Fast Growing and Concise Oblique Trees via Logistic Regression Models

Alfred Truong

Christ Church College

University of Oxford

A Thesis submitted for the degree of Doctor of Philosophy

Hilary Term 2009

Acknowledgements

Fast Growing and Concise Oblique Trees via Logistic Regression Models

Alfred Truong, Christ Church College

DPhil thesis, Department of Statistics

Hilary Term 2009

The key strength of classification trees is its interpretability which diminishes as trees grow larger. The reason why some trees are larger than others comes down to the interplay between how observations of different classes are distributed and the types of splits we use to partition them. Widely used methods of tree growth consider axis-parallel splits over continuous attributes ($X_i < c$). Though oblique splits ($\sum_i a_i X_i < c$) are better suited to partitioning observations, they are also much more computationally intensive to use. Though many have found ways of growing trees with oblique splits, these methods have not caught on. Possible reasons for this include the archaic approaches applied to finding oblique splits, poor interpretability of resulting oblique trees and the difficulty in obtaining widely accessible code. This thesis addresses each of these points by presenting a new approach to finding oblique splits that is intuitively appealing and easily extensible. By adapting tree-growth and tree-pruning techniques, more interpretable oblique trees can be obtained.

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 TRUONG

Signed:

Date:

Contents

Chapter 1

Introduction

1.1 Recent Trends in Statistics

The collection and storage of data has never been easier with data storage devices become ever larger and cheaper while data capture devices become more powerful and affordable. The computational power of computers needed to handle this surge in data has also greatly increased[1]. These phenomena have permeated through both scientific and business communities to produce large pools of specialist data. If tapped, they could potentially provide valuable information to their owners. Examples of this include data that is gathered, stored and processed as a result of satellite imagery, the human-genome project and the internet.

Applied statisticians have risen to the challenge to tackle data-related problems in areas like bioinformatics and optical character recognition among others. They have done so by marrying ideas from statistics with ever more powerful computers.

[1]Increases in computational power have so far grown according to Moore's Law (1965) (16) which roughly states that the number of transistors in integrated circuits doubles every 2 years. Computational power has increased by the order of a thousand since the early 1980s till now.

The result has been to greatly increase the range of problems that statistics can be used to solve.

Though much progress has already been made, the difficulty in obtaining implementations of these tools and in applying them to data cannot have help ideas from applied statistics reach their potential (proprietary software written by individuals and companies for profit being a contributing factor). There has however been movement in recent years towards providing widely accessible open-source software[2] to fill this gap.

Together, increases in computational power and greater sharing of code has made it worthwhile to revisit problems in applied statistics that were once considered too difficult.

1.2 Aim of this Thesis

The classification tree is a widely used classifier introduced in its current form by Breiman *et al* (3) in the 1980's. Its main strength is its interpretability. At the time of its conception, processing power of computers was a thousandth of that of current levels and so great consideration was given to ways of speeding up *tree growth*[3]. As such, a redimentary approach to partitioning observations was favoured over more computationally intensive (though better suited) approaches. Tests over continuous attributes were restricted to be of the form $X_i < c$ (often called *axis-parallel*

[2]R (18) is a free software environment for statistical computing which 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 the ease of its use.

[3]Classification trees are said to be *grown* (as well as trained) to data.

splits) instead of tests of the form $\sum_i a_i X_i < c$. As trees with more tests are less interpretable, it is desireable to grow them using these more general tests.

With ever increasing computing power, many have tried to do this and so tests of the form $\sum_i a_i X_i < c$ are known by many names including linear machines, linear-combination tests, multivariate tests and *oblique splits* (used in this thesis). Despite numerous ways of growing *oblique trees* (trees grown by considering oblique splits over continuous attributes), none have caught on. This thesis seeks to identify and address possible reasons for this and in doing so, hopes to allow oblique trees to become viable alternatives to *axis-parallel trees* (trees grown by considering axis-parallel splits over continuous attributes) that are still widely used today.

1.3 Organization of this Thesis

This thesis is organised as follows. Chapter 1 contains a brief overview of recent developments in information technology and its relation to developments in applied statistics. A description of the aim of this thesis follows along with an overview of this thesis.

Chapter 2 seeks to identify possible reasons as to why oblique trees are still shunned by users. Beginning with a recap of classification trees, this chapter moves on to suggest the use of oblique splits as a way of improving the interpretability of axis-parallel trees as well as analysing the inherent difficulty of finding such splits. A summary of issues that I believe have contributed to its lack of use follows.

Chapter 3 gives an in-depth survey of existing methods to growing oblique trees.

Existing work is separated into two categories as defined by the motivation behind each method. This gives a clearer understanding its nature and of the progression of the area.

Chapter 4 presents a new method for finding oblique splits. It follows with a heuristic argument to further back the methodology used.

Chapter 5 present ways of obtaining oblique trees that are interpretable. Methods are separating into two categories, those applied during tree growth and those applied post-tree growth.

An analysis of techniques presented in this thesis follows in Chapter 6 giving an assessment of their limitations.

Chapter 7 finishes with closing remarks for this thesis.

Chapter 2

Background

Before delving into reasons why I believe existing attempts to growing oblique trees have not caught on, it is instructive to review various aspects of tree growth. Section 2.1 begins by revisiting the basic idea of a classification tree and Section 2.2 continues to cover issues that affect the interpretability of a tree. To understand why oblique trees are so difficult to grow, Section 2.3 reviews how tests are chosen during tree growth and its implications to the growth of oblique trees. A summary of issues that I believe have contributed to the lack of use of existing methods follow in Section 2.4.

2.1 Classification Trees

Used by researchers in botany, computer science and statistics among other areas, the classification tree is a generic idea for the partition and classification of observations. Given classification data of the form $(C, \mathbf{X}) = (C, X_1, \ldots, X_p)$, a classification tree is grown to a training set by recursively partitioning observations into smaller subsets where observations have more similar classes. This process is

Figure 2.1: Example Classification Tree and its Associated Decision Boundaries

naturally representable as a tree with

- *nodes* were tests partition observations,
- *branches* from nodes representing outcomes of these tests and
- *leaves* at the bottom[1] of the tree.

When classification trees are used for prediction, observations are *dropped down the tree* and classified by the class associated to the leaf in which it falls.

The purpose of a classification tree and indeed its key strength is interpretability. Figure 2.1 gives an example to illustrate this and shows the associated decision boundaries that the tree defines. This example shows the sitation where both attributes are continuous ($p = 2$, $\mathbf{X} = \mathbb{R}^2$) and observations come from one of four *classes* ($C \in \{1, \ldots, 4\}$). Other than our preference for interpretable trees, there is no reason why tests cannot be arbitrarily complex and why they cannot partition observations into more than two *child nodes*.

