OBLIQUE DECISION TREES IN TRANSFORMED SPACES

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Statistics
at the University of Canterbury

By

Darshana Chitraka Wickramarachchi

School of Mathematics and Statistics
Faculty of Engineering
University of Canterbury
New Zealand

September 2015

Table of Contents

List of Tables

List of Figures

Abstract

Decision trees (DTs) play a vital role in statistical modelling. Simplicity and interpretability of the solution structure have made the method popular in a wide range of disciplines. In data classification problems, DTs recursively partition the feature space into disjoint sub-regions until each sub-region becomes homogeneous with respect to a particular class. Axis parallel splits, the simplest form of splits, partition the feature space parallel to feature axes. However, for some problem domains DTs with axis parallel splits can produce complicated boundary structures. As an alternative, oblique splits are used to partition the feature space potentially simplifying the boundary structure. Various approaches have been explored to find optimal oblique splits. One approach is based on optimisation techniques. This is considered the benchmark approach, however, its major limitation is that the tree induction algorithm is computationally expensive. On the other hand, split finding approaches based on heuristic arguments have gained popularity and have made improvements on benchmark methods. This thesis proposes a methodology to induce oblique decision trees in transformed spaces based on a heuristic argument.

As the first goal of the thesis, a new oblique decision tree algorithm, called HHCART (HouseHolder Classification and Regression Tree) is proposed. The proposed algorithm utilises a series of Householder matrices to reflect the training data at each non-terminal node during the tree construction. Householder matrices are constructed using the eigenvectors from each classes' covariance matrix. Axis parallel splits in the reflected (or transformed) spaces provide an efficient way of finding oblique splits in

the original space. Experimental results show that the accuracy and size of the HHCART trees are comparable with some benchmark methods in the literature. The appealing features of HHCART is that it can handle both qualitative and quantitative features in the same oblique split, conceptually simple and computationally efficient.

Data mining applications often come with massive example sets and inducing oblique DTs for such example sets often consumes considerable time. HHCART is a serial computing memory resident algorithm which may be ineffective when handling massive example sets. As the second goal of the thesis parallel computing and disk resident versions of the HHCART algorithm are presented so that HHCART can be used irrespective of the size of the problem.

HHCART is a flexible algorithm and the eigenvectors defining Householder matrices can be replaced by other vectors deemed effective in oblique split finding. The third endeavour of this thesis explores this aspect of HHCART. HHCART can be used with other vectors in order to improve classification results. For example, a normal vector of the angular bisector, introduced in the Geometric Decision Tree (GDT) algorithm, is used to construct the Householder reflection matrix. The proposed method produces better results than GDT for some problem domains. In the second case, *Class Representative Vectors* are introduced and used to construct Householder reflection matrices. The results of this experiment show that these oblique trees produce classification results competitive with those achieved with some benchmark decision trees.

DTs are constructed using two approaches, namely: top-down and bottom-up. HHCART is a top-down tree, which is the most common approach. As the fourth idea of the thesis, the concept of HHCART is used to induce a new DT, HHBUT, using the bottom-up approach. The bottom-up approach performs cluster analysis prior to the tree building to identify the terminal nodes. The use of the Bayesian Information Criterion (BIC) to determine the number of clusters leads to accurate and compact trees when compared with Cross Validation (CV) based bottom-up trees. We suggest that HHBUT is a good alternative to the existing bottom-up tree especially when the

number of examples is much higher than the number of features.

Acknowledgments

In looking back after a three year PhD journey, there are many people to thank who have helped and supported me in various ways. However, there are some individuals who have been in the forefront and had a significant impact in bringing me thus far and I would like to pay tribute to them.

For the last three years, every Thursday I sat with four supervisors around me: Professor Jennifer Brown, Dr. Blair Robertson, Associate Professor Marco Reale and Associate Professor Chris Price. I was very lucky to have such a wonderful set of scholars in my supervisory panel. All their kind advice, guidance, encouragement and also the freedom they gave me to work on my own imagination helped me immensely to explore my research interests.

I may have not been able to pursue the PhD at the School of Mathematics and Statistics at the University of Canterbury if Professor Jennifer Brown had not consented to supervise me. Her guidance helped me in all the time of research. I thank her for her unwavering support.

I was privileged to have Dr. Blair Robertson as one of my supervisors. I still can remember that, at an early stage of the PhD, he was ready to help me every time I knocked on his door. He never let me down. Moreover, his support never faltered during his tenure at the Department of Statistics, University of Wyoming, USA.

I am deeply indebted to Associate Professor Marco Reale for his tremendous supervision in limitless ways. His discussions were not limited to the subject being studied. He shared his knowledge and experiences in many areas of Statistics. Moreover, he advised me how to become a good researcher, a good lecturer and, more

importantly, how to live a balanced academic life. Truly, he is a remarkable mentor.

Finally, to Associate Professor Chris Price, I extend my profound gratitude for his fullest support in giving me precious advice, suggestions and insightful comments in this academic work.

