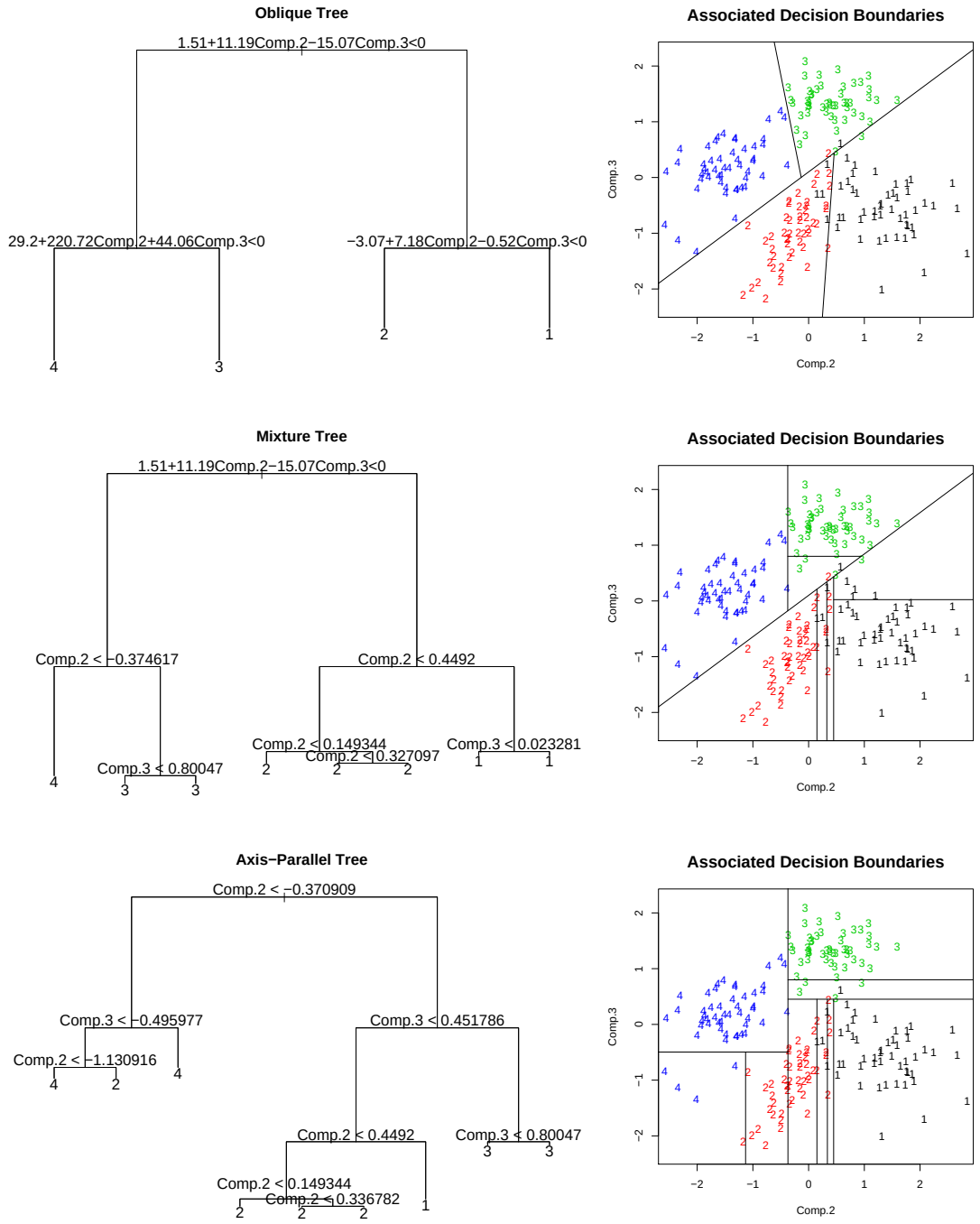


Figure 4.2: The augmented crabs dataset is obtained by projecting the original crabs dataset onto its second and third principal components. Though some information is lost, decision boundaries of trees grown to this dataset can be seen as shown above. The oblique tree only considers full oblique splits at each stage of tree-growth, the mixture tree considers both axis-parallel and oblique splits (applying the one with the lowest value of impurity) while the axis-parallel tree only considers axis-parallel splits. Looking at these trees, one immediately sees that using oblique splits allows smaller trees to be grown.

4.4.2 Pima Indians Dataset

The Pima Indians dataset [Smith et al., 1988] consists of 7 measurements of 532 women of Pima Indian heritage living near Pheonix, Arizona who were tested for the presence (or lack thereof) of diabetes. A subset of 200 observations called `Pima.tr` found in the *MASS* package in R is used as the training set (the others found in `Pima.te` are used as a test set). Various trees are grown to this dataset as shown in Figure 4.3. Table 4.1 shows computation times.

Looking at these trees, one immediately sees that the oblique tree uses much fewer tests than the other trees (the mixture and axis-parallel trees both have 22 leaves when the oblique tree has only 5). Though one might suspect this occurs only because the leaves of the oblique tree are less pure, examining training errors shows it is not so ($\frac{33}{200}$ oblique tree, $\frac{14}{200}$ mixture tree, $\frac{20}{200}$ axis-parallel tree); only a few oblique splits are needed to produce a tree with similar predictive power. Examining test error over `Pima.te` shows that the oblique tree actually generalizes better ($\frac{75}{332}$ oblique tree, $\frac{100}{332}$ mixture tree, $\frac{101}{332}$ axis-parallel tree) as well. It is important however to note that this comparison is not fair as cost-complexity pruning is often applied to maximal trees.

4.5 Locally Optimal Subset

As there are $\sum_{k=0}^{q} \binom{L_{gen}-1}{k} - 1$ splits in the oblique split dictionary it is surprising that a subset of only $2^{R-1}-1$ splits (constant in both N and q) can be used to grow such good trees. The following observation explains why most splits in the oblique split dictionary can and should be ignored which further helps to explain why ideal splits are so good. We begin with a definition,

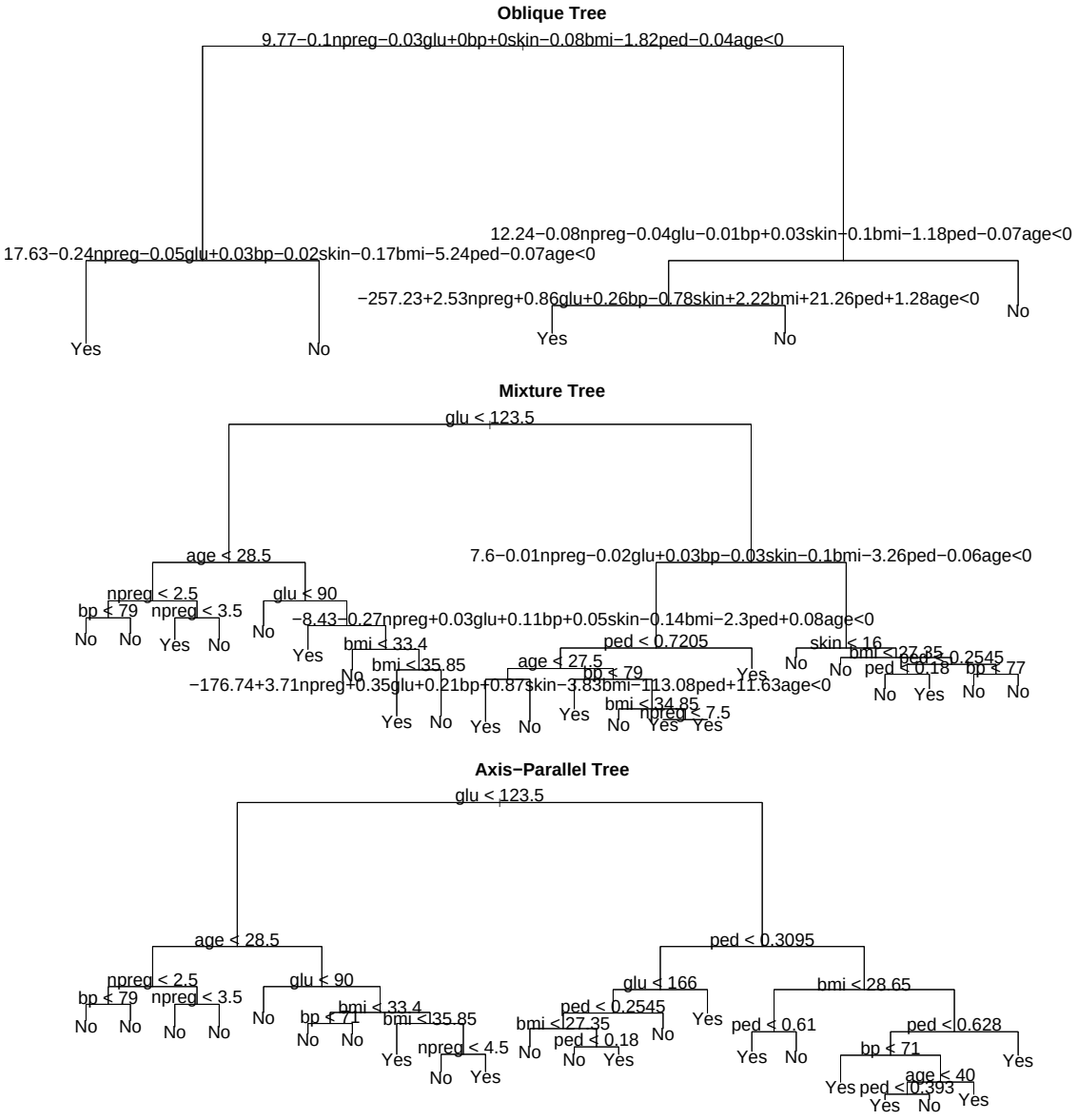


Figure 4.3: The Pima Indians training set consists of 7 measurements of 200 women tested for the presence of diabetes. Oblique, mixture and axis-parallel trees are grown to this dataset. Though the mixture tree has the same number of leaves as the axis-parallel tree, the oblique tree has much fewer leaves. This reduction in size comes at only a small cost to training error.

Definition 4.5.1 (Impurity Plot)**.** *Given a split dictionary and a measure of split impurity $j()$, an* impurity plot *shows attained values of $j()$ over all members of a split dictionary.*

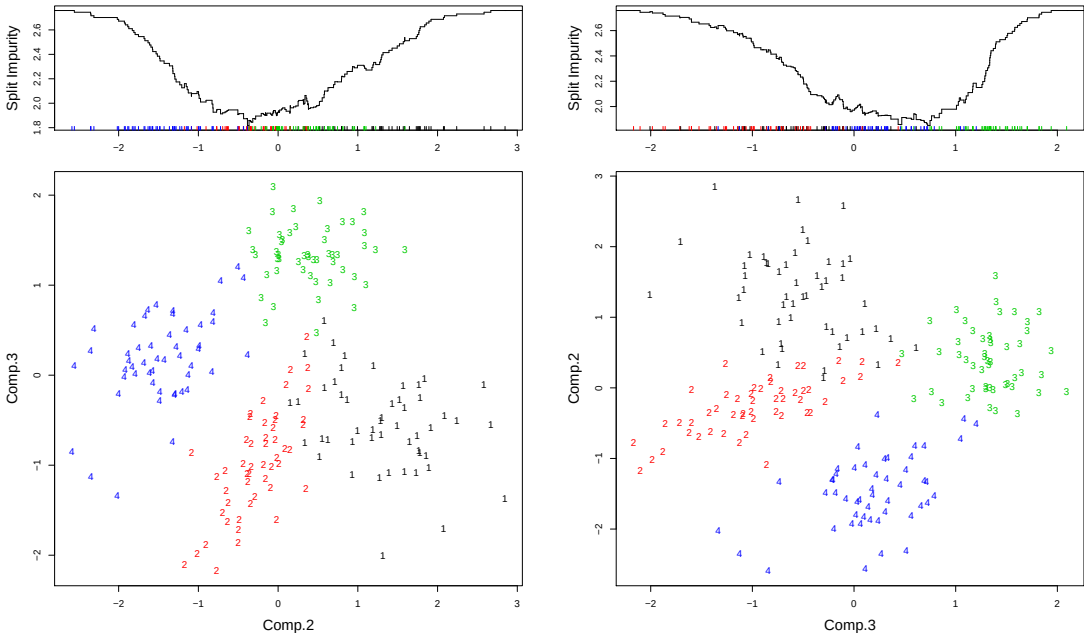
If the outcome of two tests S_1 and S_2 differ by only a small number of observations, $j(S_1)$ and $j(S_2)$ will be similar in value (for sufficiently large N). In addition to this, $j()$ should not jump around wildly as similar splits are close to each other. We confirm this empirically by examining impurity plots over axis-parallel and oblique split dictionaries for the augmented crabs dataset.

Figure 4.4 shows impurity plots over axis-parallel split dictionaries for each attribute of the augmented crabs dataset. Each plot is constructed as follows,

1. c, the cut-off point defining the axis parallel split in $X_i \leq c$ is varied to identify a split dictionary over X_i,
2. the impurity of each member of the split dictionary is evaluated,
3. $j()$ is plot against c and is known completely as it only changes at attained values of X_i.

Though we know $j()$ is a step function, it evolves in a fluid manner over this dataset. A large depression can be seen in the middle of each plot with smaller sub-depressions also visible upon closer examination. Though peculiarities in the dataset can cause such irregularities, they can also be attributed to situations where one class cluster terminates or when another begins (when viewing the projected data on X_i).

The information shown in Figure 4.5 is more complex, it shows the lowest value



(a) Evolution of $j()$ over axis-parallel splits on *Comp.2*.

(b) Evolution of $j()$ over axis-parallel splits on *Comp.3*.

Figure 4.4: The above plots show the evolution of the deviance split impurity $j()$ over axis-parallel splits on the augmented crabs dataset. Both evolve in a smooth fashion with a large depression in the middle of the range. Sub-depressions are also visible upon closer examination. Looking at Figure 4.4(a) in particular one sees that they occur when *Comp.*2 is around -0.8 (when observations of class 4 terminate and those of class 2 begin), -0.4 (class 4 terminates, class 3 begins), 0.3 (class 2 terminates, class 1 begins) and 1.2 (class 3 terminates). With Figure 4.4(b) they occur when *Comp.*3 is around -0.3 (class 2 terminates, class 4 begins), -0.1 (class 1 and 2 terminate) and 0.8 (class 4 terminates, class 3 begins).

of impurity attainable by oblique splits that pass through each point of the feature space over the augmented crabs dataset. It is constructed as follows,

1. a 100×100 grid of equally spaced points is defined over the continuous portion of the feature space $\mathbb{R}^q$ (as $q = 2$),

2. at each point of this grid, a restricted oblique split dictionary (whose splits pass through this point) is identified,

3. the lowest value of impurity that is attainable by oblique splits passing through each point is found by examining the restricted dictionary,

4. $j()$ is plot over the grid with others points found by interpolation.

Looking at Figure 4.5, though $j()$ is only shown approximately one sees that it does not jump around wildly over the feature space. A great deal of structure can also be seen in $j()$, it is low in areas between class clusters and high otherwise. Two valleys are formed which means that oblique splits with two types of orientation have low values of $j()$.

With these examples in mind, one sees that $j()$ does not jump around wildly over members of a split dictionary and though many tests may exist only a small number is of interest (for this dataset, only 7 axis-parallel splits from a total of around $398 = 2 \times 199$ and only 2 oblique splits from around $19900 = \sum_{k=0}^{2} \binom{200-1}{k} - 1$), i.e. similar splits can and should be avoided. As oblique splits with low values of $j()$ can also be characterized as lying in areas between contiguous class clusters, focusing on these areas should lead us to good oblique splits; which is exactly what ideal splits attempt to do.