Widely used trees introduced by Breiman *et al* (3) however, only allow univariate binary splits. Tests are univariate in the sense that data from only one attribute is required to form each test and binary in the sense that they produce exactly two outcomes. This ensures tests are interpretable whilst also simplifying the problem. The exact form of these tests depend on the type of data that each attribute measures. They are as follows,

X_i **continuous:** tests constrained to be $X_i < c$ v.s. $X_i \geq c$ for some constant $c \in \mathbb{R}$

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

Figure 2.2: An interpretable tree compared to a less interpretable tree

X_i **categoric:** tests constrained to be $X_i \in A$ v.s. $X_i \notin A$ for some subset A of $\{1, \ldots, |X_i|\}$ (the levels of categoric attribute X_i)

An understanding of tree growth and tree pruning techniques introduced by Breiman *et al* in *Classification and Regression Trees* (3) has been conveniently summarized in the Appendix and is assumed.

2.2 Tree Interpretability

Though univariate binary splits are highly interpretable in isolation, this does not guarantee that the *final tree* (the fully grown tree) is equally interpretable. Trees with more tests are less interpretable as Figure 2.2 shows. The reason as to why some trees require more tests than others comes down to the interplay between the way observations of different classes are distributed and the type of tests used to partition them; trees recursively partition observations into smaller subsets and will do so until they are sufficiently pure with whatever tool it is provided with. As interpretability is a key strength of classification trees, finding ways of growing *smaller trees* (trees that use fewer tests) is desireable.

Growing smaller trees is easy to do in theory and can be achieved by simply using tests that are better able to partition observations during tree growth. A simple extension of axis-parallel splits $X_i < c$ gives oblique splits $\sum_{i=1}^{q} a_i X_i < c$ (assuming all continuous attributes lie in the first q positions of feature vectors) and as Figure 2.3 shows, are better able to partition observations in a non-axis-parallel fashion.

Figure 2.3: Many axis-parallel splits required to approximate a single oblique split

As linear functions, oblique splits are interpretable and one would think that this translates to oblique trees that are also interpretable. Unfortunately, interpretability of oblique trees is a delicate matter. To understand why, consider the following. *Full oblique splits* (oblique splits that use inputs from every continuous attribute) are less interpretable than those that use fewer attributes, e.g. $X_1 > 0.25X_2$ is *more concise* than $X_1 > 0.25X_2 + 0.5X_3 + X_4$. Though splits that perfectly partition observations are nice (a full oblique split say), an axis-parallel split that partitions several observations incorrectly is preferable as it is much easier to understand. This problem becomes more acute with greater numbers of continuous attributes.

In summary, smaller axis-parallel trees are more interpretable and there are cases where oblique splits can help to achieve this. In order to obtain oblique trees that are interpretable however, splits should be as concise as possible.

2.3 Finding Oblique Splits

Putting aside the issue of how one finds concise oblique splits for the meantime, it is instructive to examine the computational problem posed when using oblique splits to partition observations.

In widely used tree growth techniques, one would ideally use the *best* test (the test that gives the lowest impurity value) at each node which implies that every possible test must be evaluated. The magnitude of this problem is conveniently

Axis-Parallel Split Dictionary Oblique Split Dictionary

Figure 2.4: Examples of split dictionaries for 4 observations with 2 continuous attributes

summarized by considering *split dictionaries* (for a particular family of splits).

A split dictionary for a family of splits is defined to be the smallest set of splits that produces every possible partitioning of observations in the training set permitted by that family of splits. Figure 2.4 illustrates this concept by showing split dictionaries for axis-parallel and oblique splits on the same dataset. Though split dictionaries are not unique (small perturbations of these splits give new split dictionaries), its size however remain constant.

Thinking in terms of split dictionaries is useful as its size exactly quantifies the number of splits that must be evaluated to find the best split. By enumerating the size of axis-parallel and oblique split dictionaries for a generic dataset, it is possible to directly compare the magnitude of the computational tasks posed when finding the best axis-parallel and oblique split.

Starting with axis-parallel splits, though there are uncountably many splits of the form $X_i = c$ (as $c \in B \subset \mathbb{R}$), only a finite number of unique outcomes can ever occur over the training set. If there are L_i unique values of X_i, there are exactly $L_i - 1$ unique outcomes and so the size of the axis-parallel split dictionary for X_i is $L_i - 1$. With q continuous attributes, the size of the axis-parallel split dictionary is exactly $\sum_{i=1}^{q}(L_i - 1)$. Assuming each L_i is similar to N (the number of observations), the axis-parallel split dictionary grows linearly in both N and q. Growing axis-parallel trees is straightforward.

The size of the oblique split dictionary is $\left\{2\sum_{k=0}^{q}\binom{L_{gen}-1}{k}-2\right\}/2$ where L_{gen} is the number of *generally positioned*[2] (q-dimensional) observations. It is given by examining a combinatorial geometry result[3] by Cover (1965) (10). L_{gen} is similar to N (the only way for it not to be is when all q-dimensional observations lie in a space of much lower dimensionality) and so the size of the oblique split dictionary grows super-linearly in both N and q.

As classification datasets are often large, growing oblique trees that use the best test at each node is impractical. To avoid this computational explosion, existing methods of oblique tree growth choose to either focus on a subset of the split dictionary or to apply stochastic searches which are forced to terminate. Existing methods are covered in greater detail in Chapter 3.

2.4 Why Have Oblique Trees Not Caught On

Having consider various issues relating to tree growth, it is easy to see that oblique trees are difficult to grow. Despite several methods of growing oblique trees that have since arisen, none have caught on. I have identified a variety of issues which I feel has contributed to this which are given below.

Intellectual Appeal of the Method: Existing attempts to growing oblique trees

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

[3]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 the hyperplane $\{x : w \cdot x = 0\}$ they fall. Cover (1965) (10) extended this result to enumerate the number of ways there are to partition points with hyperplanes of the form $\{x : w \cdot \phi(x) = 0\}$ for arbitrary functions ϕ. Choosing $\phi(x) = (1, x)$ in particular shows there are exactly $2\sum_{k=0}^{q}\binom{L_{gen}-1}{k}$ ways to partition points into two bins across hyperplanes with arbitrary intercept. This includes the trivial cases where all points are assigned to the same bin and also double counts across both bins.

use archaic approaches to find oblique splits. As simple as it sounds, I feel that this does not give confidence to users in the suitability of these methods.

Multiplicity of Trees: In order to avoid the computational problems noted, some methods apply stochastic searches to find oblique splits. In doing so, a different tree is grown each time for which no well-defined approach to choosing the "final" classifier is given. This again does not give confidence to users.