There are many more in the School of Mathematics and Statistics to thank for the support given me during my life at the School. Special thanks to Dr. Clemency Montelle and Dr. Miguel Moyers-Gonzalez for being our post-graduate coordinators, Dr. Patrick W. Saart being my mid-PhD examiner, Dr. Carl Scarrott, Dr. Raazesh Sainudiin, and Dr. Nuttanan (Nate) Wichitaksorn. I must also thank Paul Brouwers and Steve Gourdie for taking care of all computer related issues and giving me a trouble free computing environment. At the same time I would also like to thank all other department members and post-graduate students who were with me to share the ups and downs.

It is my pleasure to remember and thank Dr. Sung Eun Bae, Dr. Franois Bissey and Dr. Celine Cattoen from BlueFern Supercomputing of the University of Canterbury who helped me by providing valuable advice on parallel processing and the comments/suggestions on work flow presented in Chapter 4.

I must also thank the University of Canterbury for awarding me one of the prestigious scholarships, the University of Canterbury International Doctoral Scholarship, for the entire three years period.

Moreover, I would like to thank the University of Sri Jayewardenepura, Sri Lanka, for granting me paid study leave for 39 months to do my PhD research without any disturbance.

Last, but not least, I thank my loving wife Chamila and four children Chathuli, Chamudi, Gayaru and Ganguli for their blessings, patience and understanding throughout the period. Also I thank my parents and all family members who gave their blessings and support.

Chapter 1

Introduction

1.1 Background

With the vast development of information technology, scientists have been able to gather large amounts of data. For example, meteorologists receive an enormous amount of weather data from satellites and DNA micro-array experiments facilitate a quantitative study of thousands of genes simultaneously. Parallel to the development of data gathering technology, extracting information from the data has become a challenge. Because information has to be extracted as fast and accurately as possible, the invention of new methods of data analysis is inevitable. This led to *Data Mining*, a new field of study. According to Dahan, Cohen, Rokach, and Maimon (2014, p. 1), data mining refers to a variety of methods for automatically exploring, analysing and modelling large data repositories in attempt to identify valid, novel, useful, and understandable patterns. Data mining provides tools and techniques that add intelligence to data warehousing and organising.

Hastie, Tibshirani, and Friedman (2009) propose that extracting information, patterns and trends from large amounts of data can be called "learning from data" as an alias for Statistical Learning. Statistical learning can be categorized into supervised learning and unsupervised learning. In supervised learning the objective is to develop

a statistical model to describe the relationship between input variables and a response variable. In unsupervised learning there is no response variable and hence, the objective is to describe the relationships among input variables. Statistical learning theory mainly deals with supervised learning problems.

Supervised learning problems can be considered as two different problems. If the response variable is quantitative, then the problem is a regression problem and if the response is qualitative then the problem is a classification problem. There are many statistical techniques that have been developed to address supervised learning problems, for example, fitting General Linear Models (Montgomery, Peck, & Vining, 2012) or Generalized Linear Models (Nelder & McCullagh, 1989). These conventional modelling approaches assume that postulate model, or the assumed relationship between the conditional expectation of the response and the predictor variables, is to remain the same all over the predictor variable space (Montgomery et al., 2012) and (Breiman, Olshen, Friedman, & Stone, 1984). For example, the model:

$$E(Y|X) = f(X) \text{ where } f(X) = \beta_0 + \sum_{l=1}^{p} \beta_i X_i, \quad \beta_i, X_i \in \mathbb{R}, i = 1, 2, \ldots, p, \text{ and } X \in \mathbb{R}^p \tag{1.1.1}$$

assumes that $E(Y|X)$ can be modelled (or predicted) as a linear combination of predictor variables for all $X \in \mathbb{R}^p$. However, this homogeneity in the relationship between $E(Y|X)$ and $f(X)$ and the smoothness of $f(X)$, are rarely met and may vary in different sub-regions of predictor space especially in higher dimensions (Breiman et al., 1984). On the other hand, piecewise continuous functions can be fitted to disjoint sub-regions in the space of X as a solution to the above problem. One such example of these local methods is k-nearest-neighbour procedure (Hastie et al., 2009). However, this also fails in higher dimensions due to sparseness of data. For instance, in order to capture a fraction r of a set of points uniformly distributed in a p-dimensional unit hypercube, one needs to cover a hypercube of edge length $r^{1/p}$ (Hastie et al., 2009).

That is, to capture 10% of points in a 10 dimension hypercube, 80% of the range in each edge has to be covered and hence, the method is no longer a local. These consequences are commonly known as the "curse of dimensionality"[1] (Breiman et al., 1984).

Therefore, alternative methods have been considered for supervised learning. For example, tree structured models are one kind of non-parametric partition-based prediction models. Tree based models also belong to the class of piecewise continuous functions. There are two types of tree structured models depending on the context of the problem. Regression trees are used for regression problems and classification trees are used for classification problems. In this research, the classification trees are being dealt with as they are very common in practice. DT applications to medicine can be found Breiman et al. (1984); Decaestecker et al. (1996); Podgorelec, Kokol, Stiglic, and Rozman (2002). Bell (1996); Friedl and Brodley (1997); Scull, Franklin, and Chadwick (2005) use DT for environmental sciences problems. Examples of applications in engineering are given by Braha and Shmilovici (2003). Moro, Laureano, and Cortez (2011) and Tirenni, Kaiser, and Herrmann (2007) use DTs in marketing.