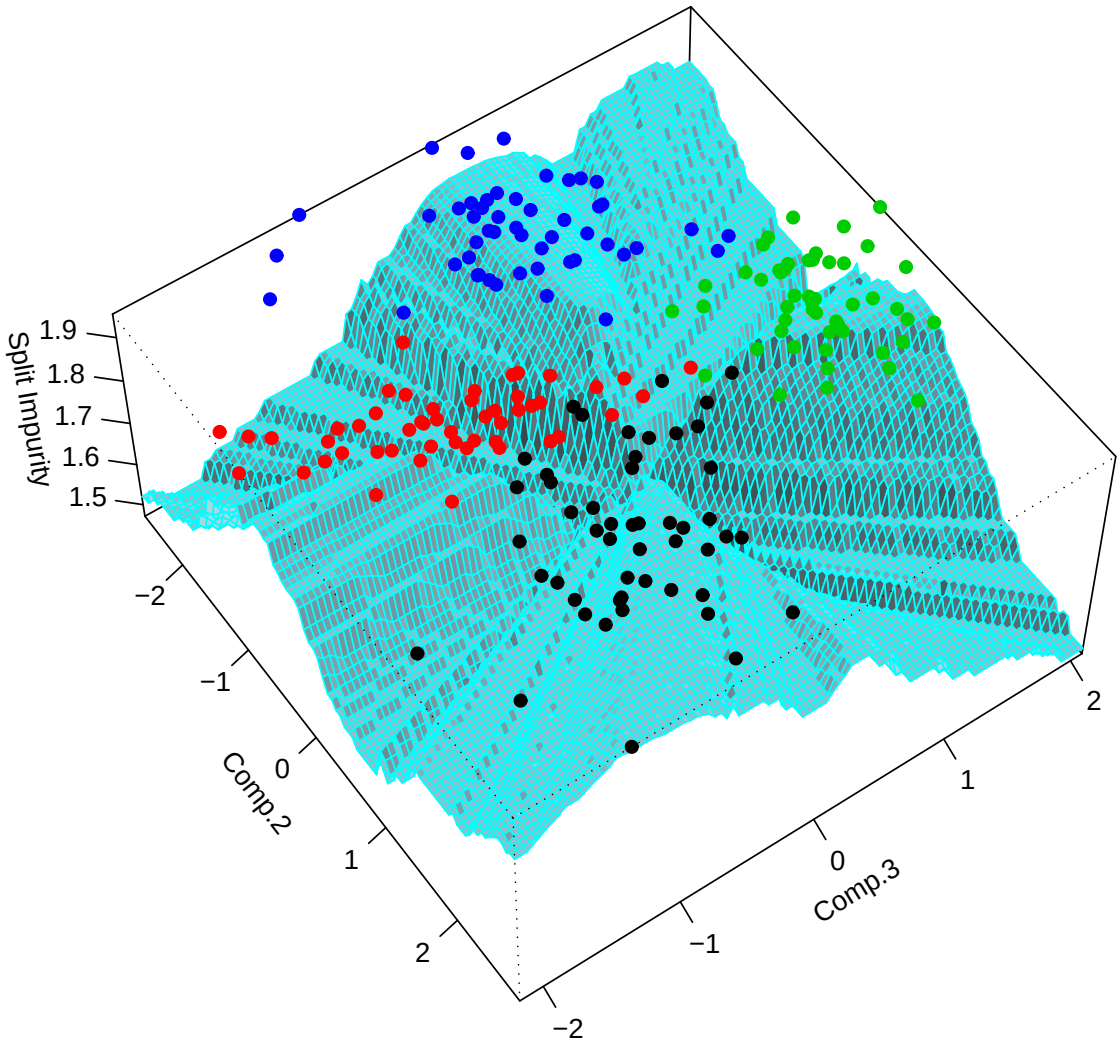


Figure 4.5: This plot shows the lowest value of the (deviance) split impurity attainable by oblique splits that pass through each point of the feature space over the augmented crabs dataset. The impurity surface does not jump around wildly and one can also see a great deal of structure; hills peak at centres of class clusters and valleys in areas in between. Two criss-crossing valleys are formed which suggests that oblique splits with only two types of orientations are really of interest over the entire split dictionary.

Chapter 5

More Interpretable Oblique Trees

Being able to grow oblique trees quickly is not enough: the end result must also be interpretable. This chapter introduces methods for improving the interpretability of oblique trees. They come in two forms, those applied

- during tree-growth or
- post-tree-growth.

All methods make use of well-founded variable selection ideas: possible because ideal splits are realized with logistic regression.

This chapter begins in Section 5.1 with a recap of variable selection techniques for logistic regression as it forms the basis for the work that follows. Methods for obtaining more interpretable oblique trees that are applied during tree-growth are presented in Section 5.2 with those applied post-tree-growth following in Section 5.3. Examples are given throughout.

5.1 Obtaining Concise Oblique Splits

As ideal splits are simply decision boundaries of logistic regression models, finding concise oblique splits at each stage of tree-growth simply becomes an issue of applying variable selection techniques to logistic regression models. We review two approaches for achieving this, model selection and penalized likelihood methods.

5.1.1 Model Selection

Given competing models for explaining some phenomenon, model selection provides a formal basis by which they can be compared in the spirit of Occam's Razor: we want

> an explanation that is as simple as possible, but no simpler.

We prefer simple models over more complex ones that fit the data "about equally well". Different ways of quantifying "about equally well" result in different model selection methods.

AIC

Akaike's "An Information Criterion" [1973, 1974] is easy to use. Given the log-likelihood of a model $l(\theta)$, calculate

$$AIC = -2l(\hat{\theta}_{MLE}) + 2q$$

where $\hat{\theta}_{MLE}$ is the maximum likelihood estimate of θ and where q is the number of parameters in the model. Models with low values of AIC are preferred as they generalize better to new data "on average" (the model with the lowest value of AIC

is often chosen).

AIC is motivated by the need to obtain models that predict well to previously unobserved observations. As training error underestimates test error, it is calculated that $2q$ quantifies this systematic optimism in the case where the true model is contained in the parametric family that is considered.

NIC

Murata et al.'s "Network Information Criterion" [1991, 1993, 1994] extends AIC to quantify the systematic optimism when the parametric family is not assumed to contain the true model[1]. It is motivated and used in the same way as AIC. NIC calculates

$$NIC = -2l(\hat{\theta}_{MLE}) + 2q^{\star}$$

where $q^{\star} = tr(KJ^{-1})$ with $K = var\frac{\partial l(\theta)}{\partial \theta}$ and $J = -E\frac{\partial^2 l(\theta)}{\partial \theta \partial \theta^T}$. Though NIC is well-founded, AIC is often used as it is easier to compute.

BIC

Schwarz's Information Criterion, more commonly known as the "Bayesian Information Criterion" [1978][2] provides a similar result. BIC calculates

$$BIC = -2l(\hat{\theta}_{MLE}) + q \log N$$

where $\hat{\theta}_{MLE}$ and q are as before and where N is the size of the dataset. As with AIC, models with the low values of BIC are preferred.

[1] A concise account of AIC and NIC is given by Ripley [1996, pg. 31–35].
[2] A concise account of BIC is given by Ripley [1996, pg. 62–65].

Motivated from a Bayesian framework, BIC's assumptions differ from those of AIC. Seeking to estimate the Bayes Factor for competing models, BIC looks for models that explain the data well rather than those that are good predictors so it in theory neither overfits or underfits to the true model (if the true model is considered and given enough data).

Concise Oblique Splits via Model Selection

Though motivated from different frameworks, AIC and BIC are used in the same way, the model with the lowest value of the information criterion is chosen. Though finding such a model is a difficult problem in itself, there are several established ways of tackling it,

Exhaustive Search: With q attributes, there are 2^q submodels. Every submodel is considered. Though it is possible to implement an exhaustive search for small values of q, it quickly becomes infeasible for q larger than 20.

Forward Search: Starting with a logistic regression model with only the intercept, the attribute that most improves the fit is added to the model until some stopping criteria is met.

Backward Search: Starting with all q attributes, the attribute that least degrades the fit is removed from the model until some stopping criteria is met.

Stepwise Search: Given some initial model, forward and backward searches are simultaneously considered until some stopping criteria is met.

Though an exhaustive search is the only way to ensure that the submodel with the lowest value of the information criterion is found, its computational cost is

too high; approximate searches must be used. Though little differentiates them, stepwise search from the full model is preferred for the following reasons,

1. the full model allows the oblique split that best replicates the ideal outcome to be found,

2. stepping away from it to slightly worse splits is better than looking for it from less informed states,

3. stepwise search offers greater flexibility without significant increase in computational cost.

5.1.2 Penalized Likelihood

Applying regression with highly correlated covariates or ill-conditioned problems produces poor parameter estimates, they can be wildly large and fitted models generalize poorly[3]. Penalized likelihood methods were originally developed to tackle these problems for linear regression [Hoerl and Kennard, 1970] and have since been extended to logistic regression models.

In penalized logistic regression, parameter estimates are found by maximizing an augmented log-likelihood that penalizes large coefficients (not including the intercept), i.e. given the model

$$\log \frac{P(G=2|X=x)}{P(G=1|X=x)} = a_0 + a^T X,$$

[3]Similar issues arise when logistic regression is applied to linearly separable data [Albert and Anderson, 1984].

parameter estimates maximize

$$PL(a_0, a) = l(a_0, a) - \lambda J(a)$$

for some penalty function J and penalization parameter $\lambda \in [0, \infty]$ with l as before.

It is interesting to note that original maximum likelihood estimates are recovered as $\lambda \to 0$ which is to say that $\tilde{\theta}_{PL} \to \hat{\theta}_{MLE}$ as $\lambda \to 0$. As $\lambda \to \infty$, parameter estimates are shrunk to zero as governed by the choice of J. Popular choices include,

L2 Ridge Penalty where $J(a) = \sum_{i=1}^{q} a_i^2$

It can be shown [Cessie and Houwelingen, 1992] that parameter estimates are shrunk to zero in a continuous manner when the L2 penalty is used (parameter estimates are nonzero until $\lambda = \infty$ upon which they all become zero).

L1 Lasso Penalty where $J(a) = \sum_{i=1}^{q} |a_i|$

Tibshirani introduced the lasso to linear regression [1996] where it was observed that parameter estimates are shrunk to zero in a nonlinear manner (some become zero before others). Similar behaviour has also been observed for logistic regression by Park and Hastie [2007] and Hastie et al. [2008] who compute entire paths traced by parameter estimates as λ varies from 0 to ∞).

Figure 5.1 illustrates how parameter estimates are shrunk for different choices of J.

As fitting penalized likelihood models with the L1 penalty exhibits behaviour that is similar to variable selection it often used for this task. Performing variable selection then becomes a problem of choosing the "right value" of λ which we now consider.

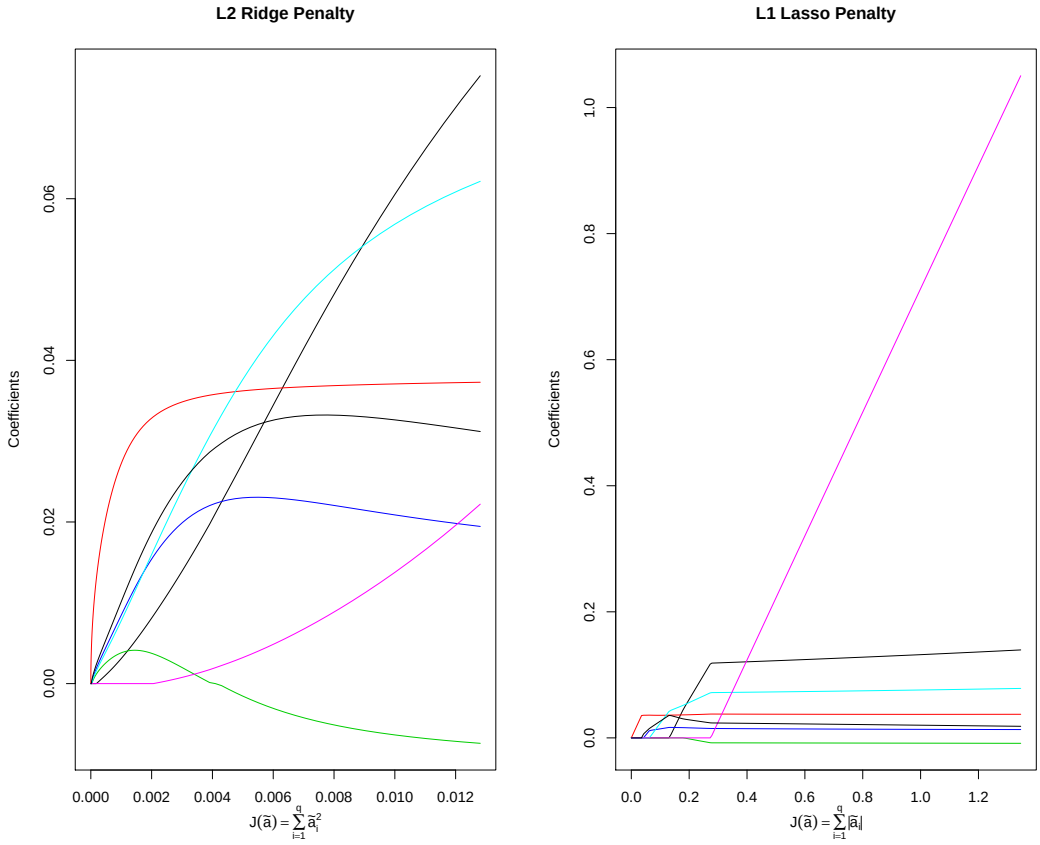


Figure 5.1: Penalized logistic regression finds parameter estimates by maximizing $PL(a_0, a) = l(a_0, a) - \lambda J(a)$ for some penalty function J and penalization parameter $\lambda \geq 0$. The choice of J determines how parameter estimates approach zero as $\lambda \to \infty$. The above plots show paths traced out by parameter estimates against attained values of J when L2 and L1 penalties are used (computed using the R package *glmnet* [Hastie et al., 2008]). For the L2 penalty, it can be analytically shown that parameter estimates are shrunk to zero continuously as seen here (coefficients do not reach zero until $\lambda = \infty$). The L1 penalty however focuses on removing one attribute at a time and so shrinkage occurs non-linearly. As such, L1 penalized logistic regression is often used for variable selection.

Concise Oblique Splits via Penalized Likelihood

Definition 5.1.1 (Transition Points of λ)**.** *When fitting L1 penalized likelihood models,* transition points *of the penalization parameter λ are those values where attributes enter or leave the fitted model.*

The only submodels that are considered are those that occur at transition points of λ which are easy to identify with Hastie et al.'s *glmnet* [2008] or Park and Hastie's *glmpath* [2007] R packages[4]). Among these, the transition point that produces the submodel that best generalizes is chosen. Though cross-validation is often used for this task, model selection ideas can also be applied to avoid additional computation which we now elaborate on.

In the case of non-maximum likelihood estimates, NIC and Moody's "effective number of parameters" p_{eff} [1991, 1992][5] can be used for this task. Park and Hastie [2006] propose computing

$$IC = -2l(\tilde{\theta}_{PL}(\lambda)) + cp \times edf$$

where l and $\tilde{\theta}_{PL}$ are as before, cp is some complexity parameter and edf is a measure of the effective degrees of freedom [Hastie and Tibshirani, 1990] of the penalized model. IC is used as an information criterion as before so $cp \in \{2, \log N\}$. Rather than computing Moody's p_{eff} Park and Hastie simply approximate it by q. They justifying it as follows. Zou et al. [2007] show that edf is exactly equal to q for L1 penalized linear regression models when considered in Stein's framework for unbiased risk estimation [1981]. As $\hat{\theta}_{MLE}$ and $\tilde{\theta}_{PL}$ are similar, they approximate

[4] `glmpath` approximates the paths of parameter estimates with piecewise constants. It is exact at transition points of λ.

[5] See Ripley [1996, pg. 35 and 140].

edf by q for L1 penalized logistic regression models as well (claiming that it performs well under simulation studies).

5.2 During Tree-Growth

As concise oblique splits are more interpretable than full oblique splits, trees grown with concise oblique splits should automatically be more interpretable. Though the idea is simple, complications arise with its implementation which are fortunately easy overcome. They include,

$2^{R-1} - 1$ Ideal Splits

Applying variable selection to find a single concise oblique split is easy, doing it $2^{R-1} - 1$ times becomes expensive. To reduce the cost of applying concise oblique splits, the only concise oblique split that is considered is the one found from the best ideal split[6].

Overly Concise Splits

Applying variable selection to an ideal split can render it illegal (the child node threshold of the stopping criteria mentioned in Section 2.1.2 is satisfied). As this is simply an artifact of the rules of tree-growth, applying a less concise split (found by accepting a larger value of the information criterion or a smaller value of λ) suffices.

The R function `oblique.tree()` used to grow full oblique trees in Chapter 4 can also be used to grow more interpretable oblique trees in this manner. It does

[6]We justify this computational shortcut as follows. The impurity of an ideal split is indicative of the linear separability of the ideal outcome upon which it is based. Ideal splits with high values of impurity are likely to be based on ideal outcomes that simply cannot be partitioned well with linear decision boundaries (as seen in Figure 4.1). If a full oblique split has a high value of impurity it is unlikely a concise oblique split found from it will have a low value of impurity. We examine the viability of this computational shortcut in Section 6.4.

so as follows, at each stage of tree-growth,

1. the impurity of all $2^{R-1} - 1$ ideal splits are evaluated,
2. the best ideal split is found,
3. variable selection is applied to this split to find the most concise oblique split that does not satisfy the child node threshold,
4. the concise oblique split found is applied and tree-growth proceeds as usual.

An argument called `variable.selection` specifies how variable selection is to be applied (defaulting to `"none"` which grows full oblique trees).

5.2.1 via Model Selection

Trees grown by considering concise oblique splits found by applying model selection to the best ideal split are called *model selection trees*.

`oblique.tree(...,variable.selection="model.selection.aic")`
grows *model selection AIC trees* that apply model selection with AIC while
`oblique.tree(...,variable.selection="model.selection.bic")`
grows *model selection BIC trees* that does the same with BIC.