Interpretability of Oblique Tree: Most methods simply grow oblique trees with full oblique splits. Though such trees are smaller than axis-parallel trees, they are not interpretable and so they have little to differentiate themselves with other classifiers.

Easily Accessible Code: It is difficult to get hold of and to apply implementations of these oblique trees to real data. This has added another unnecessary barrier to their use.

This thesis seeks to address these shortcomings.

Chapter 3

Existing Work

Before presenting an approach to growing more interpretable oblique trees, this chapter reviews existing methods of growing oblique trees. As noted in Section 2.3, it is infeasible to find the best oblique split at each stage of tree growth. Existing methods of finding oblique splits avoid this computational explosion in one of two ways,

Direct Search Methods: search for splits with low impurity values in the split dictionary while enforcing some stopping criterion for the search

Indirect Search Methods: focus on a subset of the split dictionary found by solving well-defined problems (which effectively produce oblique splits)

Each methodology has its inherent strengths and weaknesses for which it is important to understand.

This chapter is organised as follows. Existing work is categorised into the above methodologies and presented chronologically as this allows a clearer understanding of its progression. Section 3.1 presents direct methods and Section 3.2 follows with

indirect methods. Section 3.3 finishes by highlighting the strengths and weaknesses associated with these methodologies.

3.1 Direct Search

The methodology taken by researchers in this section can be summarized as follows,

1. oblique splits can be thought of as hyperplanes ($\sum_{i=1}^{q} a_i X_i = c$)
2. consider the $(q+1)$-dimensional space defined by these hyperplane coefficients
3. define a goodness (or badness) function $G(\sum_{i=1}^{q} a_i X_i = c)$ for hyperplanes
4. find the global maxima (or minima) of $G()$ over R^{q+1}

In viewing the search for the best split in this way, the computational problem persists. It does however render the problem suitable to hill-climbing ideas for finding the global minimum of $G()$. Notable attempts that use this methodology are presented below.

3.1.1 CART with Linear Combinations

One of the earliest direct search methods was proposed by Breiman *et al* as an aside in their CART book (3). As axis-parallel trees were the main focus of their work, they did not name this method though it has since been dubbed *CART with Linear Combinations* (CART-LC) by others.

Their approach to finding oblique splits works as follows. Starting from the best axis-parallel split, look for full oblique split with high goodness values $G()$ in this vacinity. From the best full oblique split found, remove as many attributes

as possible without affecting the goodness measure too much. This allows concise oblique splits to be found upon which, tree growth continues as usual. A pseudocode description is given.

Algorithm 1 CART-LC

Require: (a) a goodness measure $G()$ for comparing hyperplanes

{*Scale all continuous attributes*}
Centre all continuous attributes at their median and divide by their IQR

{*Find best axis-parallel and concise oblique split*}
$L_{axis} \leftarrow$ best axis-parallel split
$L_{concise} \leftarrow$ concise oblique split found by calling ***CART-LC - Concise***
return the better of L_{axis} and $L_{concise}$ w.r.t. $G()$

Though CART-LC allows concise oblique splits to be found, it does so in a highly heuristic manner. A further criticism to add is the restrictiveness of its search over the $q+1$-dimensional space of oblique splits. There is no guarantee that local maxima of $G()$ in the vacinity of the best axis-parallel split is even close to the global minima.

3.1.2 Simulated-Annealing Decision Trees

In terms of the restrictiveness of the search-space, Heath *et al*'s (1993) (12) Simulated-Annealing Decision Trees (SADT) greatly improves upon CART-LC as it theoretically allows a search over the entire $q + 1$-dimensional space. Starting at an arbitrarily chosen hyperplane, coefficients are perturbed with Unif[-0.5,0.5) random variables under a simulated-annealing (13)(9) regime whereby perturbed hyperplanes that improve $G()$ are always accepted while those that do not are accepted with decreasing probability. SADT terminates when $G()$ remains unchanged for a prespecified number of iterations (3000-30000 iterations were used in their paper).

Algorithm 2 CART-LC - Concise

{*Find a concise oblique split by perturbing the best axis-parallel split and removing insignificant attributes*}

Require: (a) a goodness measure $G()$ to compare hyperplanes and (b) a tolerance variable β

{*Find full oblique split by perturbing the best axis-parallel split*}
$L_{axis} \leftarrow$ best axis-parallel split
$M \leftarrow \{1, \ldots, q\}$
$\sum_M a_i X_i = c \leftarrow$ ***CART-LC - Oblique***(L_{axis}, M)

{*Improve interpretability of $L_{oblique}$ by elimination attributes*}
loop
  $\Delta^\star \leftarrow G(\sum_M a_i X_i = c)$

  {*Consider effect of removing each attribute singly*}
  for each $m \in M$ **do**
    Find c_m that maximises $G()$ for the hyperplane $\sum_{M \backslash m} a_i X_i = c_m$
    $\Delta_m \leftarrow G(\sum_{M \backslash m} a_i X_i = c_m)$
  end for

  {*If large reduction in $\Delta^\star$ is possible, remove attribute and continue*}
  if $\Delta^\star - \max_m \Delta_m < \beta(\Delta^\star - \min_m \Delta_m)$ **then**
    $m^\star \leftarrow \arg\min \Delta_m$
    $a_{m^\star} \leftarrow 0$
    $c \leftarrow c_{m^\star}$
    $M \leftarrow M \backslash m^\star$
  else
    Exit **loop**
  end if
end loop

{*Find best concise oblique split that use the same continuous attributes starting from concise oblique split just found*}
$L^\star_{concise} \leftarrow$ ***CART-LC - Oblique***$(\sum_M a_i X_i = c, M)$
return $L^\star_{concise}$

Algorithm 3 CART-LC - Oblique

{*Find locally optimal oblique split by perturbing current hyperplane*}

Require: (a) the current split $\sum_{i=1}^{q} a_i X_i = c$, (b) a set $M \subset \{1, \ldots, q\}$ denoting the continuous attributes improvements are considered upon, (c) a goodness measure $G()$ to compare hyperplanes and (d) a tolerance variable ϵ

{*Perturb current hyperplane until improvements to* $G()$ *are small*}
$i \leftarrow 1$
Label current hyperplane L_i
repeat
  {*Find pair* (a_m, c) *that locally maximises* $G()$ *for each* $m \in M$}
  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 maximises $G()$ for the hyperplane $\nu - \delta(X_m + \gamma) = c$
    $a_m \leftarrow a_m - \delta^\star$
    $c \leftarrow c + \delta^\star \gamma^\star$
  end for
  {*Find intercept* c *that maximises* $G()$ *when holding* $(a_1, \ldots, a_q)$ *fixed*}
  Keeping ν fixed, find $c^\star$ to maximise $G()$
  {*Prepare for next iteration*}
  $i \leftarrow i + 1$
  Label current hyperplane L_i
until $|G(L_{i-1}) - G(L_i)| < \epsilon$
return current hyperplane L_i

As simulated-annealing is stochastic in nature, a different tree is grown at each attempt. Though Heath *et al* market this as a strength, I argue that it simply leaves unaddressed the issue as to how one chooses among the 1000's of trees that can be grown. Also, no attempt is made to find concise oblique splits, SADT trees use full oblique splits at every node and so are not interpretable. A final point to note is the run-time of SADT. As the time taken to find oblique splits is dictated by cooling rate of the simulated-annealing regime, this overhead is propogated through the tree and so unnecessarily prolongs tree growth.