The following section introduces data classification and classification trees. Hereafter, DTs refer to classification trees.

1.2 Data classification

1.2.1 Terminology

This thesis uses terms used in DT literature. Some commonly used terms are given in Table 1.1.

[1] a phrase due to Bellman (1961)

Table 1.1: Terminologies.

Term	Description
Example/Set of examples	Observation/Sample
Feature variable	Predictor/Independent variable
Class variable	Response variable
Node/Region	In this thesis, Node and Region are used interchangeably depending on the context. Node represents a region or a sub-region in a classification tree. Region/Sub-region is used to refer space/sub-space of the feature space.
Training Data/ Learning Data	Dataset used to build the tree
Test Data	Dataset used to test the tree

1.2.2 What is data classification?

Data classification is a process of determining the class, Y, of an example based on its p features X_i, where $i = 1, \ldots, p$. In this case, the class of an example is given by a value from a finite set $\mathbb{C} = \{1, 2, 3, ..., C\}$ and the class for the example is assigned according to the rule, $g(x)$. The rule $g(x)$ is called a **classifier**.

A **classification algorithm** is applied to a training set (X_1, Y_1), (X_2, Y_2), $\ldots$, $(X_n, Y_n) \in \mathcal{X} \times \mathbb{C}$ where $\mathcal{X} = \{X_1, X_2, \ldots X_n\}$ and constructs a classifier $g(x)$. In the statistical learning theory paradigm, there are no assumptions made on the space of $\mathcal{X} \times \mathbb{C}$, when constructing the classifier $g(x)$. However, some assumptions are made about the mechanism that generates the training example set and they are:

[1] There exists a joint probability distribution P on $\mathcal{X} \times \mathbb{C}$, which is unknown but fixed.

[2] Examples in the training set are independent to each other.

This is different from the Fisher's paradigm where classification is done by using maximum likelihood estimation (Fisher discriminant analysis) based on the normality

assumption (Vapnik, 2000).

1.3 Classification trees

A classification tree is a tree structured classifier. The typical structure of a binary classification tree (discussed in Section 1.4) is shown in Figure 1.1. This tree classifies points as being either Red or Blue and uses two feature variables (X_1 and X_2). For this illustration, assume that X_1 and X_2 are normalised such that $0 \leq X_i \leq 1$ for $i = 1, 2$. A DT consists of terminal nodes and non-terminal nodes. Nodes which

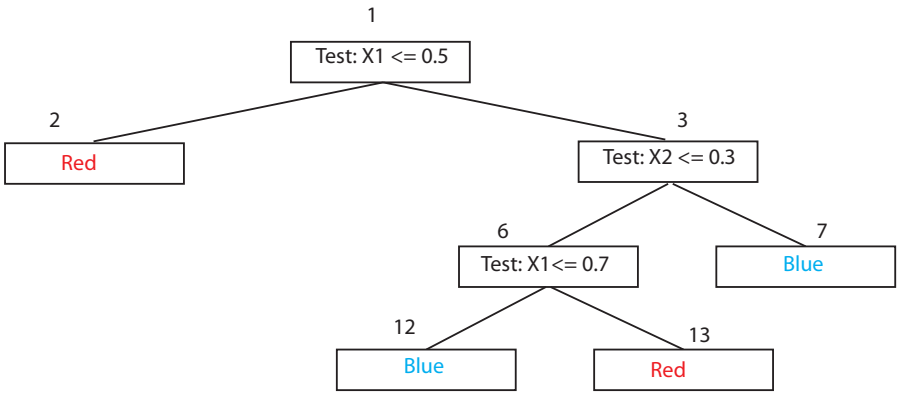


Figure 1.1: Basic structure of a classification tree.

have no lower order nodes (child nodes) are called terminal nodes. A node in a tree represents a sub-region in the feature space. The very first node is called the *Root Node*. At each non-terminal node a test or query is carried out. In Figure 1.1, those nodes are labeled as "Test" and are numbered as $1, 3,$ and 6. The test (or split) can use a single feature variable or combination of feature variables as given in Section 1.5. The predicted class of an example is the class label given to the terminal node to which the example is assigned by the tree. In Figure 1.1 terminal nodes are labeled as either "Red" or "Blue". The route which the example travels from root node to

its terminal node is called the path. The space partitions corresponding to the above DT are given in Figure 1.2.

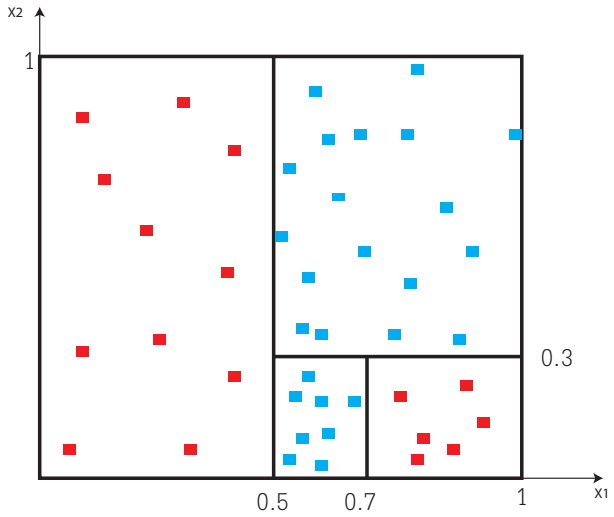


Figure 1.2: Feature space partitions.