Figures 5.2 and 5.3 show model selection trees grown to the augmented crabs and Pima Indians datasets respectively and compares them with full oblique trees, Table 5.1 shows their computation times. Looking at decision boundaries of model selection trees in Figure 5.2 one sees that concise oblique splits applied make sense; the initial split is retained while latter splits are replaced with concise oblique splits

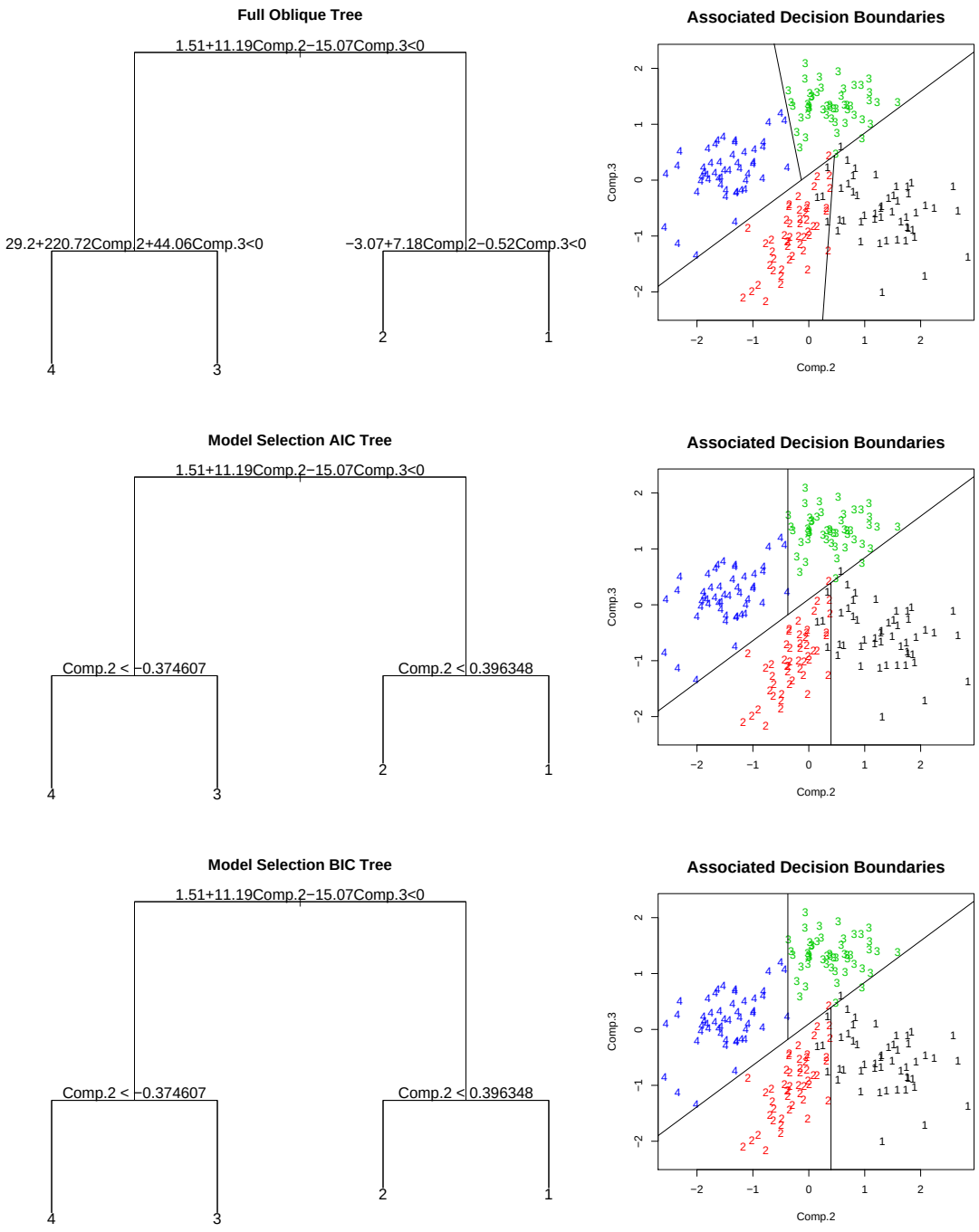


Figure 5.2: Model selection trees are grown to the augmented crabs dataset and compared with the full oblique tree. Looking at decision boundaries of the full oblique tree, one sees that latter splits partition observations well though little would be lost were axis-parallel splits instead used. This is done by both model selection trees. Training set observations are partitioned very well.

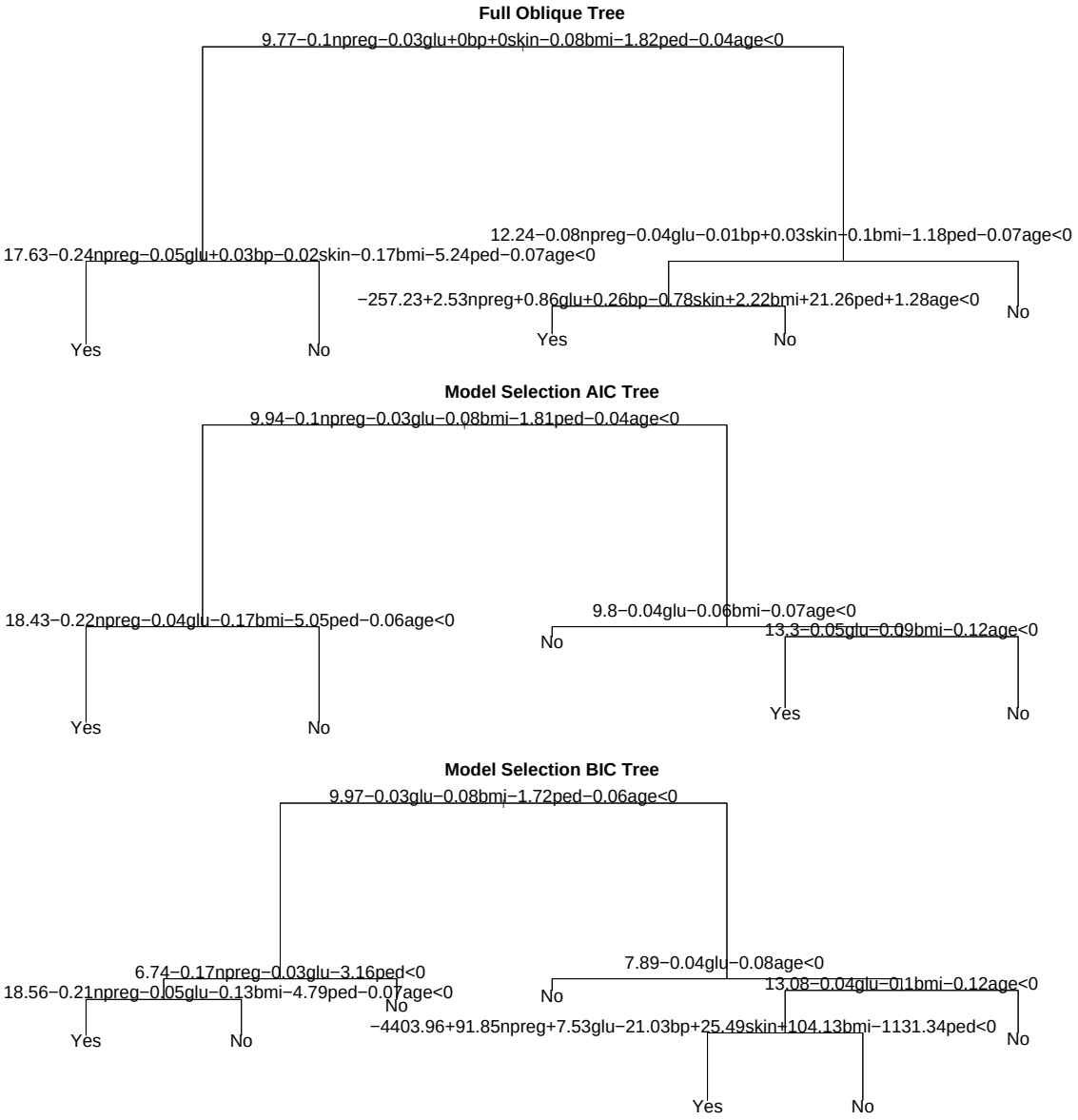


Figure 5.3: In the above plots, the depth of the lines between each node represent the reduction in deviance that results when such a split is applied. All three trees have similar tree structure, the initial split produces a large reduction in deviance and subsequent splits follow with smaller reductions. The structure of the model selection AIC tree in particular is identical to that of the full oblique tree, this suggests that splits from the full oblique tree are replicated well by concise oblique splits. The structure of the model selection BIC tree however differs, its initial split is so concise that additional splits are required in subsequent nodes. Both model selection trees use around half of the available attributes at each node and are more interpretable as a result.

Type of Tree	Time (seconds)	
	Augmented Crabs	Pima Indians
Full Oblique Tree	0.548	0.096
Model Selection AIC Tree	0.816	1.266
Model Selection BIC Tree	0.800	1.792

Table 5.1: This table shows the time taken to grow model selection trees to the augmented crabs and Pima Indians datasets (averaged over 5 attempts) and compares them to the time taken to grow a full oblique tree. As model selection is only applied to the best ideal split at each stage of tree-growth, concise oblique trees are grown quickly.

even though the child nodes they produce are less pure (test error becomes more important than training error). Though the same tree is grown irrespective of whether AIC or BIC is applied, one sees that its choice is important for trees grown to the Pima Indians dataset in Figure 5.3; splits applied by the model selection BIC tree are more concise.

Looking at Table 5.1 one sees that model selection trees are grown quickly. The difference between the times taken to grow full oblique and model selection trees depend on the dataset. As the Pima Indians dataset has 7 continuous attributes, applying a stepwise search from the full model is more computationally intensive than the augmented crabs dataset with only 2. The difference would be much higher were it not for only considering concise oblique splits from the best ideal split.

5.2.2 via Penalized Likelihood

Trees grown by considering concise oblique splits found by applying L1 penalized logistic regression to the best ideal split are called *lasso trees*.

```
oblique.tree(...,variable.selection="lasso.aic")
```

Type of Tree	Time (seconds)
Full Oblique Tree	26.128
Lasso AIC Tree	27.132
Lasso BIC Tree	36.568

Table 5.2: This table shows the time taken to grow lasso trees to the E. Coli dataset [Horton and Nakai, 1996] (averaged over 5 attempts) and compares them to the time taken to grow a full oblique tree.

grows *lasso AIC trees* that choose the penalization parameter λ with AIC while

`oblique.tree(...,variable.selection="lasso.bic")`

grows *lasso BIC trees* that does the same with BIC.

Figure 5.4 shows lasso trees grown to the E. Coli dataset [Horton and Nakai, 1996] and compares them with the full oblique tree. Computation times are shown in Table 5.2. The E. Coli dataset consists of 7 measurements of 336 E. Coli protein sequences from 8 classes (5 of which represent 97% $\approx \frac{327}{336}$ of the data, namely `cp`, `im`, `pp`, `imU` and `om`). Each tree is shown with uniform spacing between nodes to show each split more clearly. Though the structure of lasso trees is similar to that of the full oblique tree, they are much easier to understand as each split is more concise (especially so when BIC is used).

Looking at Table 5.2 one sees that the time taken to grow the full oblique tree is longer than in previous examples. This is simply due to the fact that observations come from more classes and so more ideal splits are considered at each node. Though this may be the case, 26 seconds is acceptable. The time taken to grow lasso trees is slightly inflated due to the efficiency of the path searching algorithms.

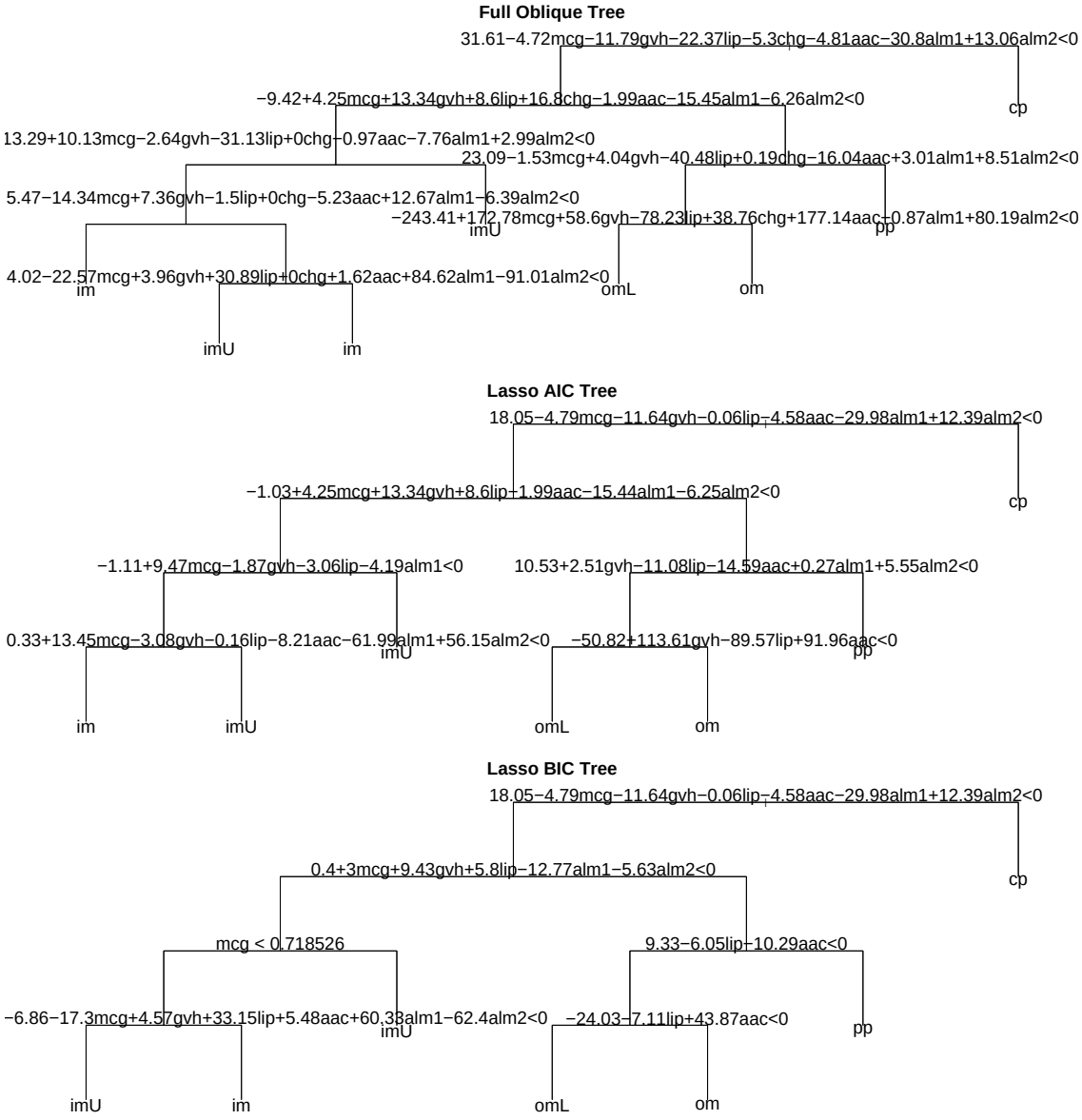


Figure 5.4: In the above plots, each tree is shown with uniform spacing between splits to show the splits more clearly. Looking at the structure of each tree, one sees they are very similar (lasso trees omit only one split). Splits applied by lasso trees are concise and are especially so when BIC is used. The training errors of each tree (full oblique tree $\frac{30}{336}$, lasso AIC tree $\frac{33}{336}$ and lasso BIC tree $\frac{33}{336}$) show that little predictive power is lost even though more interpretable trees are obtained.

5.3 Post-Tree-Growth

Though growing trees with concise oblique splits results in more interpretable oblique trees, what can be done if resulting trees are still not sufficiently interpretable? As the interpretability of a tree depends on the interplay between how observations of different classes are distributed and the splits that are used to partition them, little can be done other than using more complex splits unless we are willing to sacrifice the predictive accuracy of a tree. Given an oblique tree, post-tree-growth methods produce more interpretable oblique trees by extending cost-complexity pruning ideas to oblique trees.

5.3.1 Tree Pruning

Though originally developed for axis-parallel trees, cost-complexity pruning is equally applicable to oblique trees (Ripley's proofs [1996, pg. 221–224] do not require splits to be axis-parallel). To recap, for any given k cost-complexity pruning looks for rooted subtrees that minimize

$$R_k(T) = R(T) + k\ size(T), \tag{5.1}$$

The complexity of a CART tree is measured by counting its leaves (or equivalently 1 plus the number of splits). Though this may be reasonable for axis-parallel trees (adding a split to any part of the tree reduces its interpretability equally), it is not so for oblique trees (a tree will be less interpretable if it applies full oblique splits rather than concise oblique splits). As Ripley's proofs only require $size(T)$ to be an increasing function of the number of nodes, we generalized $size(T)$ to oblique trees as follows.

For each internal node I of a tree, let

$$attr(I) = \text{ \# attributes used by the split at } I$$

(this does not including the intercept), then define

$$comp(T) = 1 + \sum_{\text{internal nodes of } T} attr(I).$$

$comp(T)$ generalizes $size(T)$ to take into account the complexity of each split of a tree and coincide in the case of axis-parallel trees. Figure 5.5 computes its value for several trees to illustrate.