3.1.3 Oblique Classifier 1

As regards to run-time, Murthy *et al*'s (1993) (17) OC1 (Oblique Classifier 1) improve upon SADT by simply replacing the simulated-annealing regime with a deterministic minimum finding algorithm. In order to escape local minima, they allow random bumps of hyperplanes that are stuck and allows multiple random starts (trapped hyperplanes are bumped 5 times and 20-50 random restarts are used). A pseudocode description is given.

Algorithm 4 OC1

Require: (a) a badness measure $G()$ to compare hyperplanes and (b) K specifying the number of restarts

{*Find best axis-parallel and make K attempts to find best oblique split*}
$H^{axis} \leftarrow$ best axis-parallel split
for k in $1, \ldots, K$ **do**
  $H_k^{oblique} \leftarrow$ oblique split found using ***OC1:Oblique***
end for
$k^\star \leftarrow \arg\min_k G(H_k^{oblique})$
return the better of H^{axis} and $H_{k^\star}^{oblique}$ w.r.t. $G()$

The main component of OC1 is ***OC1:Minimise*** where they transform a global

Algorithm 5 OC1:Oblique

Require: (a) J specifying the number of attempts to escape local minima

$H^{oblique} \leftarrow$ randomly initial hyperplane

{*Search for global minima of* $G()$ *over* R^{q+1}}
repeat

{*Given* Π *specifing the order in which coefficients in are optimised over, minimise* $G()$}
repeat
Call ***OC1:Minimise***(a_m) to find a hyperplane with lower $G()$
until trapped at a local minima of $G()$

{*Attempt to escape local minima*}
$R \leftarrow$ randomly chosen $\mathbb{R}^{q+1}$ vector
Choose $\alpha^\star$ to minimise $G()$ over the splits $H_R^{oblique}(\alpha) = H^{oblique} + \alpha R$
if $G(H_R^{oblique}(\alpha^\star)) < G(H^{oblique})$ **then**
$H_{oblique} \leftarrow H_R^{oblique}(\alpha^\star)$
end if
until J attempts have been made to escape local minima
return $H^{oblique}$

Algorithm 6 OC1:Minimise

Require: (a) Current hyperplane coefficients H and (b) a coefficient a_m to optimise

{*Minimising* $G()$ *over* a_m *deterministically*}
for each observation j **do**
$h_j \leftarrow$ numerical value of observation j when evaluated on H
$u_j \leftarrow \frac{a_m X_{jm} - h_j}{X_{jm}}$
end for
$a_m^\star \leftarrow$ best univariate split point found over u_j's
$H^\star \leftarrow (a_1, \ldots, a_{m-1}, a_m^\star, a_{m+1}, \ldots, a_q)$

{*Accept* $H^\star$ *under a simulated-annealing-like regime*}
if $G(H^\star) < G(H)$ **then**
return $H^\star$
$P_{move} = P_{stagnation}$
else if $G(H^\star) = G(H)$ **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

minimization problem over a $(q+1)$-dimensional surface into many 1-dimensional ones. By reusing an idea from Breiman *et al*'s CART book, these smaller problems are easily solved in a deterministic manner. As the approach taken by OC1 is identical to SADT, other than improving run-time, OC1 suffers from the same problems as SADT of uninterpretable trees and multiple solutions.

3.1.4 Similar Work

Cantu-Paz *et al* (2000) (8)(7) and Tan *et al* (2004) (21) also contributed to this area using direct search ideas. Instead of maxima (minima) finding algorithms, they use genetic algorithms which essentially work in the same way. 'Mutations' of splits (randomly perturbing hyperplanes) are compared with $G()$ upon which 'fit' splits (those with lower impurity) are always accepted and 'less fit' ones (which do not reduce impurity) are accepted with decreasing probability.

3.2 Indirect Searches for Oblique Splits

The methodology taken by researchers in this section can be summarized as follows,

1. oblique splits ($\sum_i a_i X_i = c$) are simply linear decision boundaries
2. specify a family of classification problems to solve (which effectively defines a set of oblique splits, i.e. a subset of the oblique split dictionary)
3. focusing on this subset, continue tree growth as usual by finding the split with the lowest impurity

Many researchers have made use of this idea to create tree-like classifiers. Though each researcher has a different emphasis, it is still useful to understand each attempt.

3.2.1 Fast Algorithm for Classification Trees

One of the earliest attempts to growing oblique trees with this methodology is FACT (Fast Algorithm for Classification Trees). Proposed by Loh *et al* (1988) (14), it uses LDA to perform *tests* at each node. By fitting an LDA classifier to *residual observations* (those observations in the training set that fall to a particular node), each node has as many child nodes as there are *residual classes*; FACT grows *multi-way* trees. With fewer observations of each class in latter nodes during tree growth, Loh *et al* worry that covariance matrices will become singular and so apply LDA to the projections of observations to those principal components whose eigenvalues are greater than 0.05. Though this is a rather artificial way of growing multi-way classification trees, it is instructive to understand this idea.

3.2.2 Linear Machine Decision Trees

Proposed by Utgoff *et al* (1991) (22)(4)(5), Linear Machine Decision Trees (LMDT) grow multi-way oblique trees using perceptron (19) training ideas. At each stage of tree growth, residual observations are firstly sphered. If there are R residual classes at a node, R linear discriminants $g_i(Y) = W_i^T Y$ are found by training *thermal perceptrons* (11). A thermal perceptron is essentially a perceptron that is forced to converge by progressively down-weigh misclassifications that are distant from the perceptron.

3.2.3 Linear Programming

Brown *et al* (1996) (6) set up linear programming problems that are very similar to those posed by support vector machines to grow binary-splitting oblique trees. To find oblique splits during tree growth, the following family of linear programming

problems are solved. Let $C_R = \{C_1, \ldots, C_R\}$ denote residual classes at a node and consider the following problem for class C_i,

Minimise δ subject to:

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

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

$$b + \sum w_k = const$$

The solution to each problem produces an oblique split that tried to separate observations of class C_i from all others. By solving all R problems, tree growth continues as usual by applying the split with the lowest impurity.