In the DT shown in Figure 1.1, the split at node 1 divides the data set into two subsets based on $X_1 \leq 0.5$. This split is shown in Figure 1.2 by the line $X_1 = 0.5$. Examples whose X_1 value is less than 0.5 go to node 2 and are categorised as "Red". The feature space partition corresponding to node 2 is shown in Figure 1.2 by the rectangular region below $X_1 = 0.5$. The examples whose X_1 value is greater than 0.5 go to node 3 and require further splitting. The set of examples come to node 3 and that satisfy $X_2 > 0.3$ are sent to node 7 and are assigned to the Blue class. The region corresponding to node 7 is shown in the rectangular region bounded by $X_1 > 0.5$ and $X_2 > 0.3$ in Figure 1.2. Examples which do not reach node 7 go to node 6 and are tested on X_1 to categorise as "Red" or "Blue". The rules generated by the DT are given below:

Rule 1 : If $X_1 \leq 0.5$, then Class = Red.

Rule 2 : If $X_1 > 0.5 \wedge X_2 \leq 0.3 \wedge X_1 \leq 0.7$, then Class = Blue.

Rule 3 : If $X_1 > 0.5 \wedge X_2 \leq 0.3 \wedge X_1 > 0.7$, then Class = Red.

Rule 4 : If $X_1 > 0.5 \wedge X_2 > 0.3$, then Class = Blue.

In summary, DTs represent a disjunction of conjunctions of tests on feature variables. Each path from the root node to a terminal node corresponds to a conjunction of tests and the tree itself a disjunction of these conjunctions.

1.4 Categorization of classification trees

Tree structured classifiers are categorised in various ways based on characteristics of a tree induction procedure. Some of the characteristics which this thesis is interested in are about the way a tree is induced, the type of the splits used and the number of child nodes per non-terminal node. A brief account of these is given in the following sections.

1.4.1 Axis-parallel versus oblique trees

Trees which use a single feature variable to split regions are called axis-parallel trees. On the other hand, trees which use a linear combination of feature variables to split regions are called oblique trees. Axis-parallel splits are suitable when the class boundaries are parallel to the feature axes. Oblique splits are useful when the class boundaries can be represented as linear combinations of feature variables. An oblique split is a generalisation of an axis-parallel split. Finding oblique splits can be more computationally expensive than searching for axis-parallel splits (Heath, Kasif, & Salzberg, 1993). However, many studies have shown that trees which use oblique splits generally produce smaller trees with better accuracy compared with axis-parallel trees for some problem domains (Brodley & Utgoff, 1995; X. B. Li et al., 2003; Murthy, Kasif, & Salzberg, 1994). Axis-parallel splits can easily be understood, but these trees are

in general larger in size (many nodes) and hence, understanding the entire tree would be difficult. On the contrary, oblique trees may result in shorter trees (fewer nodes), yet an individual split may be difficult to understand (Brodley & Utgoff, 1995).

1.4.2 Binary versus non-binary trees

In binary trees a region is split into two mutually exclusive sub-regions whereas in non-binary trees (Utgoff & Brodley, 1991) a region is split into more than two mutually exclusive sub-regions. Generally splits based on qualitative feature variables, which have more than two levels, produce non-binary partitions. Since non-binary trees can make many partitions at a node, the size of the example set decreases rapidly when going down the tree. However, a non-binary tree can be reduced to a binary tree (Devroye, Györfi, & Lugosi, 1996). Binary trees are popular because they can easily be interpreted (especially with axis-parallel splits) as it is a matter of answering a query of only two possible answers (yes and no) at each node. Furthermore, a split at a node in a binary tree can generally be formalised by a one dimensional optimisation problem whereas in a non-binary tree it would be required to solve an optimisation problem having dimension more than one (Duda, Hart, & Stork, 1999). In this research, a new oblique binary classification tree induction algorithm is proposed. Hence the discussion is mostly limited to binary trees.

1.4.3 Top-down versus bottom-up

In the top-down binary tree induction approach, the first split is made such that the whole training example set is partitioned into two mutually exclusive sets. Then for each subset, a split is made to divide the subset into two further mutually exclusive subsets. This process is carried out until a stopping condition is met (see Section 1.5). In the bottom-up approach, first, terminal nodes are identified using

a clustering algorithm where each terminal node contains examples from one class. Then the clusters are merged one-by-one until one cluster, the root node, reached.