$comp(T)$ penalizes leaves that arise from oblique splits more heavily than those from concise oblique splits. Though the axis-parallel tree in Figure 5.5 has 9 leaves, it has the same value of $comp(T)$ as the mixture tree with only 8 leaves (as it applies an oblique split at its root node). This contrast is more pronounced for trees grown to the Pima Indians dataset. As $comp(T)$ is better able to quantify the complexity oblique trees, it is more appropriate to cost-complexity prune with $comp(T)$ rather than $size(T)$.

The R function `prune.oblique.tree()` implements this idea. An argument called `penalty` allows users to toggle between each measure of tree complexity.

`prune.oblique.tree(...,penalty="complexity")`
cost-complexity prunes with $comp(T)$ while
`prune.oblique.tree(...,penalty="size")`

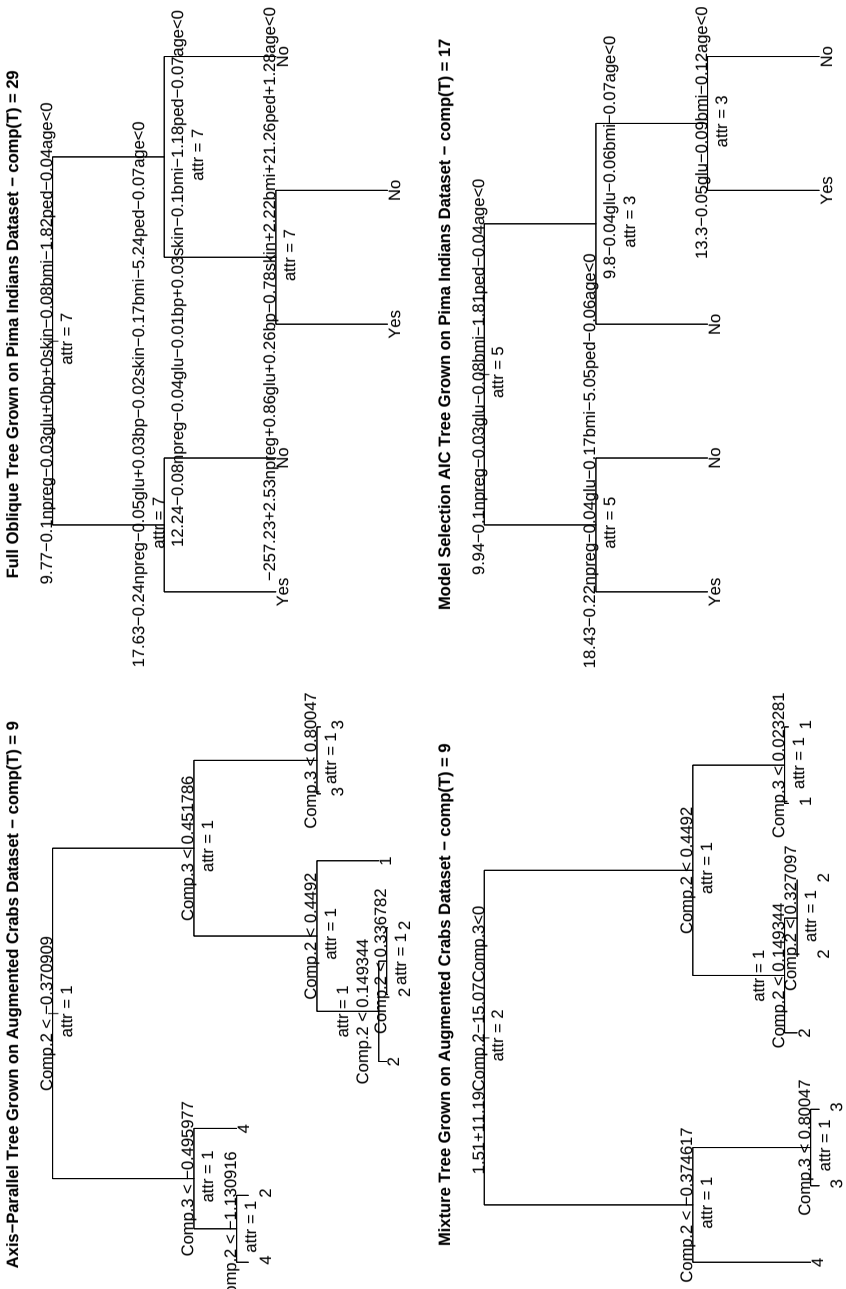


Figure 5.5: $comp(T)$ is a measure of tree complexity that takes into account the complexity of each split of a tree. We calculate it for trees grown to the augmented crabs and Pima Indians datasets. For the axis-parallel tree in the top-left, $comp(T) = size(T) = 9$. Though the mixture tree below it has one less leaf, it uses an oblique split in its root node so its value of $comp(T)$ is also 9. For trees grown to the Pima Indians dataset, though $size(T) = 5$ for both trees $comp(T)$ is inflated. The full oblique tree has a $comp(T)$ value of 29 as it uses all 7 attributes at each node, the model selection AIC tree however uses concise oblique splits so it has a smaller value of $comp(T)$ at 17.

does the same with $size(T)$.

The painters dataset found in the *MASS* package in R consists of 4 subjective scores (on a 0 to 20 integer scale) of 54 classical painters and is attributable to an Eighteenth century art critic called de Piles. Tree sequences found by pruning a model selection AIC tree grown to this dataset with either measures of complexity are shown in Figures 5.6 and 5.7. Rooted subtrees that minimize $R_k(T)$ as k varies over $\mathbb{R}$ are shown. Though both tree sequences start by pruning the same oblique split, they differ in latter stages.

5.3.2 Tree Trimming

Though growing trees with concise oblique splits allow more interpretable oblique trees to be found, one has little control over just how concise splits actually are. As the interpretability of a tree is highly dependent on the splits that are applied, *tree trimming* forces the splits of a tree to become more concise while keeping the structure of a tree fixed (this is analogous to cost-complexity pruning which instead keeps splits fixed while changing the structure of the tree).

Definition 5.3.1. *T^{con} is a* concise subtree *of T if*

1. *its tree structure is contained within that of T and*
2. *its splits are composed of a subset of those attributes used by T*

Figure 5.8 shows concise subtrees of a small tree to illustrate this idea.

Given a tree T, tree trimming tries to look for concise subtrees of T with low values of

$$R_h(T) = R(T) + h \text{ comp}(T),$$

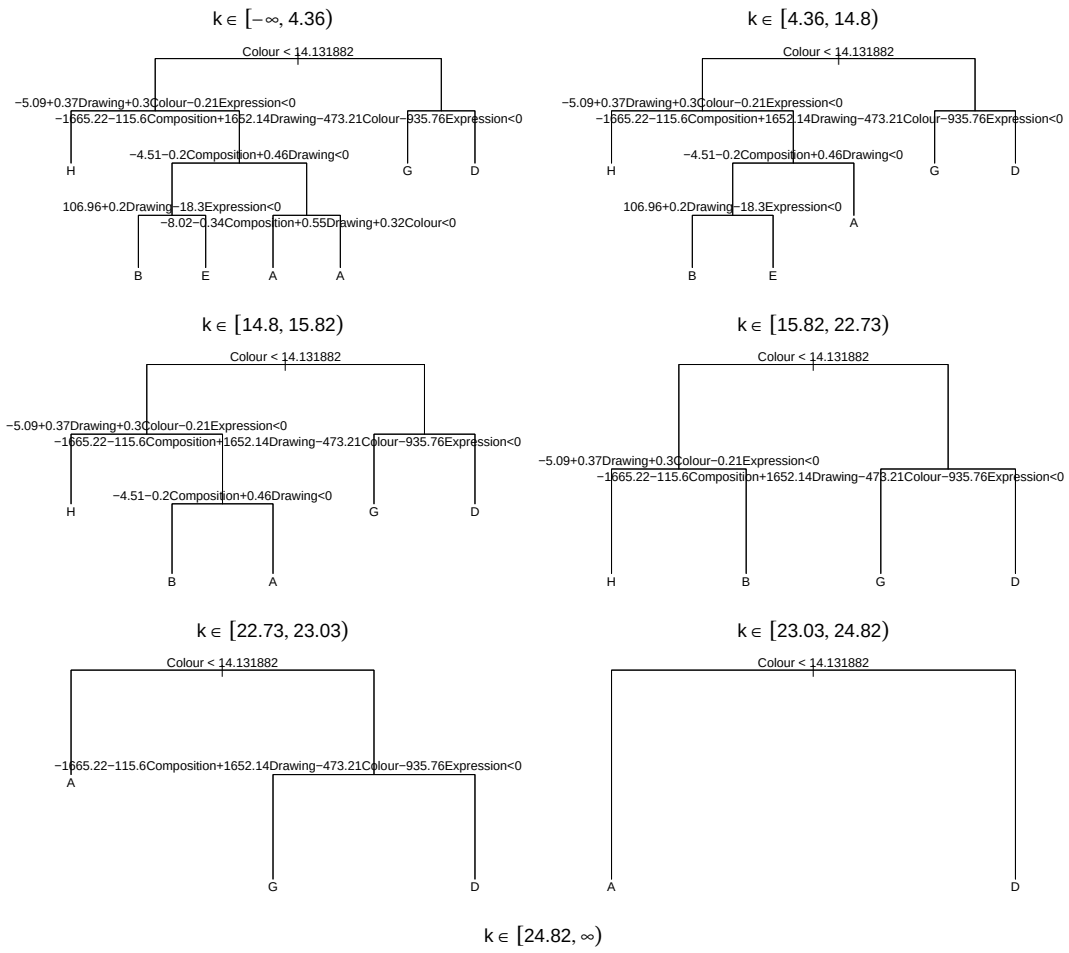


Figure 5.6: The above plot shows the tree sequence found when a model selection AIC tree grown to the painters dataset is pruned with the $size(T)$ measure of complexity.

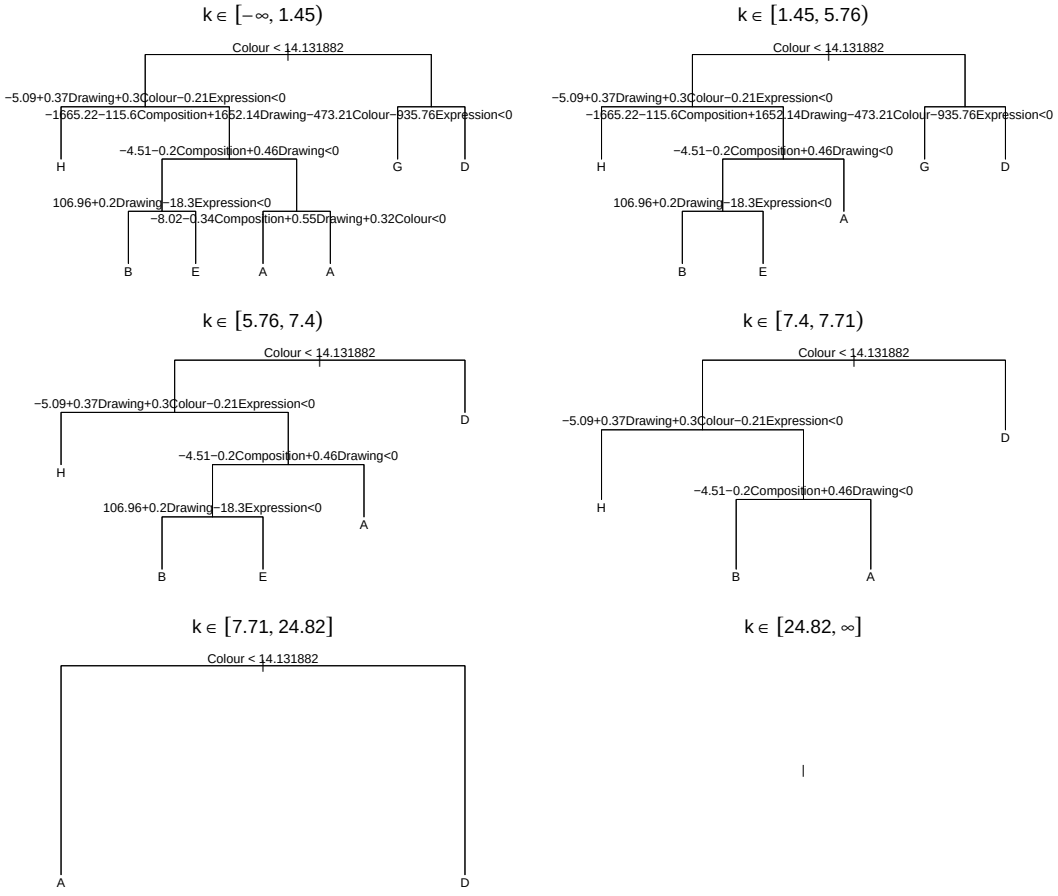


Figure 5.7: The above plot shows the tree sequence found when a model selection AIC tree grown to the painters dataset is pruned with the $comp(T)$ measure of complexity.

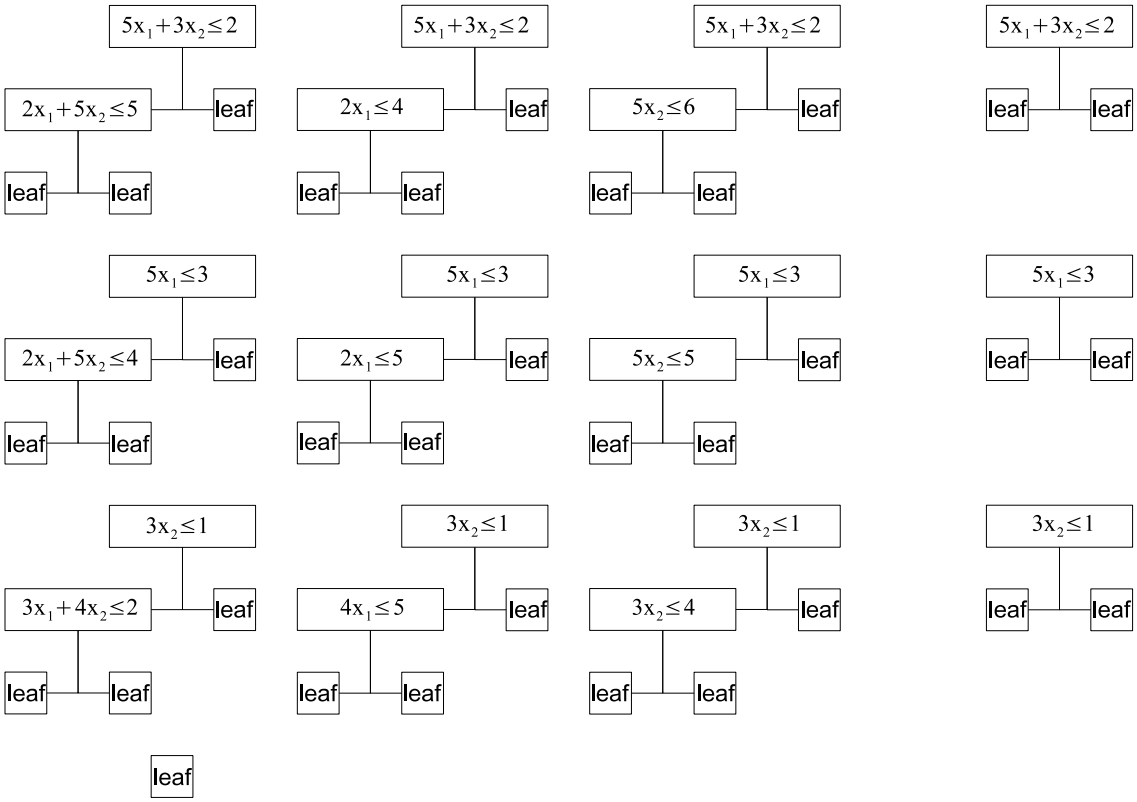


Figure 5.8: Concise subtrees of a tree are those that have the same tree structure and whose splits are composed of a subset of those attributes used by T, the parameter values need not be the same. All concise subtrees of the oblique tree in the top-left are shown. Though it is an artificial example, one sees that the number of concise subtrees of a tree is potentially large especially if many attributes are used by oblique splits.

more interpretable trees are sought at the lowest cost to predictive power.

Though easy to state, the elegant theory of cost-complexity pruning that allows efficient navigation of the trees of interest no longer hold (computational shortcuts afforded by the nested nature of rooted subtrees are no longer present). A greedy search is proposed to find concise subtrees of T with low values of $R_h(T)$.

At each internal node t of a non-trivial tree T, let T_t be the subtree rooted at t and T_t^{con} be a concise subtree of T_t whose split at t uses fewer attributes than T_t (the others remaining unchanged). Consider

$$g(T_t^{con}, T_t) = \frac{R(T_t^{con}) - R(T_t)}{comp(T_t) - comp(T_t^{con})}.$$

$g(T_t^{con}, T_t)$ compares the increase in R with the reduction in complexity that results when a more concise split is applied at t. By replacing the subtree at the node with the lowest value of $g(T_t^{con}, T_t)$ with its associated concise subtree, T is made more interpretable. Tree trimming performs this recursively until the tree can no longer be trimmed.

Though tree trimming is already highly restrictive of the concise subtrees it considers, there is still a great deal of freedom as to which concise subtree is considered at each internal node. Variable selection techniques can be used as T_t^{con} only differs from T_t in the split at t. In particular we use L1 penalized logistic regression as it allows oblique splits that range from the full oblique split to the constant predictor to be identified by just varying λ. Furthermore, if the constant predictor is considered to be a "split", T_t is effectively collapsed into a leaf (complete trimming), the

tree structure is retained if it is not (partial trimming). Either way, tree trimming produces a sequence of concise subtrees from which the one that best generalizes can be found and used.