3.2.4 Perceptron-Based Oblique Trees

The last method shown uses logistic regression to find oblique splits. Misleadingly called Perceptron-Based Oblique Trees (P-BOT), Axelrod *et al*'s (2005) (2) method grows two-level tree-like structures by concatenating logistic regression classifiers. Predictions from the logistic regression classifier in the first-level paritions observations into several child nodes upon which another logistic regression classifier is trained to residual observations in each child node. This second-level classifier provides classifies observations. As the focus of their work is to simply create more flexible tree-like classifiers, no attention is paid to making P-BOT interpretable (all attributes are used as inputs to the logistic regression classifiers). P-BOT is essentially a two-level multi-way oblique tree that is not interpretable.

3.3 Summary of Existing Work

As noted at the start of this chapter, direct search methods and indirect search methods seek to accomplish the same goal; allowing good oblique splits (oblique splits with low values of impurity) to be quickly found. There are two fundamentally conflicting goals to balance to achieve this.

Quality of splits: The more splits the method considers, the more likely that splits with low impurity values are found. Direct search methods in theory allow every split in the split dictionary to be considered while indirect search methods focus on some predefined subset. Whichever method used, splits with low impurity values should be considered by the subset.

Computation cost: The more splits considered, the greater the computational cost of the method. Indirect searches allow as many (or as few) splits to be considered as one desires and so allows oblique trees to be grown quickly.

In addition to this, as noted in Section 2.2 concise oblique splits should be used whenever possible. Unfortunately, very little attention is given to this by methods from both methodologies. Chapter 4 presents an new approach to finding oblique splits that incorporates the best from both methodologies.

Chapter 4

Oblique Splits via Probabilistic Models

This chapter presents a new approach to growing *full oblique trees* (trees grown by considering full oblique splits). The question of how one obtains more interpretable oblique trees is deferred to Chapter 5.

This chapter is organised as follows. By revisiting original ideas behind tree growth in Section 4.1, several natural steps are taken which allow a small yet comprehensive set of oblique splits to be found. With this, tree growth can continue as usual to grow oblique trees. A summary of these steps is as follows. By identifying a family of two-class classification problems similar to that of Brown *et al*, they can be solved with linear classifiers which effectively produce oblique splits. Section 4.2 discusses reasons as to why logistic regression is used for this task. Section 4.3 follows with a demonstration of this approach by growing oblique trees on several widely used classification datasets. This chapter ends with a heuristic argument as to why this set of oblique splits performs so well given its relatively small size.

Figure 4.1: Ideal outcomes when of observations are from 4 residual classes

4.1 Ideal Outcomes

Existing methods for finding oblique splits seek to find the split that minimises impurity over some subset of the oblique split dictionary. Rather than jumping straight into this problem, it is useful to recall why splits with low impurity are desireable to begin with.

As covered in Section 2.1, a classification tree recursively partitions observations into smaller subsets where observations have more similar classes. *Node impurity* is a measure that quantifies this idea of class similarity which according to Breiman *et al* (3)(p. 24), should be zero when observed class probabilities are concentrated on one class and maximal when they are uniformly distributed over all classes. Extending this idea of node impurity, *split impurity* is a measure that quantifies the ability of a split to partition observations. It is defined to be the weighted average impurity over child nodes that would result where such a split applied to the training set. In summary, split impurity is a measure that quantifies some notion of separation.

How would an ideal test partition observations at a node? *Ideal outcomes* are the result of tests that perfectly partition observations of several classes against all others. With R residual classes at a node, there are $2^{R-1} - 1$ ideal outcomes as there are $2^{R-1} - 1$ ways of binning them R balls into two non-empty buckets. To illustrate the idea of ideal outcomes, when there are 4 residual classes, there are 7 $(= 2^{4-1} - 1)$ ideal outcomes. They are listed in the titles of plots in Figure 4.1.

Ideal outcome {1}{2,3,4} denotes a test that perfectly separates class 1 observations from all class 2, 3 and 4 observations. These tests are exactly what one wishes to use during tree growth as perfectly partitions observations at each stage of tree growth quickly produces pure nodes. Ideal outcomes are interesting targets to aim for.

By considering ideal outcomes as two-class classification problems, they can be solved with linear classifiers to give linear decision boundaries that best approximate them. As linear decision boundaries are synomynous with oblique splits, *ideal splits* (oblique splits that best approximate ideal outcomes) are thus found. They are shown Figure 4.1. As is easy to see, ideal splits replicate ideal outcomes rather well. Ideal split {1}{2,3,4} manages to partition most class 1 observations against all others as do ideal splits {3}{1,2,4}, {4}{1,2,3}, {1,3}{2,4} and {1,2}{3,4}. Some ideal outcomes simply are not easily reproduced with oblique splits including {2}{1,3,4} and {1,4}{2,3}. Even so, ideal outcomes allow useful oblique splits to be found.

As ideal splits are based on ideal outcomes which perfectly partition observations, ideal splits should also partition observations well and so give low impurity values. As such, with R residual classes at a node, $2^{R-1} - 1$ interesting oblique splits can be found.

4.2 Realising Oblique Splits

With many linear classifers, there is great freedom as to which is used. Other than being linear, additional qualities are also desireable for such a classifier. These include,

Computationally light: $2^{R-1} - 1$ classification problems must be solved at each node, the cost of training such a classifier is greatly magnified during tree growth.

Automated training: With so many classification problems needing to be solved, such a classifier should require minimal human input (if any at all).

Able to find separating hyperplanes: Where a two-class classification problem is linearly separable[1], the classifier should find separating hyperplanes.

Applicable to feature selection ideas: Though unnecessary for finding full oblique splits, the ability to remove insignificant attributes allows concise oblique splits to be found.

With these criteria in mind, reviewing several linear classifiers quickly shows why logistic regression (15) is used.

Perceptron Learners: Perceptron learners do not converge in the presence of non-linearly separable data.

Support Vector Machines: Soft-margin support vector machines allow "separating hyperplanes" to be found on non-linearly separable data. Unfortunately, a quadratic programming problem must be solved making it computationally intensive. In addition to this, feature selection ideas do not seem to be applicable.

Linear Discriminant Analysis: Though straightforward to train, LDA does not always find separating hyperplanes on linearly separable data.

[1]Two-class classification data is linearly separable if there exists a hyperplane that partitions all observations of one class from the other

Linear Regression: Feature selection techniques are directly applicable however linear regression does not always find separating hyperplanes on linearly separable data.

Logistic Regression: Though there are situations where logistic regression fails to converge, checks can be put in place to overcome them. Seeking to maximise a likelihood, logistic regression finds separating hyperplanes where they exist and feature selection ideas are also applicable.