1.5 Top-down tree induction methodology

Consider a classification problem with a training data set $\mathfrak{D}(X, Y)$ where $X = \{X_i \in \mathbb{R}^p : i = 1, 2, \ldots, N\}$. The objective is to find a tree classifier such that:

$$T(X_i) : \mathbb{R}^p \rightarrow \mathbb{C}.$$

DT induction methods recursively partition the feature space $\mathbb{R}^p$ into disjoint sub-regions until each sub-region becomes homogeneous or near homogeneous with respect to a particular class in $\mathbb{C}$. A sub-region (or a node) can be partitioned using splits of one of three forms. The three forms are:

[1] Is $X_j \leq s$?
This test is based on one feature variable and is called a univariate split or axis-parallel split. This creates hyper-rectangular partitions in the feature space.

[2] Is $a_1X_1 + a_2X_2 + \cdots + a_pX_p \leq s$?
This test is based on a linear combination of feature variables and is called an oblique split. This creates polyhedral partitions in the feature space.

[3] Is $\Psi(X_1, X_2, \ldots, X_p) \leq 0$? where $\Psi(.)$ is an any function of $X_1, X_2, \ldots, X_p$.
This general form allows for a non-linear combinations of feature variables. Tests [1] and [2] are special cases of this form.

where $s, a_i, X_j \in \mathbb{R}$. Tests [1] and [2] are the most common types of tests used in DT induction, for example: Breiman et al. (1984), Murthy et al. (1994) and Amasyah and

Ersoy (2008). The test [3] is not popular due to its complicated solution structure. However, several attempts have been made to induce DTs using tests of this form (Ittner & Schlosser, 1996). A major drawback of non-linear splits is that they need large example sets for training and tend to over-fit in the lower nodes of the tree (Y. Li, Dong, & Kothari, 2005). Also the computational effort for finding such splits is much higher than that of the first two forms of tests.

Once the split is made at a node, it generates two child nodes each of which is either homogeneous (or near homogeneous) with respect to a particular class or heterogeneous. If a child node is heterogeneous, then it is split further. This procedure is recursively applied to a node until at least one of the following conditions is met:

[1] All examples in the node belong to a one class (homogeneous node).

[2] The number of examples in the node is less than a user specified threshold.

[3] Misclassification rate at the node is less than a user specified threshold (near homogeneous).

These conditions are called ***stopping rules***. If splitting stops due to [2] or [3], then the resultant child nodes are not necessarily homogeneous. A node which meets any of above conditions is called a terminal node and is given a class label based on a criterion. The most common criterion is to use majority rule. In the majority rule, the class label of the most frequent class in the node is assigned to the terminal node. However, if there exist a set majority classes, the terminal node is arbitrary given the label of the lower indexed element of the set. The tree building process finishes if there are no nodes that require further splitting. A tree in which all the terminal nodes are homogeneous with respect to a class, is said to be a fully grown tree. A fully grown tree generally over-fits to the training data (Breiman et al., 1984). Examples that can easily be classified reside near the root node. The examples that are harder to

classify cause the tree to go further and can produce an over-fitted tree (Manwani & Sastry, 2012). Over-fitted trees are not suitable for predictions (Breiman et al., 1984, p 61) and hence, tree simplification procedures are applied to reduce over-fitting. This process is called tree pruning. The aim of pruning is to obtain a smaller tree from the full tree by eliminating its lower branches that are considered unreliable, based on misclassification rate. Several tree pruning methods are available and one used in this study is discussed in Section 1.7. The top-down tree building process finishes with the pruning process completed.

1.6 Impurity measures

Impurity measures play a vital role in DT induction. They measure the purity or impurity of a node based on the class probability distribution of the node. Several different mathematical measures of impurity have been proposed. According to the literature, the performances of impurity measures on classification results vary with the DT algorithm. Breiman et al. (1984, p. 38) found that the properties of the final tree are insensitive to the choice of impurity measure for CART. Mingers (1989b) found the choice of measure affects the size of a tree but not its accuracy for the decision tree called, ID3 (Quinlan, 1986). In this section, the impurity measure given by Breiman et al. (1984) is defined and some impurity measures which are commonly used in tree induction algorithms are introduced.

Definition 1.6.1. An impurity function is a function ϕ defined on the all set of $\mathrm{C} - tuples$ of numbers $(p_1, p_2, \ldots, p_{\mathrm{C}})$ satisfying $p_j \geq 0,\ j = 1, 2, \ldots, \mathrm{C}$ and $\{\sum_{j=1}^{\mathrm{C}} p_j = 1\}$ with the properties:

[1] The maximum of ϕ occurs only at the point $(\frac{1}{\mathrm{C}}, \frac{1}{\mathrm{C}}, \ldots, \frac{1}{\mathrm{C}})$.

[2] The minimum of ϕ occurs only at points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)$.

[3] ϕ is a symmetric function of $(p_1, p_2, \ldots, p_C)$.

The impurity measures which are used in this thesis are given below. The other criteria can be found in Rokach (2008).

[1] Information measure

The use of the information measure for DTs was proposed by Quinlan (1986) and the function is based on information theory. This is also called as Shanon Entropy. The function measures the impurity at node t by:
$I(t) = -\sum_{k=1}^{C} p_k \log_2 p_k$ where $\log 0 = 0$.