The R function `trim.oblique.tree()` implements this idea. An argument called `trim.depth` controls whether trimming is `partial` or `complete` while an argument called `trim.impurity` specifies how $R(T)$ is to be measured.

`trim.oblique.tree(...,trim.depth="partial")`
trims trees until splits become axis-parallel while
`trim.oblique.tree(...,trim.depth="complete")`
trims trees until splits become leaves.

Tree trimming is illustrated in Figures 5.9 and 5.10. Figure 5.9 shows the *trim sequence* that is found when a model selection AIC tree grown to the Pima Indians dataset is partially trimmed. One sees that the trim sequence becomes more interpretable with each trim while the structure of the trees remains the same. It is interesting to note that as this happens, training set observations are allocated to different leaves of the tree so some splits may become redundant temporarily. Complete tree trimming however has the ability to simply prune away subtrees as seen in Figure 5.10.

Figure 5.11 examines training and test errors of concise subtrees from the complete trim sequence. Though training error increases as trees become more concise, test error (evaluated over `Pima.te`) broadly stays the same before dipping and increasing sharply. This suggests that the concise subtree with $comp(T) = 7$ gener-

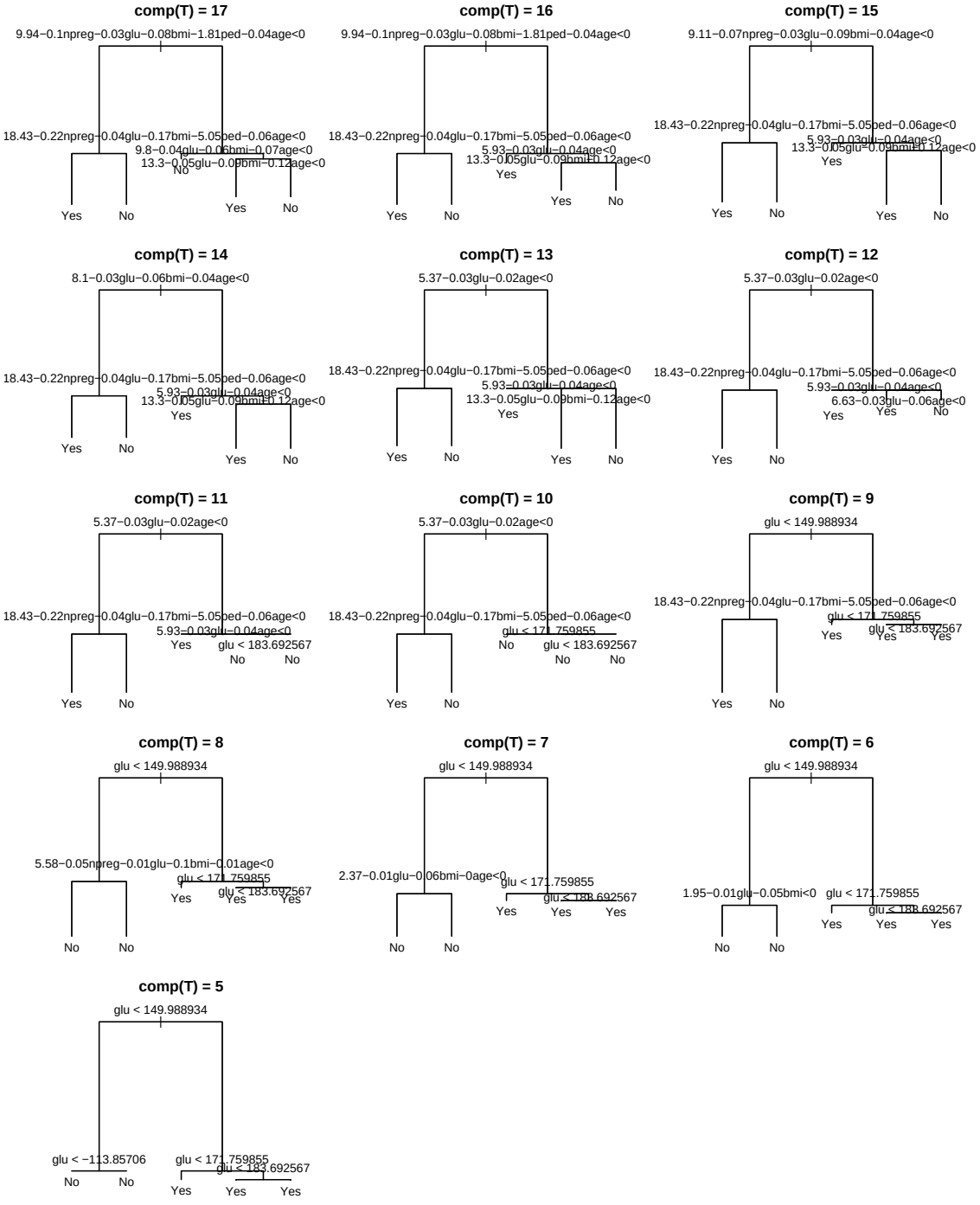


Figure 5.9: Partial tree trimming forces splits of a tree towards axis-parallel splits and is applied to a model selection AIC tree grown to the Pima Indians dataset. Training set observations are allocated to different leaves of the tree as splits become more concise. As the tree structure is retained, some splits may temporarily become redundant as seen with the oblique split immediately right of the root node. Concise subtrees with $comp(T) = 13, \ldots, 10$ simply pass training set observations to the right child node.

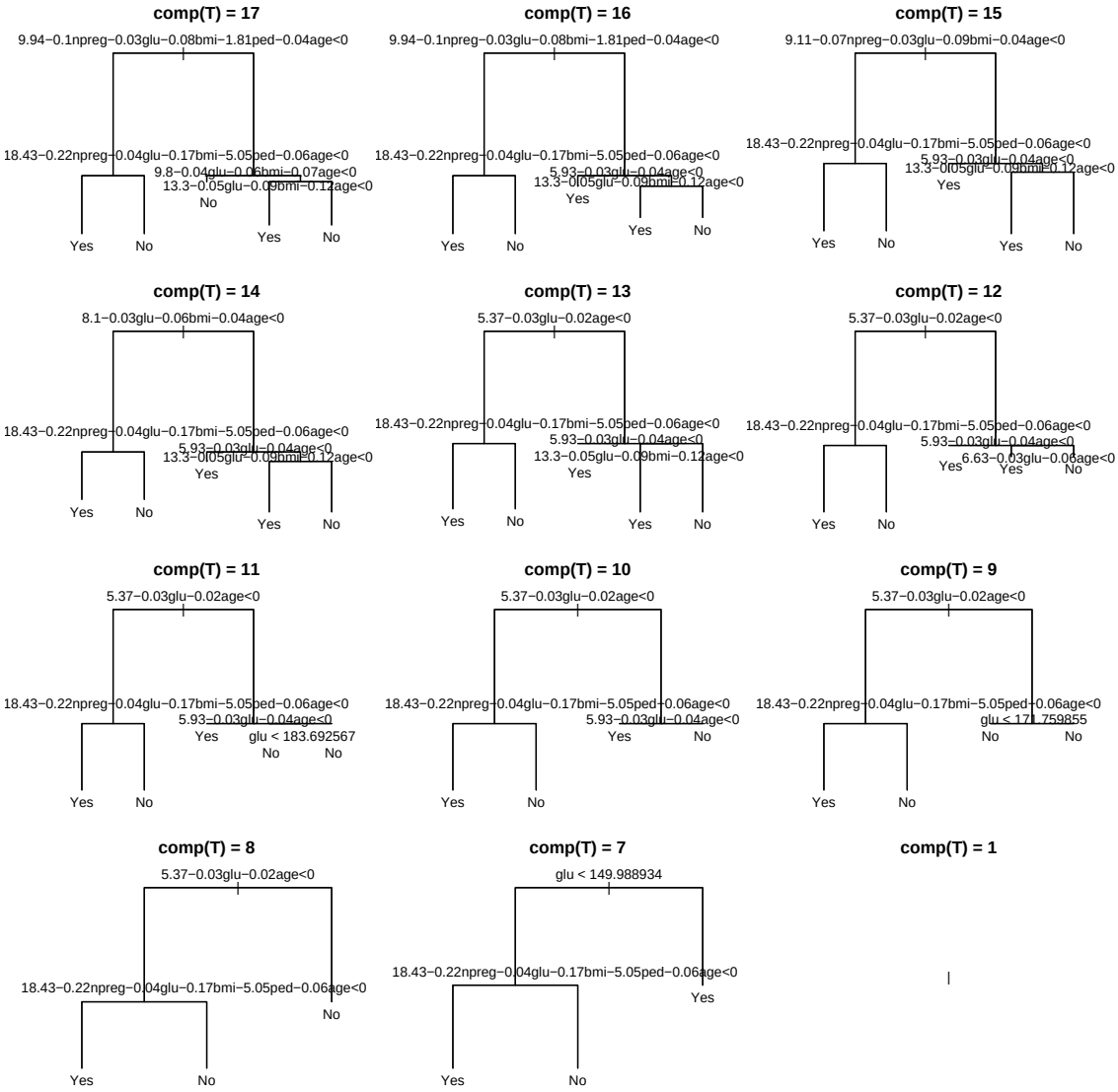


Figure 5.10: Complete tree trimming forces splits of a tree into leaves and is applied to the same tree in Figure 5.9 for comparison. Complete tree trimming differs from partial tree trimming in that axis-parallel splits are also considered for trimming, i.e. it can effectively prunes the tree. Though concise subtrees $comp(T) = 17, \ldots, 11$ are identical to those of the partial trim sequence, they differ from there on.

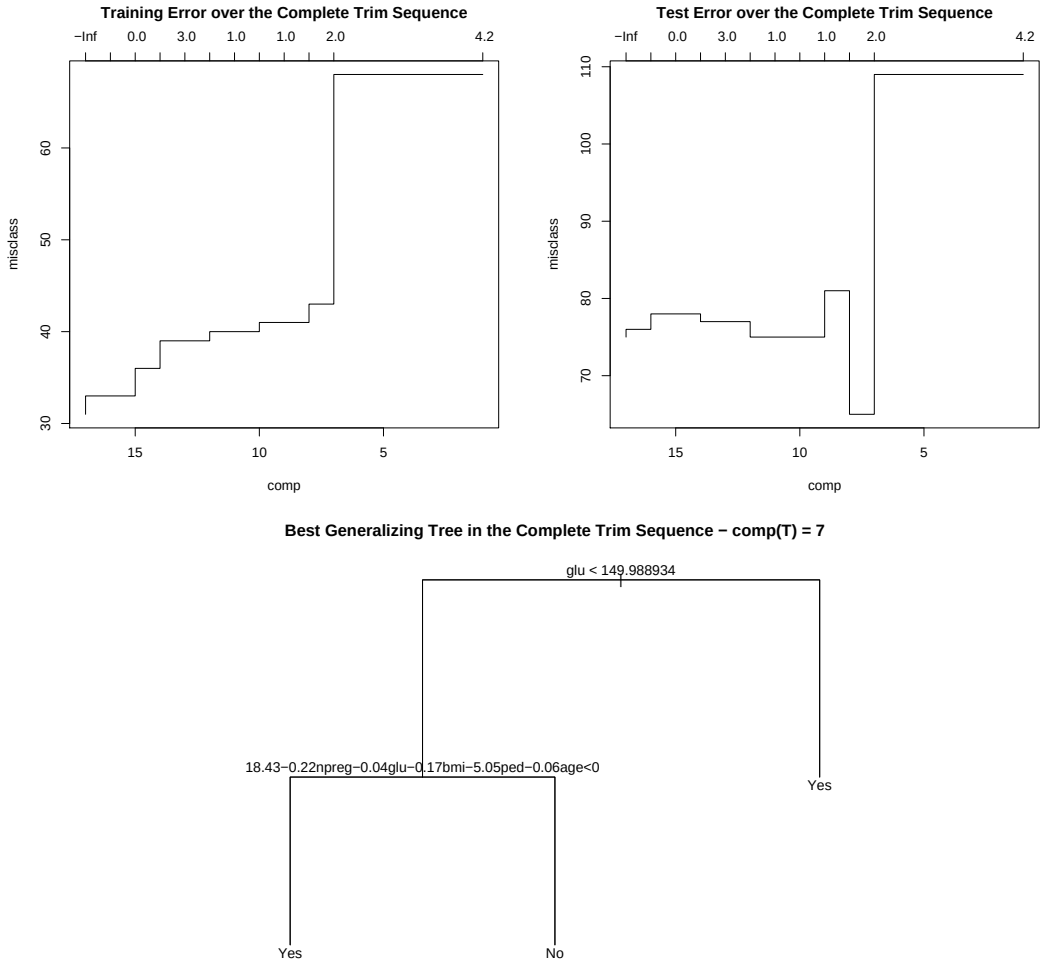


Figure 5.11: The above plots show misclassification rates for concise subtrees from the complete trim sequence over the Pima Indians training and test sets. Though training error increases as trees become more concise, test error broadly stays the same until $comp(T) = 7$ where it dips before jumping sharply. This suggests that the concise subtree with $comp(T) = 7$ generalizes best among all concise subtrees found when trimming completely.

alizes best over all concise oblique subtrees found when trimming completely.

Chapter 6

Further Considerations on Tree-Growth

Having proposed a new approach to growing oblique trees as well as various approaches to making them more interpretable, this chapter reflects on some of the finer details of its implementation. Section 6.1 begins with a discussion on how these approaches might be used to produce more interpretable oblique trees. As the proposed approach to tree-growth relies on the ability of ideal splits to find oblique splits with low values of impurity, Section 6.2 considers situations where they might fail to do so. Section 6.3 continues to analyze the computational complexity of these approaches and Section 6.4 concludes by analyzing the viability of a computational shortcut that is taken when growing concise oblique trees.

6.1 Obtaining More Interpretable Oblique Trees

Two types of oblique trees and various approaches to making them more interpretable are proposed. These include,

Types of Trees

- Oblique Trees
- Mixture Trees

During Tree-Growth:

- Model Selection AIC/BIC Trees
- Lasso AIC/BIC Trees

Post-Tree-Growth:

- Tree Pruning
- Complete/Partial Tree Trimming

Approaches that produce more interpretable trees applied during tree-growth are complementary to those that are applied post-tree-growth. In addition, tree pruning is complementary to tree trimming. How should one proceed with so many options?

As seen in examples, the effectiveness of each approach is problem dependent. As a full oblique tree is the best possible representation of the data with oblique splits, concise oblique trees should be compared to it before post-tree-growth methods are applied. If the full oblique tree is complicated, it is unlikely that highly interpretable oblique trees will be found. Having settled with a tree, tree pruning and trimming should then only be seen as tools for marginally improving its interpretability.