In summary, logistic regression is used to extract oblique splits.

Having fit a logistic regression model, one needs to find an expression for its decision boundary. This is straightforward to do. The binomial logistic regression model assumes that posterior log-odds of one class against another (say the 2nd to the 1st) takes the following form

$$\log \frac{P(G = 2|X = x)}{P(G = 1|X = x)} = a + \sum_{i=1}^{q} a_i x_i.$$

As the decision boundary is located where the posterior odds of both classes are equal, it is similarly where the log-odds equals zero, i.e. the oblique split found using logistic regression that approximates an ideal outcome is given by the linear form with plug-in maximum likelihood estimates

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

4.3 Examples of Oblique Trees

As mentioned in Section 3.3, one of the shortcomings of existing work is the unavailability of easily useable implementations. Ideas mentioned in this thesis are all implemented in R to address this issue and are used throughout this thesis to demonstrate ideas.

The function `oblique.tree(formula,data,subset,split.impurity,...)` allows various types of classification trees to be grown including,

1. Axis-parallel trees: that use axis-parallel splits exclusively
   `oblique.tree(...,oblique.splits="off")`
2. Mixture trees: that consider both axis-parallel and oblique splits (whichever has the lowest impurity)
   `oblique.tree(...,oblique.splits="on")`
3. Oblique trees: that use oblique splits exclusively
   `oblique.tree(...,oblique.splits="only")`

The exact stopping criteria of tree growth is as follows. Nodes are no longer partitioned if any of the following is true.

1. Pure nodes: if all observation are of the same class
2. Deviance threshold: if the deviance of a node is below some prespecified proportion of the deviance of the root node (defaulted to 0.01)
3. Size threshold: if the number of observations is less than some prespecified limit (defaulted to 10)

Figure 4.2: Axis-parallel tree, mixture tree and oblique tree grown on original crabs dataset

4. Child node threshold: if all splits result in a child node with less than some prespecified number of observations (defaulted to 5).

As oblique splits are better able to partition observations, the above three trees should require progressively fewer tests in theory. All three are grown on several classification datasets with the same stopping criteria to allow fair comparison and no attempt is made to improve their interpretability.

4.3.1 Original Crabs Data

Starting with the crabs dataset, it consists of 5 morphological measurements of 200 crabs (50 crabs of two colours from both sexes) collected at Fremantle, W. Australia. As significant variation the data arises due to differing maturities of crabs, this dataset is often used to demonstrate the principal components analysis technique. Using this data, Figure 4.2 shows the above three trees are grown to this data.

It is known that observations of different classes are well separated by oblique splits in this dataset. The axis-parallel tree grown is rather large as requires concatentated use of axis-parallel splits to partition observations. Allowing oblique splits, mixture trees and oblique trees use oblique splits to produce large reductions in deviance.

It should also be noted that when using oblique splits exclusively, only 7 ($=2^{4-1}-1$) splits need to be evaluated at the root node as compared to the 995 odd ($\approx (200-1)\times 5$) in the case of axis-parallel trees.

Figure 4.3: Axis-parallel tree grown on the Augmented Crabs dataset with its associated decision boundaries

Figure 4.4: Mixture tree grown on the Augmented Crabs dataset with its associated decision boundaries

4.3.2 Augmented Crabs Data

By projecting the crabs dataset onto its principal components second and third principal components, observations of different classes are much better separated. Axis-parallel trees should perform better on this *Augmented Crabs dataset* as a result. With a 2-dimensional dataset, it is also possible to view the decision boundaries of each tree as is shown in Figures 4.3- 4.5.

Looking at the trees, the axis-parallel tree is indeed smaller than the one grown on the original crabs dataset however when compared to mixture and oblique trees, it still uses more tests. By examining decision boundaries of each tree, it is easy to see why. Observations are best partitioned with oblique splits.

4.3.3 Pima Indians Data

The Pima Indians dataset contains 7 measurements from 532 women of Pima Indian heritage living near Pheonix, Arizona tested for the presence (or lackthereof) of diabetes. A subset containing 200 observations called `Pima.tr` found in the *Modern Applied Statistics with S-Plus* library in R is used as the training set. Trees grown on this dataset are shown in Figure 4.6 both with and without text for greater clarity.

Figure 4.5: Oblique tree grown on the Augmented Crabs dataset with its associated decision boundaries

Figure 4.6: Trees grown on the Pima Indians dataset and their associated tree skeletons

Figure 4.7: Trees grown on the Glass Fragments dataset and their associated tree skeletons

As with before, much few tests are used when oblique splits are used.

4.3.4 Glass Fragments Data

The same can be said for the Glass Fragments dataset.

4.4 Locally Optimal Subset

The set of oblique splits specified in Section 4.1 is intuitively appealing and as examples in Section 4.3 show, allow interesting oblique trees to be grown. Given that the oblique split dictionary is of size $\sum_{k=0}^{q} \binom{L_{gen}-1}{k} - 1$, it is interesting to consider how a subset of size $2^{R-1} - 1$ allows *good oblique splits* (splits with low impurity values) to be found. By giving a heuristic arguing as to why most splits in the split dictionary can be ignored and with empirical evidence to back this claim, this sections explains why this is so.

As has already been noted, widely used methods of tree growth find splits over continuous attributes by ideally evaluating the impurity of *every* split in the split dictionary, a great deal of duplicated work for finding the best split. Contains splits that produce all possible partitioning of observations to child nodes, some splits differ by only an observation or two (in its child nodes) are considered separate splits in the split dictionary. Such splits are said to be *similar* and it follows that

Figure 4.8: Plot showing smooth evolution of impurity measure over splits in the axis-parallel split dictionary

Figure 4.9: Plot showing smooth evolution of impurity surface over sampled splits from the oblique split dictionary

their impurities are also very similar. Just how many similar splits are there in the axis-parallel and oblique split dictionaries?

For any given axis-parallel split $X_i = c$, increasing (or decreasing) c to the next unique value of X_i in the training set can result in a similar split (failing to hold only when many observations take the same value of X_i). There are therefore around 2 similar splits for any given axis-parallel split. The evolution of the impurity measure over a typical training sets are shown in Figure 4.8.

For an arbitrary oblique split $\sum_{i=1}^{q} a_i X_i = c$, there is much greater freedom to produce similar splits. Increasing (or decreasing) any of the q coefficients results in similar splits (failing to hold only in the unlikely event that many observations take the same value over *all* q continuous attributes). Though these 2^q similar splits need not produce unique outcomes, there are still anything between $2 \leq m \leq 2^q$ similar outcomes to any given oblique split. Figure 4.9 shows a typical evolution of the impurity measure when $q = 2$.