[2] Gini Diversity Index (GDI)

GDI was proposed for DTs by Breiman et al. (1984) and measures the impurity of a node t as:
$I(t) = 1 - \sum_{k=1}^{C} p_k^2$.

[3] Twoing Criterion

This measure was also proposed by Breiman et al. (1984). The distinct feature of this measure is that it directly computes how good the split is. It computes the difference between the purity of a node (R_t) before it is split and the purity of two sub-regions obtained after splitting R_t and it is given by:
$\Delta I(t) = \frac{p_L p_R}{4}[\sum_{k=1}^{C} |p(k|t_L) - p(k|t_R)|]^2$
where p_L and p_R are the proportions of examples that fall into the left and right nodes respectively and $p(k|t_L)$ and $p(k|t_R)$ are the proportions of the k^{th} class examples that fall into the left and right nodes respectively.

[4] Max Minority

Max Minority can be found in Murthy et al. (1994) and the impurity at a node

t is define as:

$Minority\_Left = \sum_{k=1,k\neq MaxL_k}^{C} L_i$

$Minority\_Right = \sum_{k=1,k\neq MaxR_k}^{C} R_i$

$MaxMinority = \max(Minority\_Left, Minority\_Right)$

[5] Sum Minority

This measure can also be found in Murthy et al. (1994) is just the sum of $Minority\_Left$ and $Minority\_Right$ which as defined in [4].

1.7 Tree pruning algorithms

The purpose of tree pruning is to avoid over-fitting. A DT can be pruned while it is being built or after the tree is fully grown. The former is known as pre-pruning while the latter is known as post-pruning. The second and third stopping rules mentioned in Section 1.5 are examples of pre-pruning. However, it is recommended to use post-pruning rather than pre-pruning (Breiman et al., 1984, p 62). Pre-pruning stops a node from being split further when a user specified threshold is met. This may inhibit the chance a desirable split being found at a descendent node. Post-pruning methods allow the tree to be grown fully and then the tree is pruned upwards. Some post-pruning methods are listed below:

[1] Minimal Cost-Complexity pruning (Breiman et al., 1984).

[2] Reduced Error pruning (Quinlan, 1987).

[3] Pessimistic pruning (Quinlan, 1987).

[4] Critical value pruning (Mingers, 1987).

Quinlan (1987) and Mingers (1989a) show that pruning methods produce smaller and more accurate DTs. Comparisons between different pruning methods can be

found in Esposito, Malerba, Semeraro, and Kay (1997); Malerba, Esposito, and Semeraro (1996); Mingers (1989a); Patil, Wadhai, and Gokhale (2010); Quinlan (1987). Based on the empirical comparisons between various pruning methods, Mingers (1989a) concludes that in general the minimal cost complexity pruning consistently produces lower error rates. In this research, the minimal cost-complexity pruning, introduced by Breiman et al. (1984) is used. This method is widely used in DT induction procedures and the following subsection briefly explains how the minimal cost-complexity pruning algorithm works.

1.7.1 Minimal cost complexity pruning

Minimal Cost Complexity pruning (MCC-Pruning) prunes a tree in two stages. First, it generates a sequence of sub-trees from a fully grown tree and then selects the smallest tree with the highest classification accuracy estimated on an independent test set or cross validation samples. The first stage uses the complexity measure:

$$R_\alpha(T) = R(T) + \alpha \left|\tilde{T}\right|, \tag{1.7.1}$$

where $\tilde{T}$ is the set of terminal nodes in the tree T, $\left|\tilde{T}\right|$ is the cardinality of $\tilde{T}$ and $\left|\tilde{T}\right| \geq 1$, $R(T)$ is the re-substitution estimate[2] (RSE) of misclassification rate of the tree T and $\alpha \geq 0$ is a scalar. Therefore, the complexity measure is a function of RSE of misclassification rate and number of terminal nodes in the tree. Consider a branch T_t rooted from node t. The cost complexity measure of T_t is defined by:

$$R_\alpha(T_t) = R(T_t) + \alpha \left|\tilde{T_t}\right|, \tag{1.7.2}$$

where $\left|\tilde{T_t}\right| > 1$. If all the branches of T_t are pruned, then $\left|\tilde{T_t}\right| = 1$ and the cost complexity measure of node t can be defined as:

$$R_\alpha(t) = R(t) + \alpha. \tag{1.7.3}$$

[2]definition is given in Section 1.9

Also[3]:

$$R_\alpha(T_t) < R_\alpha(t), \text{ when } \alpha = 0 \qquad \because R(T_t) < R(t) \tag{1.7.4}$$

When α increases, both $R_\alpha(T_t)$ and $R_\alpha(t)$ increase. The branch T_t remains in the tree as long as $R_\alpha(T_t) < R_\alpha(t)$. However, the rate of increase in $R_\alpha(T_t)$ is greater than that of $R_\alpha(t)$. Therefore, when α increases to α^* where $\alpha^* = \frac{R(t)-R(T_t)}{\left|\tilde{T}_t\right|-1}$, the two cost complexities become equal ($R_\alpha(T_t) = R_\alpha(t)$). Since, the two complexities are equal and node t is just a one single node, it is preferable to have node t in the tree instead of having the entire T_t rooted from the node t. Therefore, T_t can be pruned from the tree at $\alpha = \alpha^*$.