6.2 Poor Ideal Oblique Splits

The success of the proposed approach to growing oblique trees relies on the ability of realized ideal splits to have low values of impurity. As ideal splits are found via

logistic regression models, ideal outcomes that are linearly separable will always be perfectly reproduced. It was also shown in Figure 4.1 that when there is slight overlap between contiguous class clusters, realized ideal splits still have low values of impurity. We consider more extreme cases here to see where they might fail to do so, these include

Overlapping Class Clusters: when there is significant overlap of class clusters

Irregular Class Clusters: when contiguous class clusters are irregular

Disjoint Class Clusters: when classes appear with multiple clusters

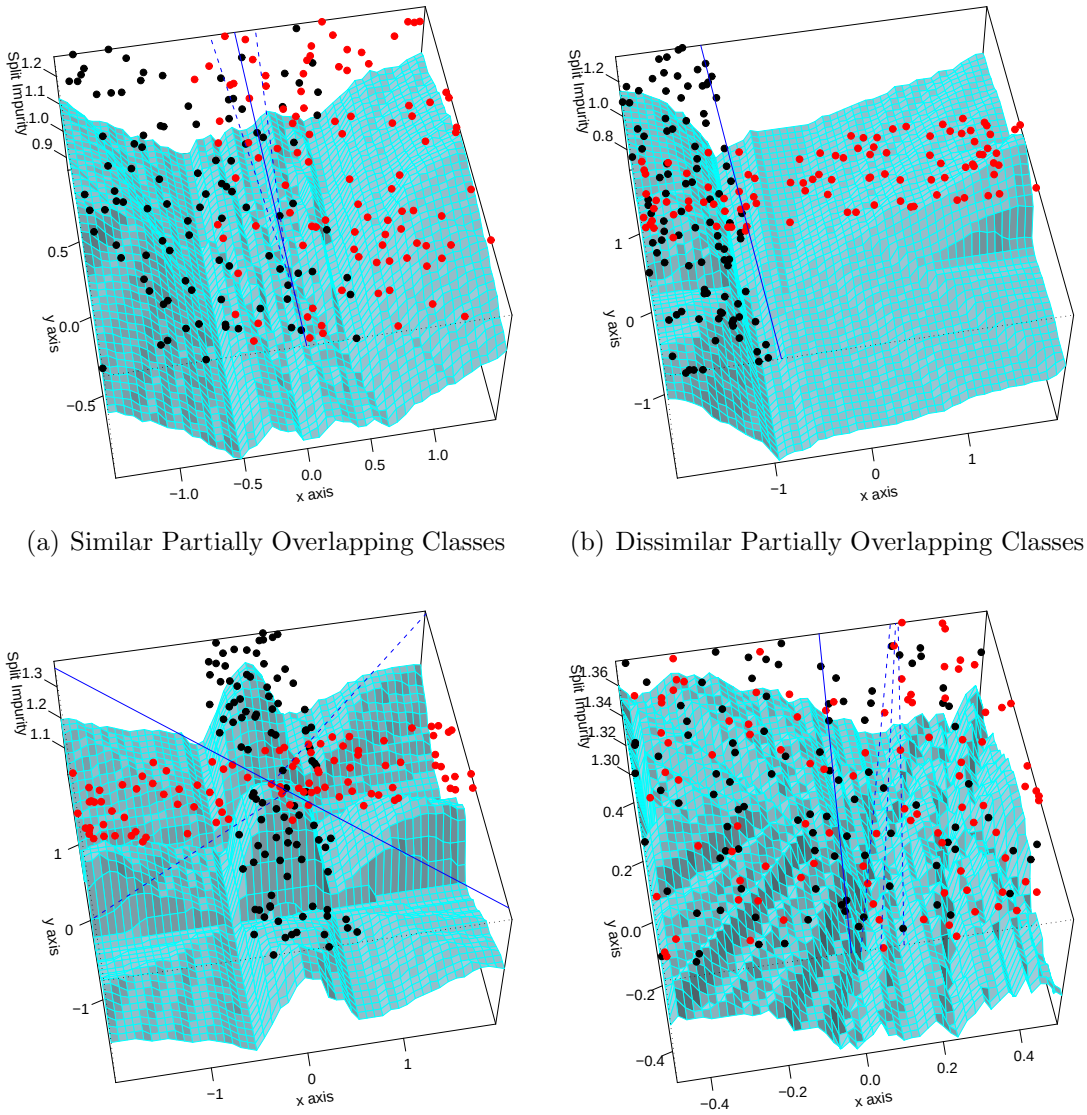
Two-class problems (with equal numbers of observations of each class) are considered in each case.

To demonstrate how ideal splits perform in each case, an impurity plot showing the lowest value of impurity that is attainable by oblique splits passing through each point of the feature space is shown along with decision boundaries of a full oblique tree grown to this data (the initial split is shown by a solid line and subsequent splits by dashed lines).

6.2.1 Overlapping Class Clusters

There are many ways class clusters can overlap which in turn affects how ideal splits are realized. Figure 6.1 considers various generic cases.

Looking at associated decision boundaries in Figures 6.1(a)-6.1(c), one sees that realized ideal splits that are found are interesting. When class clusters fully overlap however as shown in Figure 6.1(d), the ideal splits that are found do nothing of



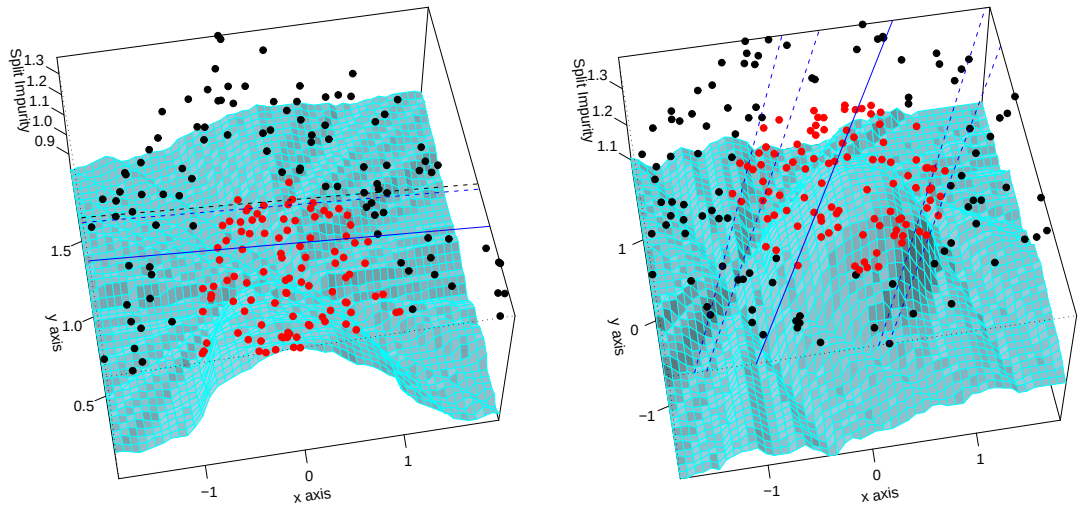
(a) Similar Partially Overlapping Classes

(b) Dissimilar Partially Overlapping Classes

(c) Dissimilar Fully Overlapping Classes

(d) Similar Fully Overlapping Classes

Figure 6.1: Overlapping (contiguous) class clusters are a small step away from linearly separable (contiguous) class clusters. Various cases are considered as the shape of class clusters affects how ideal splits are realized. Impurity plots for oblique splits are shown for each case as well as the decision boundaries of a full oblique tree grown to this data (the initial split is shown by a solid line and subsequent splits by dashed lines). In Figure 6.1(a), the initial split replicates the ideal outcome well as is true in Figure 6.1(b). In Figure 6.1(c) however the initial split no longer has a low value of impurity, applying it does however allow subsequent nodes to be partitioned well. In Figure 6.1(d), no oblique split could possibly replicate the ideal outcome as class clusters fully overlap. The ideal splits that are found do no better or worse than can be expected of them.



(a) A Partially Encircled Class (b) A Completely Encircled Class

Figure 6.2: With irregular (contiguous) class clusters in two-class classification problems, many situations can be likened to the above situations. With a partially encircled class in Figure 6.2(a), the initial split has a low value of impurity. This is not the case however when one class is completely encircled by the other as in Figure 6.2(b); the initial split has a high value of impurity. It is interesting to note that this happens when the ideal outcome upon which it is based simply cannot be replicated well with oblique splits.

value though nothing can and should be expected of them in such a situation. The only surprise occurs with the initial split in Figure 6.1(c). Though applying it allows an interesting oblique tree to be grown, it has a high value of impurity and the child nodes it produces are not purer. It is interesting to note however that the ideal outcome upon which it is based simply cannot be reproduced with oblique splits.

6.2.2 Irregular Class Clusters

Two situations are identified for irregular (contiguous) class clusters, the complete and partial encirclement of one class by the other.

With partial encirclement in Figure 6.2(a) the initial split that is found has a

low value of impurity. With complete encirclement in Figure 6.2(b) however the initial split fails to produce purer child nodes (applying it does however allow child nodes to be partitioned well). It is interesting to note that this again happens when the ideal outcome simply cannot be replicated well with an oblique split.

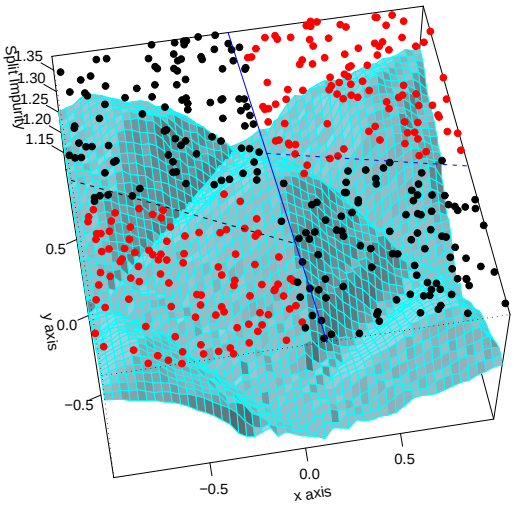
6.2.3 Disjoint Class Clusters

More extreme cases are considered here. Classes appear with multiple clusters in such a way that they simply cannot be partitioned with oblique splits. A problem that is similar to XOR is considered as well as situations where class clusters appear alternatively.

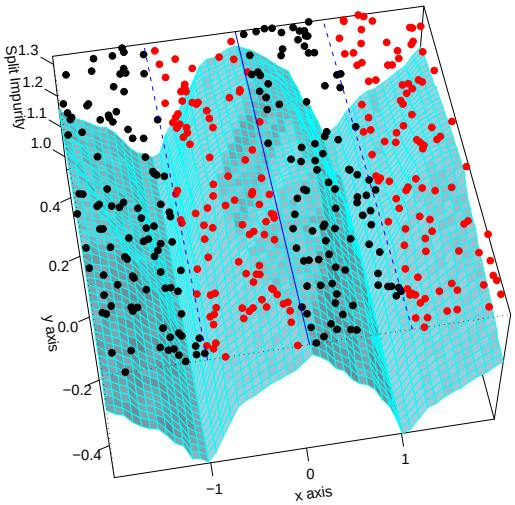
Looking at decision boundaries in Figures 6.3(a)-6.3(c), one sees that initial splits that are found are intuitively appealing and that the trees that are grown explain the structure of the data well. Though this may be the case, closer inspection reveals that all initial splits have high values of impurity.

6.2.4 Summary

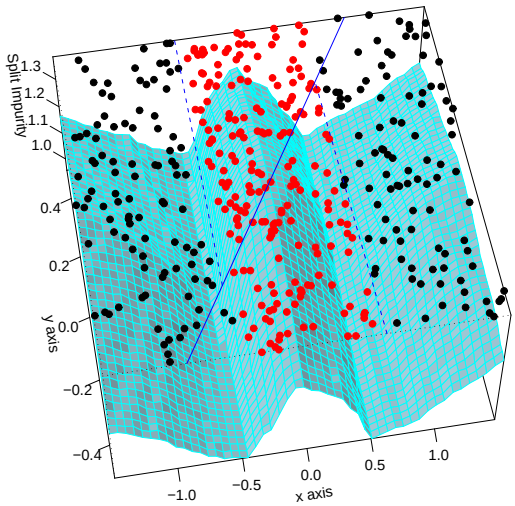
Having considered various artificial two-class classification problems, one sees that ideal splits realized with logistic regression have low values of impurity when ideal outcomes upon which they are based can be replicated well with oblique splits and fail to do so when they are not. It is comforting to see however that even in such cases, applying such a split paves the way for subsequent splits to partition the data well.



(a) XOR



(b) Unbalanced Alternating Classes



(c) Balanced Alternating Classes

Figure 6.3: Situations where observations of each class appear with multiple clusters that simply cannot be partitioned well with oblique splits are considered. The initial split in Figure 6.3(a) (XOR) makes sense though it has a high value of impurity. Its use however allows subsequent nodes to be easily partitioned. The same occurs in Figures 6.3(b) and 6.3(c) (unbalanced and balanced alternating classes).

6.3 Complexity Analysis

Having proposed various approaches of growing oblique trees (full oblique, model selection and lasso trees), we analyze their computational limitations. As the time taken to grow an oblique tree is highly variable, we instead focus on the complexity of partitioning a single node. There are three ways that a dataset can become more complex when partitioning over continuous attributes,

- $N \to \infty$, the number of observations increases

- $q \to \infty$, the number of continuous attributes increases

- $k \to \infty$, observations come from more classes

As shown in Section 2.2.2 the number of splits that need to be considered to partition a node with axis-parallel splits is $O(Nq)$. For oblique splits, though the number of splits that need to be considered is bounded above by $2^{k-1} - 1$, we also need to factor in the cost of realizing each ideal split. We thus turn our attention to the complexity of fitting binomial logistic regression models.

Seeking to maximize a likelihood, logistic regression models are fit by iterative methods as closed form solutions do not exist for maximum likelihood estimates [McCullagh and Nelder, 1989]. The Newton-Raphson algorithm is applied which gives an algorithm better known as iterative reweighted least squares (IRLS). It work as follows, letting

$\mathbf{y} \in \mathbb{R}^N$ denote the vector of binary class classifications (0 or 1),

$\mathbf{X} \in \mathbb{R}^{N \times (q+1)}$ denote the design matrix (with a column of leading 1's),

$\mathbf{p} \in \mathbb{R}^N$ denote the vector of fitted probabilities with ith element $p(x_i; \beta^{old})$ and

$\mathbf{W} \in \mathbb{R}^{N \times N}$ denote a diagonal matrix with elements $p(x_i; \beta^{old})(1 - p(x_i; \beta^{old}))$

each iteration of IRLS requires

$$\left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right) \beta^{new} = \mathbf{X}^T \mathbf{W} \mathbf{z} \tag{6.1}$$

to be solved for β^{new} where $\mathbf{z} = \mathbf{X}\beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$. This is done by QR factorization in the *oblique.tree* R package[1] at a cost of $O(Nq^2)$. With this in mind, the complexity of applying full oblique splits is $O(Nq^2 \cdot i_{IRLS} \cdot 2^k)$ where i_{IRLS} is an upper bound on the number of iterations of IRLS.

When finding concise oblique splits, complexity depends on how variable selection is applied.

Model Selection: as a stepwise search is only applied to the best ideal split, complexity is given by $O(Nq^2 \cdot i_{IRLS} \cdot \max\left\{2^k, i_{IC}\right\})$ where i_{IC} is an upper bound on the number of stepwise searches applied to the best ideal split.

Penalized Likelihood: as penalized models at all transition points of λ of the best ideal split need to be identified, complexity is given by $O(Nq^2 \cdot i_{IRLS} \cdot \max\left\{2^k, i_\lambda\right\})$ where i_λ is the cost of tracing entire paths of parameter estimates relative to the cost of a single logistic regression fit[2].

Where IRLS converges [Albert and Anderson, 1984, Santner and Duffy, 1989], 3-4 iterations normally suffice so i_{IRLS} is often negligible. In addition to this, the impact of i_{IC} and i_λ on complexity is mitigated as variable selection is only applied

[1] Logistic regression is fit using `glm()` from the *stats* R package which solves $(\mathbf{WX})\,\beta^{new} = \mathbf{Wz}$ by QR factorization.

[2] Hastie et al. [2008] claim $i_\lambda \approx 1$ when *glmnet* is used, i.e. the cost of evaluating entire paths of parameter estimates with `glmnet()` is similar to that of a single logistic regression fit. When using Park and Hastie's [2007] *glmpath*, $i_\lambda \leq q$. For ease of implementation the *oblique.tree* R package uses the latter of these.

	Complexity
Axis-Parallel Splits	$O(Nq)$
Full Oblique Splits	$O(Nq^2 \cdot i_{IRLS} \cdot 2^k)$
Concise Splits via Model Selection	$O(Nq^2 \cdot i_{IRLS} \cdot \max\left\{2^k, i_{IC}\right\})$
Concise Splits via Penalized Likelihood	$O(Nq^2 \cdot i_{IRLS} \cdot \max\left\{2^k, i_{\lambda}\right\})$

Table 6.1: This table tabulates the complexity of partitioning a node with various types of splits. It is immediate that exponential growth in k is the limiting factor of the proposed approach to tree-growth.

to the best ideal split (we examine the viability of this computational shortcut in Section 6.4).

Table 6.1 summarizes the complexity of proposed approaches and compares it with axis-parallel splits. From this one sees that exponential growth in k is the limiting factor (quadratic growth in q and linear growth in N is manageable), i.e. the proposed approach to growing oblique trees is still applicable to large datasets though subject to observations coming from a small number of classes. When $k = 17$, at most 65535 $(= 2^{17-1} - 1)$ ideal splits need to be evaluated at each node. This increases to 131071, 262143 and 524287 when $k = 18, 19$ and 20 respectively which is large though computationally feasible. As many classification problems have observations that come from fewer than 10 classes (as noted in Section 1.1), the proposed approach to growing oblique trees is applicable to many problems.

6.4 Finding Concise Oblique Splits

When growing more interpretable oblique trees, concise oblique splits are found by applying variable selection techniques to the best ideal split. This computational shortcut was justified as follows,

1. the impurity of an ideal split is indicative of the linear separability of the ideal

outcome upon which it is based,

2. if an ideal split has a high value of impurity, it is unlikely that a concise oblique split based on the same ideal outcome will have a low value of impurity,

3. as such, concise oblique splits are found from the best ideal split as it has the lowest value of impurity.