As considering similar splits only yields marginal information about the impurity surface, it is desireable to avoid similar splits. As the impurity plots show, if observations of different classes lie in contiguous portions of the feature space, split impurity is low in areas *between* these contiguous clusters of observations. For 1-dimensional plots in Figure 4.8, it is evident upon closer examination that kinks

in the impurity measure occur where class clusters terminate. This phenomenon is particularly evident in Figure 4.9, where two valleys are formed which coincide in regions between class clusters.

By focusing on regions between class clusters, one can find splits with low impurity values. It is these regions that oblique splits identified in Section 4.1 focus upon.

Chapter 5

Interpretable Oblique Trees

As interpretability is a key strength of classification trees, it is important to find ways of making oblique trees more interpretable. Apart from simply being small, guarding against unwarranted use of full oblique splits also helps trees to be more interpretable. Various approaches can be used to achieve this. They are categorised into two groups,

During Tree Growth methods applied during tree growth and

Post-Tree Growth methods applied post-tree growth.

All methods use well-founded feature selection ideas introduced to trees by solving two-class classification problems during tree growth. Section 5.1 covers methods applied during tree growth and Section 5.2 covers those applied post-tree growth.

5.1 During Tree Growth

Having seen how full oblique trees can be grown, it is easy to extend ideas in Chapter 5 to grow concise oblique trees. When looking for the best split at each stage of tree growth, the set of full oblique splits can be substituted with a set of concise

oblique splits. Instead of fitting logistic regression classifiers with all continuous attributes to two-class classification problems, feature selection techniques can be applied to remove attributes from the final model. This effectively produces concise oblique splits.

There are two established methods of finding succinct classifiers when fitting logistic regression classifiers,

- Model Selection
- Penalised Likelihood

Section 5.1.1 expands on the use of model selection ideas and Section 5.1.2 expands on penalised likelihood.

THE REST OF THIS CHAPTER MUST BE UPDATED
COMMENTS ARE WRITTEN IN BOLD

5.1.1 Model Selection (1 MORE PAGE)

INTRODUCE AIC AND BIC
COMPARE AIC AND BIC, EXPLANATORY VS PREDICTIVE
TALK ABOUT R IMPLEMENTATION FOR MODEL SELECTION
GIVE EXAMPLES OF SUCH TREES

Akaike's *An Information Criterion* (1) allows one to estimate the divergence between the log-likelihood of a probabilistic model trained on some dataset, to the log-likelihood of the same model on out-of-sample data (previously unobsersed data). A practical application of AIC involves augmenting the log-likelihood of a

model by the infamous value $2q$ (where q is the number of degrees of freedom of the model) which allows nested submodels to be compared on their (estimated) ability to accurately predict on out-of-sample data. The result of this procedure is that more succinct submodels may be preferred over the original model (which uses all q attributes). In relation to logistic regression, attributes can be automatically removed from the model to produce concise oblique splits.

Schwarz's Information Criterion, more commonly known as the *Bayesian Information Criterion* (20) provides a similarly practical result. Motivated through a bayesian framework, BIC penalises the log-likelihood of probabilistic models more severely than AIC and so for our purposes, produces oblique splits that are even more concise.

5.1.2 Penalised Likelihood (ANOTHER 3 PAGES)

EXPLAIN PENALISED LIKELIHOOD WORKS
SHOW HOW TO CAN BE USED WITH OBLIQUE TREES
TALK ABOUT R IMPLEMENTATION (USED GLMPATH) GIVE EXAMPLES OF SUCH TREES

5.2 Post-Tree Growth

Though methods presented in Section 5.1 allow oblique trees with concise oblique splits to be grown, it is not easy to specify how concise such splits should be. Post-tree growth methods are better able to do this and allows the user to choose how interpretable the resulting tree should become.

Section 5.2.1 begins with the observation that traditional cost-complexity pruning methods can also be applied to the resulting oblique trees that are grown.

5.2.1 Tree Pruning (ANOTHER 1/2 PAGE)

COST-COMPLEXITY PRUNING ALSO WORKS WITH OBLIQUE TREES

TALK ABOUT R IMPLEMENTATION

GIVE AN EXAMPLES OF PRUNED OBLIQUE TREES

By examining ideas behind cost-complexity pruning of axis-parallel trees, it is easy to see that such techniques are equally applicable to oblique trees. An large oblique tree can be pruned to reveal a family of rooted subtrees giving the user the ability to specify the size the of tree desired.

5.2.2 *Squeezing* Trees Using Shrinkage Methods

NEED TO RETHINK THIS SECTION THROUGH

At any internal node t, we can fit a penalised logistic regression model to the data. By varying the value of the penalisation constant λ_t, we can *squeeze* out unneccessary terms. Though different forms of penalisation (L_1, L_2, etc) affect the nature in which variables depart the model, this is immaterial at this stage of discussion. For each logistic regression model used at nodes t, it is important to note that $\lambda_t^{\max}$ (the smallest value of λ_t where the coefficients of the model are not affected by penalisation) varies significantly. I therefore feel that it is unwise to use a single numeric value of Λ to simultaneuously penalise all logistic regression models

in the oblique tree. Below are two better ways to holistically applying penalisation to logistic regression models in the tree,

Percentage-wise Reducing a single term, the percentage Λ_{perc}, uniformly reduce the percentage of $\lambda_t^{\max}$ used for the associated logistic regression models simultaneously

Sum-wise Reduce the single term $\Lambda_{\text{sum}} = \sum_{\text{nodes } t \text{ in } T} \lambda_t^{\max}$, this forces all coefficients down to zero as $\Lambda_{\text{sum}} \to 0$

Though both approaches seem reasonable, sum-wise penalisation seems unnecessarily complicated to implement and so I prefer the percentage-wise approach. Having said this, my experience of using penalised logistic regression suggests that extracting the transition points of λ_t where attributes leave the model is itself a difficult process. I have also noticed that the resulting logistic regression models do not remove terms with small coefficients as thoroughly as subset selection methods. This brings me to my preferred post-tree growth approach.

5.2.3 Trimming Trees (ANOTHER 4 PAGES)

IT IS POSSIBLE TO ADAPT COST-COMPLEXITY PRUNING TO PRUNE THE SPLITS OF A TREE RATHER THAN ITS NODES. DOING SO ALLOWS A GIVEN TREE TO BE *TRIMMED* SO THAT MORE CONCISE OBLIQUE SPLITS ARE USED. THIS SECTION EXPANDS UPON THIS IDEA. UNFORTUNATELY, I HAVE NOT WRITTEN OUT THE CODE FOR IT YET AND SO IT CANNOT BE TESTED YET.
THIS SECTION EXPLAINS THIS IDEA, GIVING EXAMPLES AS WELL.