The MCC-pruning method can be described as follows. First, it considers the full tree, T_{max}. For each non-terminal node in T_{max}, α^* is computed and the non-terminal node (t_1^*) is selected which minimises the α^*. Then, the branch $T_{t_1^*}$ rooted from t_1^*, is pruned. The resultant sub-tree then can be defined as $T_1 = T_{max} - T_{t_1^*}$. Now, for each non-terminal node in T_1, α^* is computed and the non-terminal node (t_2^*) is selected which minimises α^*. The branch $T_{t_2^*}$, rooted from t_2^*, is then pruned from T_1 and the next sub-tree is defined as $T_2 = T_1 - T_{t_2^*}$. The algorithm continues to prune branches until the final sub-tree contains only the root node. The explicit algorithm for the first stage of the MCC-pruning is given in Algorithm 1.

The algorithm outputs a sequence of decreasing size sub-trees $T_1 \succ T_2 \succ \ldots \{\textit{root node}\}$. In the second stage the objective is to select an optimal sub-tree (a sub-tree which minimises the estimated misclassification rate) from the sequence. Two approaches are suggested to estimate the misclassification rate by Breiman et al. (1984): (a) the use of the independent test sample, and (b) the use of cross validation samples. In this research, the independent test sample approach is used to estimate the misclassification rate of the tree to select the optimal sub-tree. Hence, the same independent test set (or pruning set) is fed through each T_i and the smallest

[3]see (Breiman et al., 1984, p 68)

initialization;
Define T =Tree;
Define T_{max} = The fully grown tree;
Define $i = 1$ ▷ Temporary counter;
$T = T_{max}$;
while $T \neq$ *Root Node* **do**
  For each non-terminal node in T, compute its α value;
  Select the branch $T_{t_i^*}$ which produces the smallest α (that is α^*) and prune it from the T;
  Let $T_i = T - T_{t_i^*}$;
  $i = i + 1$;
  Let $T = T_i$;
end

Algorithm 1: Overview of Minimal Cost-Complexity pruning algorithm - First stage

tree that minimises the misclassification rate on the independent test set is selected. Furthermore, Breiman et al. (1984) introduces c-standard error (c-SE) rule to select the optimal sub-tree. Let q^* be the estimated minimum misclassification rate. The standard error of q^* is $\sqrt{q^*(1-q^*)/n_{ts}}$ where n_{ts} is the size of the test sample. The c-SE rule selects the optimal tree, T_{k_0} where k_0 is the maximum k satisfying $R(T_{k_0}) \leq q^* + c\sqrt{q^*(1-q^*)/n_{ts}}$. In this research a 0-SE rule is used.

1.8 The best tree

As in the conventional modelling approach, the aim is to obtain the most parsimonious classification tree model for the data. That is, to obtain the **smallest** tree that has the **minimum true misclassification rate** $R^*(T)$ (see the Definition 1.8.1). The size of the tree is measured in terms of the number of terminal nodes in the tree. We call this tree *"the best tree"*.

Definition 1.8.1. Let (X, Y), where $X \in \mathbb{R}^p$ and $Y \in \mathbb{C}$, be an example

drawn randomly from a population and independent of the training set. Then $R^*(T) = p(T(X) \neq Y)$.

1.8.0.1 Finding the best tree

The procedure for finding the best tree from a learning sample can be summarised as follows:

[1] Draw a random sample from the population concerned.

[2] Construct all possible trees for the sample.

[3] Classify a very large (virtually infinite) sample from the population (or ideally the population itself) using each tree and obtain $R^*(T)$ for each tree [4].

[4] Select the smallest tree which minimises the $R^*(T)$.

Finding all possible trees for the sample is an impracticable task. Therefore, *greedy divide and conquer algorithms* are used to *approximate* the best tree. The principle of divide and conquer algorithms is to partition the feature space recursively into sub-regions until each sub-region satisfies at least one of the stopping rules mentioned in Section 1.5. The problem is how to divide the feature space. It can be divided either using axis-parallel splits or oblique splits. For a particular region having n, p-dimensional examples, the total number of all possible axis-parallel splits is $(n-1)p$. However, it cannot be predetermined which split leads to the Best Approximate (BA) tree. Therefore, the best split in the region (*locally optimal*) is used to split the region. This approach is called the *greedy approach*. The problem now reduces to how the best split is selected in each region. A popular method is searching for the split that minimises an impurity function. It can be shown that the impurity of a region before

[4] In practice, $R^*(T)$ usually is unobservable. Therefore, the estimated misclassification rate $R(T)$ is used and the estimation procedure of $R^*(T)$ is given in the Section 1.9.

it is split will be greater than, or equal to, the weighted sum of impurities of the two sub-regions generated after the split (Breiman et al., 1984, p. 126). Therefore, the split which maximises the reduction in impurity, defined in equation (1.8.1), is selected as the best split at the node.

$$\Delta I(t) = I(t) - p_L I(t_L) - p_R I(t_R), \tag{1.8.1}$$

where $I(t)$ is the impurity of node t and p_L and p_R are proportions of examples in left and right child nodes respectively. Greedy approaches do not guarantee that the induced tree is the best approximate tree. There can be situations where non-optimal splits produce a *better tree* than a tree obtained through the best splits at each node. This is illustrated below using axis-parallel splits.