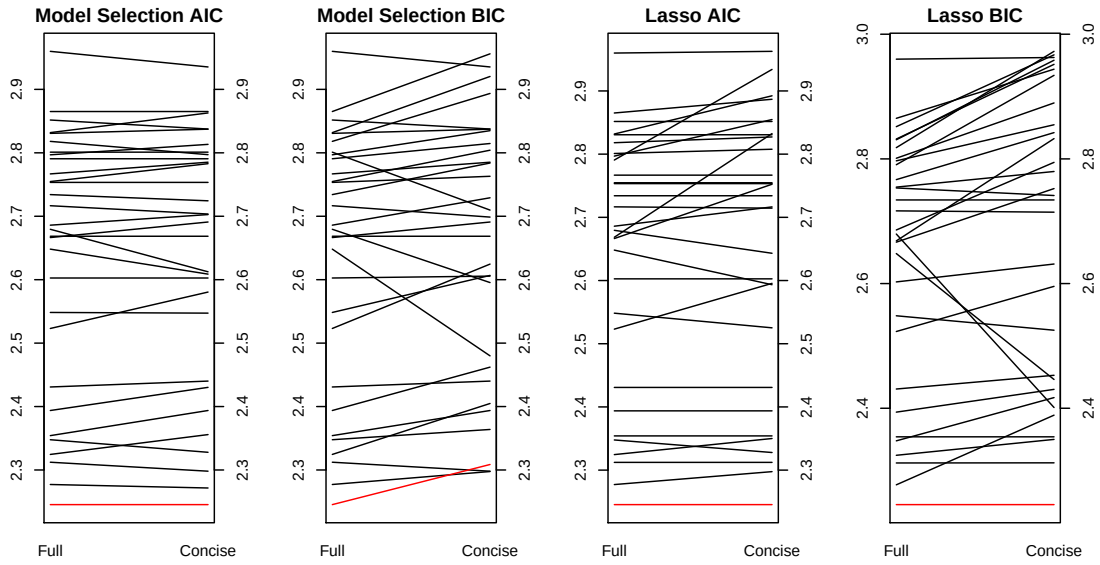
We consider how impurity of ideal splits change when variable selection techniques are applied to ascertain the validity of this claim. For each dataset considered,

1. the impurity of each ideal split is evaluated,

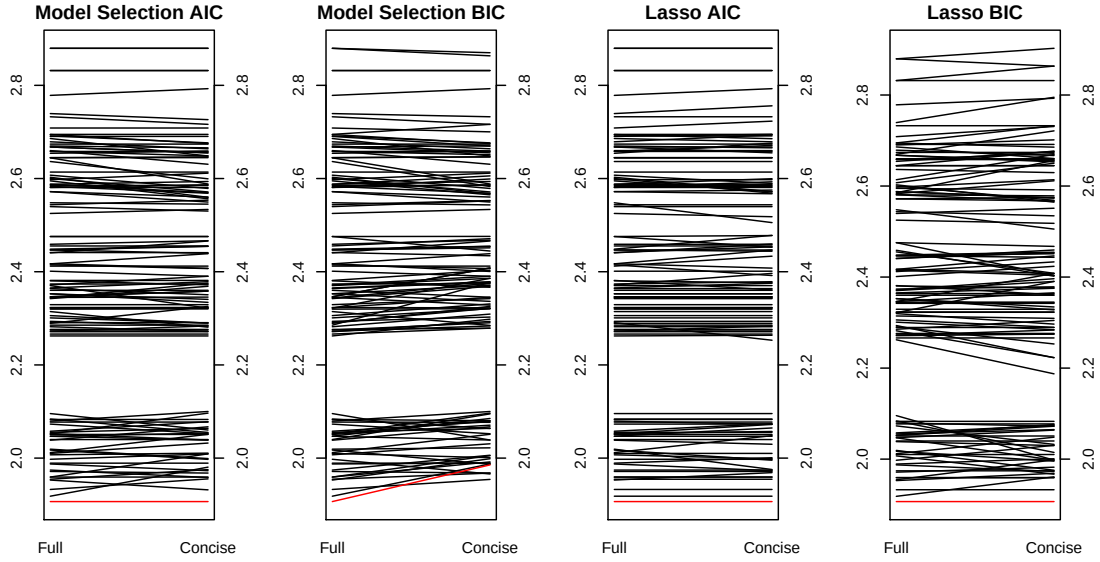
2. variable selection techniques are applied to find the associated concise oblique split from each split,

3. the impurity of each concise oblique split is evaluated.

Figure 6.4(a) shows how impurity changes when variable selection techniques are applied to ideal splits that partition the glass fragments dataset. Found in the *MASS* package in R, the glass fragments dataset consists of 9 measurements of 214 observations that come from 6 classes (so there are $31 = 2^{6-1} - 1$ ideal splits in all). Each subplot shows how impurity changes when different forms of variable selection are applied. Looking at each subplot, it is immediate that concise oblique splits found from the best ideal split (shown in red) still have low values of impurity regardless of which form of variable selection is applied. One also sees that the change in impurity is greater when variable selection is applied with BIC (whether via model selection or penalized likelihood) though the change is generally small. This is to be expected as the splits produced with BIC are more concise.



(a) Glass Fragments Dataset



(b) E. Coli Dataset

Figure 6.4: Figures 6.4(a) and 6.4(b) show how the impurity of ideal splits change when variable selection techniques are applied to ideal splits from the glass fragments and E. Coli datasets. The best ideal split is shown in red in each case. Looking at each plot, one sees that impurity generally does not change much (most lines are parallel to the x-axis) and that the change in impurity is greater when BIC is applied (be it with model selection or the Lasso). Though this may be the case, concise oblique splits found from the best ideal split still have low values of impurity. One also sees banded structure in attained values of impurity, this confirms the observation that some ideal outcomes are easier to replicate than others.

In the case of the E. Coli dataset, there are $127 = 2^{8-1} - 1$ ideal splits as observations come from 8 classes. Figure 6.4(b) shows how impurity changes. As with the glass fragments dataset concise oblique splits found from the best ideal split still have low values of impurity and the change is generally greater when variable selection is applied with BIC. With more ideal splits, banded structure of attained values of impurity before variable selection is applied can clearly be seen which confirms the observation that some compositions of ideal outcomes are more easier replicable than others.

The last example considered is over the yeast dataset. It consists of 8 measurements of 1484 yeast protein sequences from 10 classes (so there are $511 = 2^{10-1} - 1$ ideal splits). Figure 6.5 shows how impurity changes. We again note that concise oblique splits found from the best ideal split still have low values of impurity. In addition to this banded structure is present and impurity generally does not change much.

Having considered how variable selection affects the impurity of ideal splits over various datasets, one sees that impurity generally does not change much. This supports the claim that concise oblique splits found from ideal splits that have high values of impurities tend not to have low values of impurity, i.e. it is sufficient to focus on ideal splits with low values of impurity to find concise oblique splits with low values of impurity.

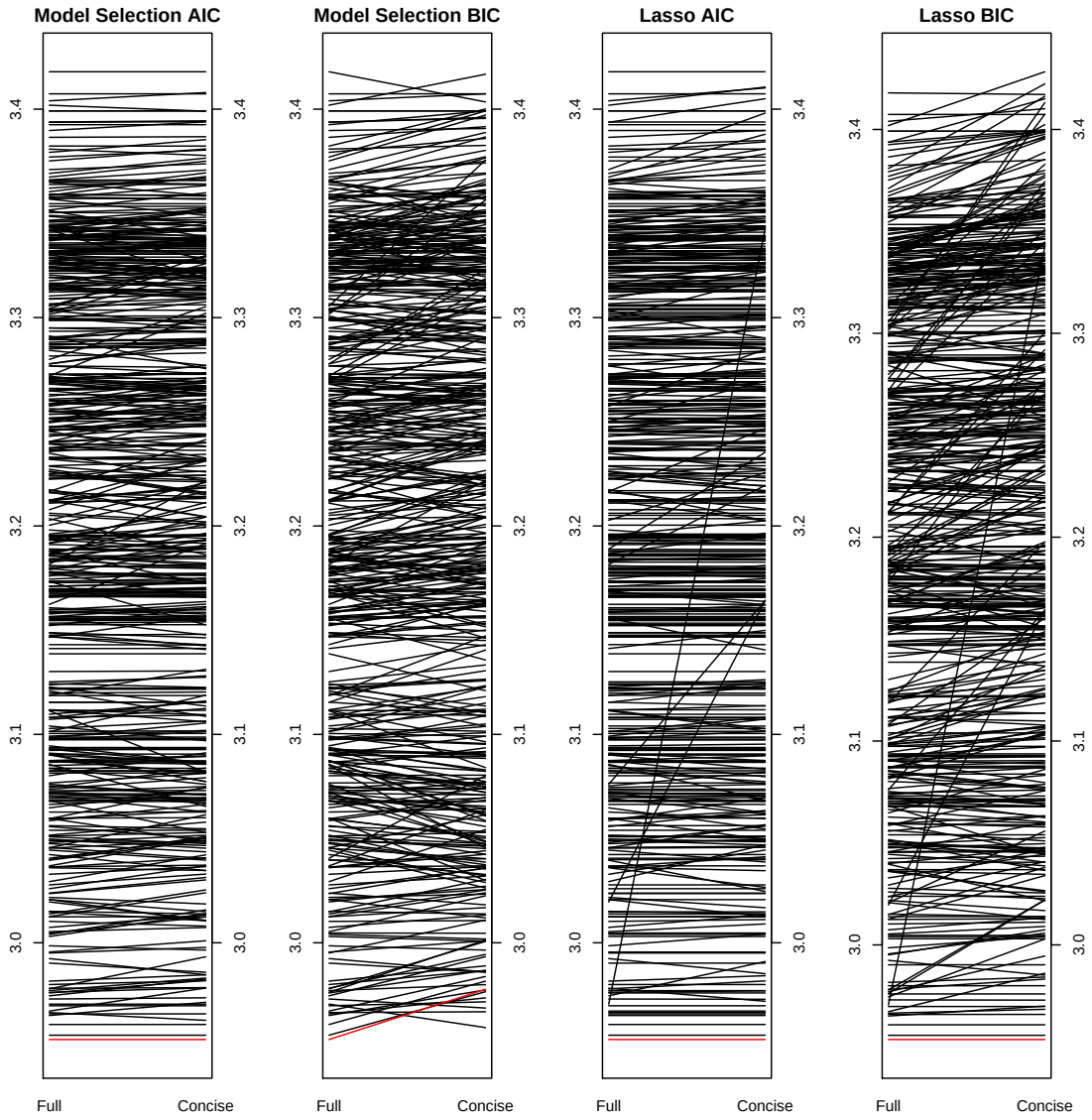


Figure 6.5: The information shown here is the same as that in Figures 6.4(a) and 6.4(b) but with the yeast dataset. One sees that concise splits found from the best ideal split still have low values of impurity.

Chapter 7

Conclusion

This thesis focuses on two important and practical aspects of oblique trees: growing them quickly and making them more interpretable.

By examining how oblique splits with low values of impurity are found, a more directed approach to finding good oblique splits is proposed; oblique splits (which are simply linear decision boundaries) are found by fitting logistic regression classifiers to a family of two-class classification problems. Empirical evidence shows that doing so does in fact find good oblique splits (though not in the case where ideal outcomes simply cannot be partitioned well with linear decision boundaries) and that this approach is able to grow oblique trees quickly.

As for more interpretable oblique trees, they can be found in one of two ways. Trees can be grown with concise oblique splits during tree-growth found by applying well-founded variable selection techniques to ideal splits. In addition to this, cost-complexity pruning ideas can also be used; oblique trees can be pruned (using more representative costs for rooted subtrees) and trimmed (to force its splits to become

more concise). It has been empirically confirmed that these approaches produce more interpretable oblique trees.

As this thesis wishes to contribute to the widespread use of oblique trees, a publicly available implementation of these ideas (documented in the Appendix) is also provided. Together with this R package, it is hoped that this work satisfactorily addresses the shortcomings of existing research. Though this may be the case, several aspects were omitted from this thesis or could be improved upon,

Computational Complexity: the number of ideal splits considered at each node increases exponentially with the number of residual classes so the approach scales poorly with the number of classes of the observations in a dataset (where it is small however, oblique trees can be grown quickly). If intelligent ways of further restricting the number of ideal splits that need to be considered at each node are proposed, this approach would better scale.

Implementation in R: as specialist techniques from statistics are used (namely the fitting of penalized and unpenalized binomial logistic regression models, the automatic selection of submodels with low values of AIC/BIC and the identification of transition points of λ from L1 penalized logistic regression models), it is difficult to implement this approach in languages other than R (as code for such techniques is readily available). Reimplemention in a fast programming language would produce great increases in speed through computational efficiencies.

Comparison with Existing Methods: a comparison of the time taken to grow oblique trees with the proposed approach with existing methods would be interesting. This is difficult however as easily accessible implementations of

existing methods are not readily available.

Regression Trees: the proposed approach to growing oblique trees makes use of prior information from the classes of observations to find ideal splits at each stage of tree-growth. Though no direct equivalent exists with regression problems, these ideas can surely be emulated (either with clustering methods or the EM algorithm say) so that regression trees can be grown with oblique splits.

Addressing these issues would have gone beyond the scope of this thesis. Other than the first point, these issues are implementational and are not relevant to the development of the theoretical framework by which more interpretable oblique trees are produced. If the proposed approach catches on, they points would surely deserve further investigation.

Appendix A

Documentation for R package *oblique.tree*

The research presented in this thesis was carried out with an R package called *oblique.tree*. As the proposed approach to growing oblique trees requires several specialist statistical techniques to be used, much of the code is implemented in R as software for these techniques are readily available. *oblique.tree* allows more interpretable oblique trees to be grown as documented in Chapters 4 and 5.

`oblique.tree()` is the main function. It allows trees that consider full oblique splits, concise oblique splits or axis-parallel splits to be grown (as well as mixture trees). Predictions from oblique trees are obtained using `predict.oblique.tree()`.

Documentation for exported functions of *oblique.tree* follow.

`oblique.tree` *Fit an Oblique Tree to Classification Data*

Description

An oblique tree is grown by binary recursive partitioning using the response in the specified formula with oblique splits composed of linear combinations of terms from the right-hand-side.

Usage

```
oblique.tree(
        formula,
        data,
        subset,
        control = tree.control(nobs, ...),
        method = "recursive.partition",
        split.impurity = c("deviance", "gini"),
        model = FALSE,
        oblique.splits = c("only", "on", "off"),
        variable.selection = c( "none",
                                "model.selection.aic",
                                "model.selection.bic",
                                "lasso.aic",
                                "lasso.bic"),
        ...)
```

Arguments

`formula`	A formula expression. The left-hand-side (response) should be a factor. The right-hand-side should be a series of numeric or factor variables separated by `+` (there should be no interaction terms). Both `.` and `-` are allowed.
`data`	A data frame in which to interpret `formula` and `subset`.
`subset`	An expression specifying the subset of cases to be used.
`control`	A list as returned by `tree.control`.
`method`	A character string specifying the method to use. The only other useful value is `"model.frame"`.
`split.impurity`	Splitting criterion to use.

`model` A model frame containing a response and predictors that can be used in place of `formula`, `data` and `subset` to allow direct specification of the problem.

`oblique.splits`
If and how oblique splits should be used during tree-growth. `only` grows trees that only consider oblique splits, `on` grows those that consider oblique and axis-parallel splits simultaneously and `off` grows trees that only consider axis-parallel splits.

`variable.selection`
If and how concise oblique splits should be found during tree-growth. `none` grows oblique trees using full oblique splits, `model.selection.aic` performs variable selection with AIC from the full model upon the best ideal split, `model.selection.bic` similarly for BIC, `lasso.aic` applies L1 regularization from the full model and chooses the penalization parameter with AIC and `lasso.bic` similarly for BIC.

`...` Additional arguments that are passed to `tree.control`. Normally used for `mincut`, `minsize` or `mindev`.

Details

An oblique tree is grown by binary recursive partitioning using the response in the specified formula and by choosing splits composed of terms from the right-hand-side. Where categorical attributes are considered levels of unordered factors are divided into two non-empty groups, where axis-parallel splits are considered numeric variables are divided into $X < a$ and $X \geq a$ and where oblique splits are considered numeric variables are divided into $\sum aX < c$ and $\sum aX \geq c$. The split that maximizes the reduction in impurity is chosen and the process repeated. Splitting continues until the terminal nodes are either pure, sufficiently pure or too small to be split.

When growing oblique trees, $2^{R-1} - 1$ logistic regression problems need to be (where R is the number of residual classes at a node). If observations come from more than 20 classes this approach will be slow.

Value

An object of class `c("oblique.tree","tree")` is returned with components

`frame` A data frame with a row for each node and `row.names` giving the node numbers. The columns include `var`, the variable used to perform each split (where "" denotes an oblique split and `"<leaf>"` a terminal node), `n`, the number of cases reaching that node, `dev` the deviance of the node, `yval`, the class associated to

that node, `split`, a two-column matrix of the labels for left and right splits at the node and `yprob`, a matrix of fitted probabilities for each response level.

`where` A vector indicating the row number of the frame detailing the node to which each case is assigned.

`terms` The terms of the formula.

`call` The matched call to `oblique.tree`.

`y` Predicted classes of each observation by the tree (this differs from the implementation in the **tree** package where `y` is instead used to denote the actual classes).

A tree with no splits is of class `"singlenode"` which inherits from class `"tree"`.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge. Chapter 7.

See Also

`tree.control` in the **tree** package, `predict.oblique.tree`, `prune.oblique.tree`, `trim.oblique.tree`

Examples

```
#create the augmented crabs dataset
data(crabs, package = "MASS")
aug.crabs.data <- data.frame(   g=factor(rep(1:4,each=50)),
                                predict(princomp(crabs[,4:8]))[,2:3])

plot(   aug.crabs.data[,-1],type="n")
text(   aug.crabs.data[,-1],
        col=as.numeric(aug.crabs.data[,1]),
        labels=as.numeric(aug.crabs.data[,1]))

#grow a full oblique tree
ob.tree <- oblique.tree(formula         = g~.,
```

```
                        data            = aug.crabs.data,
                        oblique.splits  = "only")
plot(ob.tree);text(ob.tree)
```

`predict.oblique.tree` *Predictions from Fitted Oblique Tree Object*

Description

Returns predictions from a fitted `oblique.tree` object.