As with Breiman *et al*, let $R(T)$ be a measure of the tree formed by adding contributions over its leaves and consider the penalised measure of fit

$$R_\alpha(T) = R(T) + \alpha \text{comp}(T).$$

Here, we depart slightly from traditional theory by forming $\text{comp}(T)$ defined as follows,

$$\text{comp}(T) = \sum_{B\text{: branches of } T} \frac{\#\text{ attributes used over all internal nodes of } B}{\#\text{ internal nodes of } B}.$$

Though this may appear complicated, it is instructive to note that $\text{comp}(T)$ collapses to the usual definition of $\text{size}(T)$ when T is an axis-parallel tree. For oblique trees however, this measure further penalises trees that require more decision making to reach its leaves, it rightly skews our attention to trees that have simpler splits nearer the root node.

Traditional cost-complexity pruning seeks to find the smallest rooted subtree that minimises $R_\alpha(T)$ for some α. We again depart slightly by seeking to minimise $R_\alpha(T)$ over a *different* set of candidate trees, *concise-trees*. For an oblique tree T, its family of *concise-trees* has the same splitting structure in the sense that,

1. splits occur at the same internal nodes,

2. logistic regression models at internal nodes are based on the same partitioning of residual classes from the training set and

3. only a subset of the attributes used in the associated split of the original tree T is permitted.

Figure 5.1: Example of concise-trees and comp(B) for a small oblique tree

Figure 5.1 illustrates the concept of concise-trees and comp() for individual branches more clearly. We wish to search over the set of concise-trees of T to find the tree that minimises $R_\alpha(T)$ for some α.

At this point, the elegant theory of cost-complexity pruning that allows efficient navigation over all rooted subtrees no longer holds for the more complicated family of concise-trees. Proposition 7.2 of Ripley's *PPRN* allows for a fast and efficient search for $T(\alpha)$ (the smallest rooted subtree that minimises R_α) for a particular α. The simple structure of rooted subtrees is pivotal in allowing such a computational shortcut to exist, added complications of concise-trees lends no such luxury. Though it is theoretically possible to find the concise-tree with the smallest value of R_α (with the smallest value of comp(T)) by searching over the entire family, it would take too long in reality. The following greedy heuristic search however may prove promising.

We are firstly given some penalisation constant α. For an internal node t of an unoptimised oblique tree T, there are n_t explanatory variables in the logistic regression model. Subset selection methods or shrinkage methods can reveal a new logistic regression model with $n_t - 1$ terms which results in a new oblique split. The concise-tree T_t^{concise} that replaces node t with this new logistic regression model and leaves all other nodes untouched will have cost-complexity value $R_\alpha(T_t^{\text{concise}})$. Considering all such concise-trees by looking at all nodes t, we can greedily move to the concise-tree that gives the greatest reduction in R_α. When R_α reaches a minimum, we accept this concise-tree T_α^{heur} as our approximation to T_α, the best

concise-tree.

Proposition 7.3 of Ripley's *PPRN* allows efficient traversal of all interesting values of α. Again, this shortcut is not accessible for our more complex family of concise-trees. A brute force approach to find T_α^{heur} over a mesh of values of α seems feasible, we can stop searching when T_α^{heur} becomes the trivial tree.

This approach allows a family of concise-trees with small value of R_α to be extracted from an unoptimised oblique tree from which the best generalising concise-tree can be chosen as usual.

Chapter 6

Additional Points

There are several issues related to the use of this method on real datasets I want to bring forth in this chapter.

6.1 Weakness

6.1.1 Poor Oblique Splits (ANOTHER 5 PAGES)

I believe that are times when the approach taken in Section 4 may fail to provide good oblique splits. Situations include the case where class data is non-contiguous, and non-spherical. If this were the case, oblique splits found in this way are not attracted to areas of low impurity values. This needs to be investigated further.

6.1.2 Complexity Analysis (ANOTHER 3 PAGES)

A investigation on the computational complexity of techniques mentioned in this thesis needs to be done. This should highlight situations where this approach to growing oblique trees does not perform well. A definite area of weakness includes

the case where there are a great number of classes in the classification dataset. It may also perform poorly in the situation where there are many continuous attributes also. May also include comparison with existing methods of oblique tree growth.

Chapter 7

Concluding Remarks (ANOTHER 2 PAGES)

Final remarks. This will contain my thoughts of the implications of this work. Mention R package will be provided (not that it can be made faster). Touch upon addition research areas.

- Using ideal splits as starting points for hill-climbing methods
- Extending idea to regression trees, using clustering methods
- Random forests idea over classes to speed up tree growth for large problems
- Provide an implementation in C to speed up existing R work

Bibliography

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

[2] Ben Axelrod, Stephen Campos, and John Envarli. Perceptron-Based Oblique Trees (P-BOT). Master's thesis, Georgia Institute of Technology, 2005.

[3] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

[4] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. Technical Report UM-CS-1992-083, University of Massachusetts, , 1992.

[5] Carla E. Brodley and Paul E. Utgoff. Multivariate versus univariate decision trees. Technical Report UM-CS-1992-008, University of Massachusetts, 1992.

[6] Donald E. Brown, Clarence Louis Pittard, and Han Park. Classification trees with optimal multivariate decision nodes. *Pattern Recogn. Lett.*, 17(7):699–703, 1996.

[7] Erick Cantu-Paz and Chandrika Kamath. Inducing oblique decision trees with evolutionary algorithms. citeseer.ist.psu.edu/article/cantu-paz03inducing.html.

[8] Erick Cantu-Paz and Chandrika 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, 10-12 2000. Morgan Kaufmann.

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

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

[11] Marcus Frean. A "thermal" perceptron learning rule. *Neural Computation*, 4(6):946–957, 1992.

[12] David G. Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *International Joint Conferences on Artificial Intelligence*, pages 1002–1007, 1993.

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

[14] Wei-Yin Loh and Nunta Vanichsetakul. Tree-structured classification via generalized discriminant analysis. with a comment by Leo Breiman and Jerome H. Friedman and with a reply by the authors. *Journal of the American Statistical Association*, 83(403):715–728, 1988.

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

[16] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, April 1965.

[17] Sreerama K. Murthy, Simon Kasif, Steven Salzberg, and Richard Beigel. OC1: A randomized induction of oblique decision trees. In *National Conference on Artificial Intelligence*, pages 322–327, 1993.

[18] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

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

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

[21] Peter Tan and David Dowe. MML inference of oblique decision trees. *Proceedings of the 17th Australian Joint Conference on Artificial Intelligence*, pages 1082–1088, 2004.

[22] Paul E. Utgoff and Carla E. Brodley. Linear Machine Decision Trees. Technical report, University of Massachusetts, Amherst, MA, USA, 1991.