Usage

```
predict.oblique.tree(
        object,
        newdata,
        type = c("vector", "tree", "class", "where"),
        eps = 1e-3,
        update.tree.predictions = FALSE,
        ...)
```

Arguments

`object`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`newdata`	Data frame containing the values at which predictions are required. The predictors referred to in the right side of `formula(object)` must be present by name in `newdata`. If missing, fitted values are returned.
`type`	Character string denoting how predictions are to be returned, i.e. class probabilities (default), a tree object, class predictions or predictions to leaf nodes.
`eps`	A lower bound for the probabilities, used if events of predicted probability zero occur in `newdata` when predicting a tree.
`update.tree.predictions`	Logical vector denoting whether tree predictions (`frame$yval, frame$yprob, $where, $y` etc) are updated when `newdata` is provided.
`...`	Further arguments passed to or from other methods.

Details

This function is a method for the generic function `predict()` for objects of class `c("oblique.tree","tree")`. It can be invoked by calling `predict(x)` for an object `x` of the appropriate class or directly by calling `predict.oblique.tree(x)` regardless of the class of the object.

Value

If `type = "vector"`: a matrix of predicted class probabilities is returned. This object is obtained by dropping observations down `object`.

If `type = "tree"`: an object of class `c("oblique.tree","tree")` is returned with new values for `frame$n` and `frame$dev`.

If `type = "class"`: a factor of the predicted classes (that with highest posterior probability, with ties split randomly).

If `type = "where"`: the nodes the cases reach.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge. Chapter 7.

See Also

`predict`, `oblique.tree`.

Examples

```
#grow an oblique tree to the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula         = type~.,
                        data            = Pima.tr,
                        oblique.splits  = "on")
plot(ob.tree);text(ob.tree);title(main="Oblique Tree")

#predictions to in-sample data
#class probabilities for each observation
predict(ob.tree,type="vector")
```

```
#the tree itself
predict(ob.tree,type="tree")
#class predictions for each observation
predict(ob.tree,type="class")
#the leaf where each observation falls
predict(ob.tree,type="where")

#predictions to out-of-sample data
data(Pima.te, package = "MASS")
#class probabilities for each observation
predict(ob.tree,newdata=Pima.te,type="vector")
#the tree itself
predict(ob.tree,newdata=Pima.te,type="tree")
#class predictions for each observation
predict(ob.tree,newdata=Pima.te,type="class")
#the leaf where each observation falls
predict(ob.tree,newdata=Pima.te,type="where")
```

`prune.oblique.tree` *Cost-complexity Pruning of Oblique Tree Object*

Description

Determines a nested sequence of subtrees of the supplied tree by recursively "snipping" off the least important splits.

Usage

```
prune.oblique.tree(
        tree,
        k = NULL,
        newdata,
        prune.impurity = c("deviance", "misclass"),
        penalty = c("complexity", "size"),
        eps = 1e-3)
```

Arguments

`tree`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`k`	Cost-complexity parameter defining either a specific subtree of `tree` (`k` a scalar) or the (optional) sequence of subtrees minimizing the cost-complexity measure (`k` a vector). If missing, `k` is determined algorithmically.
`newdata`	Data frame upon which the sequence of cost-complexity subtrees is evaluated. If missing, the data used to grow the tree is used.
`prune.impurity`	Character string denoting the measure of node heterogeneity used to guide cost-complexity pruning. The default is `deviance` and the alternative is `misclass` (number of misclassifications or total loss).
`penalty`	Character string denoting the measure of tree complexity used to guide cost-complexity pruning. The default is `complexity` (that considers the complexity of splits used throughout the tree) and the alternative is `size` (that counts the number of leaves of the tree).

`eps` A lower bound for the probabilities, used to compute deviances if events of predicted probability zero occur in `newdata`.

Details

Determines a nested sequence of subtrees of the supplied tree by recursively "snipping" off the least important splits, based upon the cost-complexity measure.

If `k` is supplied, the optimal subtree for that value is returned.

The response as well as the predictors referred to in the right side of the formula in `tree` must be present by name in `newdata`. These data are dropped down each tree in the tree sequence and deviances or losses calculated by comparing the supplied response to the prediction. A `plot` method exists for objects of this class. It displays the value of the deviance, the number of misclassifications or the total loss for each subtree in the tree sequence. An additional axis displays the values of the cost-complexity parameter at each subtree.

Value

If `k` is supplied and is a scalar, an object of class `c("oblique.tree","tree")` is returned that minimizes the cost-complexity measure for that `k`. If `k` is a vector, an object of class `tree.sequence` is returned. The object contains the following components:

`size` The complexity of each tree in the cost-complexity pruning sequence.

`deviance` Total deviance of each tree in the cost-complexity pruning sequence.

`k` The value of the cost-complexity pruning parameter of each tree in the sequence.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge. Chapter 7.

See Also

oblique.tree.

Examples

```
#grow an mixture tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula         = type~.,
                        data            = Pima.tr,
                        oblique.splits  = "on")
plot(ob.tree);text(ob.tree);title(main="Mixture Tree")

#examine the tree sequence
tree.seq <- prune.oblique.tree( tree    = ob.tree)
print(tree.seq);plot(tree.seq)

#examine test error over the tree sequence
data(Pima.te, package = "MASS")
tree.seq <- prune.oblique.tree( tree    = ob.tree,
                                newdata = Pima.te)
print(tree.seq);plot(tree.seq)

#deviance is least when k = 8.148267
pruned <- prune.oblique.tree(   tree    = ob.tree,
                                k       = 9)
plot(pruned);text(pruned);title(main="Pruned Tree")
```

`trim.oblique.tree` *Trims Oblique Splits of Fitted Oblique Tree Objects*

Description

Determines a sequence of concise subtrees of the supplied tree by recursively "trimming" off the least important attributes used in oblique splits.

Usage

```
trim.oblique.tree(
        tree,
        best = NULL,
        newdata,
        trim.impurity = c("deviance", "misclass"),
        trim.depth = c("partial", "complete"),
        eps = 1e-3)
```

Arguments

`tree`	Fitted model object of class `oblique.tree`. This is assumed to be the result of some function that produces an object with the same named components as that returned by `oblique.tree`.
`best`	Requests the complexity (i.e. 1 + number of attributes used throughout the tree) of the concise subtree of `tree` to return (`best` a scalar) or a (optional) sequence of concise subtrees (`best` a vector). If missing, `best` is determined algorithmically. If there is no tree in the sequence of the requested size, the next largest is returned.
`newdata`	Data frame upon which the sequence of cost-complexity subtrees is evaluated. If missing, the data used to grow the tree is used.
`trim.impurity`	Character string denoting the measure of node heterogeneity used to guide tree trimming. The default is `deviance` and the alternative is `misclass` (number of misclassifications or total loss).
`trim.depth`	A character string denoting if oblique splits should be trimmed towards axis-parallel splits `partial` or to the constant predictor `complete`.
`eps`	A lower bound for the probabilities, used to compute deviances if events of predicted probability zero occur in `newdata`.

Details

Determines a sequence of concise subtrees of the supplied tree by recursively "trimming" its splits, based upon the cost-complexity measure.

If `best` is supplied, the optimal subtree for that value is returned.

The response as well as the predictors referred to in the right side of the formula in `tree` must be present by name in `newdata`. These data are dropped down each tree in the trim sequence and deviances or losses calculated by comparing the supplied response to the prediction. A `plot` method exists for objects of this class. It displays the value of the deviance, the number of misclassifications or the total loss for each subtree in the trim sequence. An additional axis displays the values of the cost-complexity parameter at each subtree.

Value

If `best` is a scalar, a `c("oblique.tree","tree")` object of size `best` is returned. Otherwise an object of class `c("trim", "trim.sequence")` is returned. The object contains the following components:

`comp`	The complexity of each tree in the cost-complexity pruning sequence.
`dev`	Total deviance of each tree in the cost-complexity pruning sequence.
`h`	The value of the cost-complexity pruning parameter of each tree in the sequence.

Author(s)

A. Truong

References

Truong. A (2009) *Fast Growing and Interpretable Oblique Trees via Probabilistic Models*

See Also

`oblique.tree`.

Examples

```
#grow a tree on the Pima Indian dataset
data(Pima.tr, package = "MASS")
ob.tree <- oblique.tree(formula                 = type~.,
```

```
                        data                    = Pima.tr,
                        oblique.splits          = "only")
plot(ob.tree);text(ob.tree);title(main="Full Oblique Tree")

#partially trimming
#examine the tree sequence
trim.seq <- trim.oblique.tree(  tree            = ob.tree)
print(trim.seq);plot(trim.seq)

#examine test error over the trim sequence
data(Pima.te, package = "MASS")
trim.seq <- trim.oblique.tree(  tree            = ob.tree,
                                newdata         = Pima.te)
print(trim.seq);plot(trim.seq)

#deviance is least when best = 7
p.trimmed <- trim.oblique.tree( tree            = ob.tree,
                                best            = 7)
plot(p.trimmed);text(p.trimmed);title(main="Partially Trimmed Tree")

#complete trimming
#examine the tree sequence
trim.seq <- trim.oblique.tree(  tree            = ob.tree,
                                trim.depth      = "complete")
print(trim.seq);plot(trim.seq)

#examine test error over the trim sequence
data(Pima.te, package = "MASS")
trim.seq <- trim.oblique.tree(  tree            = ob.tree,
                                trim.depth      = "complete",
                                newdata = Pima.te)
print(trim.seq);plot(trim.seq)

#deviance is least when best = 9
c.trimmed <- trim.oblique.tree( tree            = ob.tree,
                                best            = 9)
plot(c.trimmed);text(c.trimmed);title(main="Completely Trimmed Tree")
```

Bibliography

J. Aitchison and I. R Dunsmore. *Statistical Prediction Analysis*. Cambridge University Press, 1975. ISBN 0521206928.

H. Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Cáski, editors, *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akademiai Kaidó. Reprinted in *Breakthroughs in Statistics*, eds S. Kotz and N. L. Johnson (1992), volume I, pp. 599–624. New York: Springer.

H. Akaike. A new look at statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

A. Albert and J. A. Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1–10, 1984.

A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

B. Axelrod, S. Campos, and J. Envarli. Perceptron-Based Oblique Trees (P-BOT). Master's thesis, Georgia Institute of Technology, 2005. URL `http://www-static.cc.gatech.edu/grads/b/baxelrod/projects/PBOT.pdf`.

K. P. Bennett. *Machine learning via mathematical programming*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 1993.

K. P. Bennett and J. Blue. A support vector machine approach to decision trees. Technical report, Rensselaer Polytechnic Institute, Troy, NY 12180, 1997. URL `citeseer.ist.psu.edu/bennett97support.html`. Department of Mathematical Sciences Math Report No. 97-100.

K. P. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41(3):295–313, 2000.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks/Cole, Monterey, CA, 1984.

C. E. Brodley and P. E. Utgoff. Multivariate decision trees. Technical Report UM-CS-1992-083, University of Massachusetts, December 1992a. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.7328`.

C. E. Brodley and P. E. Utgoff. Multivariate versus univariate decision trees. Technical Report UM-CS-1992-008, University of Massachusetts, January 1992b. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.5989`.

D. E. Brown, C. L. Pittard, and H. Park. Classification trees with optimal multivariate decision nodes. *Pattern Recognition Letters*, 17(7):699–703, June 1996. ISSN 0167-8655.

N. A. Campbell and R. J. Mahon. A multivariate study of variation in two species of rock crab of genus *Leptograpsus*. *Australian Journal of Zoology*, 22:417–425, 1974.

E. Cantu-Paz and C. Kamath. Using evolutionary algorithms to induce oblique decision trees. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 1053–1060, Las Vegas, Nevada, USA, 2000. Morgan Kaufmann.

E. Cantu-Paz and C. Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7:54–68, February 2003.

V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Optimization Theory and Applications*, 45:41–51, 1985.

S. Le Cessie and J. C. Van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.

C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, 1965.

B.A. Draper, C.E. Brodley, and P.E. Utgoff. Goal-directed classification using linear machine decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):888–893, September 1994.

M. Frean. A "thermal" perceptron learning rule. *Neural Computation*, 4(6):946–957, November 1992.

P. Geurts. *Contributions to decision tree induction: bias/variance tradeoff and time series classification*. PhD thesis, University of Liège, Belgium, May 2002. URL `http://www.montefiore.ulg.ac.be/services/stochastic/pubs/2002/Geu02`.

T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman and Hall, London., 1990.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001. ISBN 0387952845.

T. Hastie, J. Friedman, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent, July 2008. URL `http://www-stat.stanford.edu/~hastie/Papers/glmnet.pdf`. As yet unpublished.

D. G. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(2):1–32, 1993.

E. G. Henrichon, Jr. and K.-S. Fu. A nonparametric partitioning procedure for pattern classification. *IEEE Transactions on Computers*, 18:614–624, 1969.

A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970.

J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-58111-6.

P. Horton and K. Nakai. A probabilistic classification system for predicting the cellular localization sites of proteins. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, pages 109–115, 1996.

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

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

W. Y. Loh and N. Vanichsetakul. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83(403): 715–728, 1988. ISSN 0162-1459.

P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall, 2nd edition, 1989.

J. E. Moody. Note on generalization, regularization and architecture selection in nonlinear learning systems. In *First IEEE-SP Workshop on Neural Networks in Signal Processing*, pages 1–10, Los Alamitos, CA, 1991. IEEE Computer Society Press.

J. E. Moody. The *effective* number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4. Proceedings of the 1991 Conference*, pages 847–854, San Mateo, CA, 1992. Morgan Kaufmann.

J. N. Morgan and R. C. Messenger. *THAID*: a sequential search program for the analysis of nominal scale dependent variables. Survey Research Center, Institute for Social Research, University of Michigan, 1973.

N. Murata, S. Yoshizawa, and S. Amari. A criterion for determining the number of parameters in an artificial neural network model. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks. Proceedings of ICANN-91*, volume I, pages 9–14, Amsterdam, 1991. North Holland.

N. Murata, S. Yoshizawa, and S. Amari. Learning curves, model selection and complexity of neural networks. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5. Proceedings of the 1992 Conference*, pages 607–614, San Mateo, CA, 1993. Morgan Kaufmann.

N. Murata, S. Yoshizawa, and S. Amari. Network information criterion—determining the number of hidden units for artificial neural network models. *IEEE Transactions on Neural Networks*, 5:865–872, 1994.

S. K. Murthy. *On Growing Better Decision Trees from Data*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, 1995. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.9426`.

S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998.

S. K. Murthy, S. Kasif, S. Salzberg, and Beigel. OC1: A randomized induction of oblique decision trees. In *National Conference on Artificial Intelligence*, pages 322–327, 1993. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.6068`.

S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

M. Y. Park and T. Hastie. Penalized logistic regression for detecting gene interactions, 2006. URL `www-stat.stanford.edu/~hastie/Papers/plr2.bs.pdf`.

M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society, Series B*, 69:659–677(19), September 2007.

J. R. Quinlan. Discovering rules by induction from large collections of examples. In D. Michie, editor, *Expert Systems in the Micro-Electronic Age*, pages 168–201. Edinburgh University Press, Edinburgh, 1979.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

J. R. Quinlan. Decision trees and multi-valued attributes. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11*, pages 305–318, Oxford, 1988. Clarendon Press.

J. R. Quinlan. Decision trees and decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 20:339–346, 1990.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996. ISBN 0-521-46086-7.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, November 1958.

T. J. Santner and D. E. Duffy. *The Statistical Analysis of Discrete Data*. Springer-Verlag, New York, 1989. ISBN 0-387-97018-5.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978.

I. K. Sethi and G. P. R. Sarvarayudu. Hierarchical classifier design using mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:441–445, 1982.

J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In R. A. Greenes, editor, *Proceedings of the Symposium on Computer Applications in Medical Care (Washington, 1988)*, pages 261–265, Los Alamitos, CA, 1988. IEEE Computer Society Press.

C. Stein. Estimation of the mean of a multivariate normal distribution. *Annals of Statistics*, 9(6):1135–1151, 1981.

P. J. Tan and D. L. Dowe. MML inference of large margin oblique decision trees, 2004a. URL `http://serv1.ist.psu.edu:8080/viewdoc/summary?doi=10.1.1.91.1483`.

P. J. Tan and D. L. Dowe. MML inference of oblique decision trees. In *Australian Conference on Artificial Intelligence*, pages 1082–1088, 2004b. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.113.6462`.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

A Truong. *oblique.tree: Oblique Trees for Classification Data*, 2009. URL `http://cran.r-project.org/web/packages/oblique.tree/index.html`.

P. E. Utgoff. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1(4):377–391, 1989.

P. E. Utgoff and C. E. Brodley. Linear Machine Decision Trees. Technical Report UM-CS-1991-010, University of Massachusetts, Amherst, MA, USA, 1991. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.616`.

H. Zou, T. Hastie, and R Tibshirani. On the "degrees of freedom" of the lasso. *Annals of Statistics*, 35(5):2173–2192, 2007.