1.8.1 Greedy methods

Greedy methods find locally optimal splits at each node with the hope of finding the globally optimal tree. The major drawback of this approach is illustrated using an axis-parallel DT. Consider the two dimensional, two-class classification problem given in Table 1.2.

The scatter plot of the data is given in Figure 1.3. It can be seen that axis-parallel splits are suitable for feature space partitioning. Hence, an axis-parallel DT is constructed using the GDI impurity measure. The axis-parallel algorithm starts to search along the X_1 axis and then the X_2 axis to find the best split at the first node. The best split is the one that maximises the impurity reduction given in equation (1.8.1). The impurity reduction for each split along X_1 axis and X_2 axis are given in Figure 1.4 and Figure 1.5 respectively. The impurity reduction, $\Delta I(t)$, of each split along the i^{th} axis is computed for $\frac{X_j^i + X_{(j+1)}^i}{2}, i = 1, 2$ and $j = 1, 2, \ldots, (n-1)$, where X_j^i is the j^{th} value of i^{th} feature which is sorted in ascending order. At $X_1 = 4.55$, the impurity reduction attains its maximum of 0.1636. The line X_1=4.55

Table 1.2: Random sample.

Y	X_1	X_2	Y	X_1	X_2	Y	X_1	X_2
1	1.8	4.9	1	1.5	4.8	1	1.25	5.2
1	3	4.75	1	3.5	5.2	1	1.9	2.7
1	2	2	1	4.1	2.3	1	4.2	2.45
1	2.3	2.65	1	4.5	5.6	1	3.75	2.6
1	2.5	2.5	1	4	2.7	1	3.7	2.4
1	5	4.8	2	1.3	3.4	2	1.5	2.9
2	2.4	3.3	2	2.7	4	2	3	4.2
2	3.9	3.9	2	3.8	2.9	2	4.8	2.9
2	4.9	2	2	5.5	1.7	2	5	2.1
2	6.3	2.6	2	5.7	2.5	2	5.4	1.8
2	4.6	2.1	2	4.8	3.1	2	5.9	3.3
2	5.7	3.2						

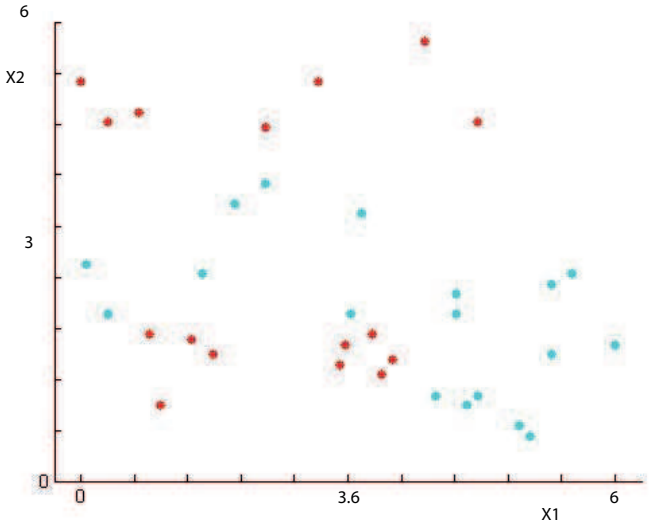


Figure 1.3: Scatter plot of examples-two-class problem.

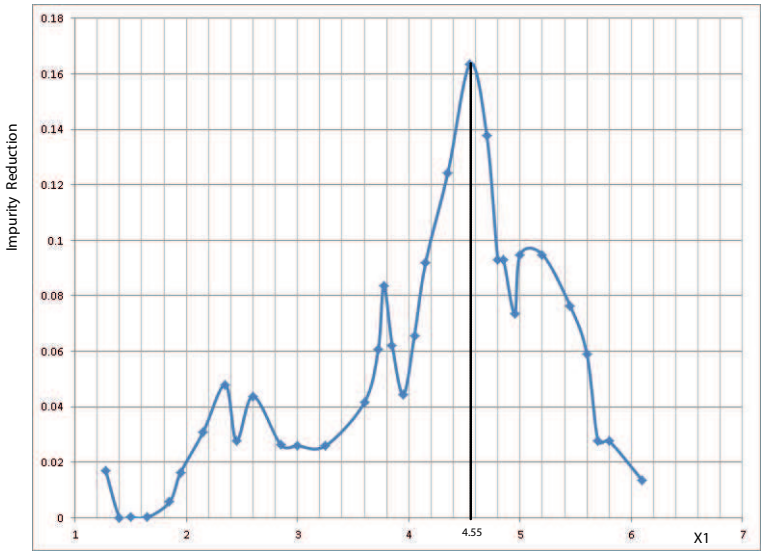


Figure 1.4: Change in impurity with respect to X1.

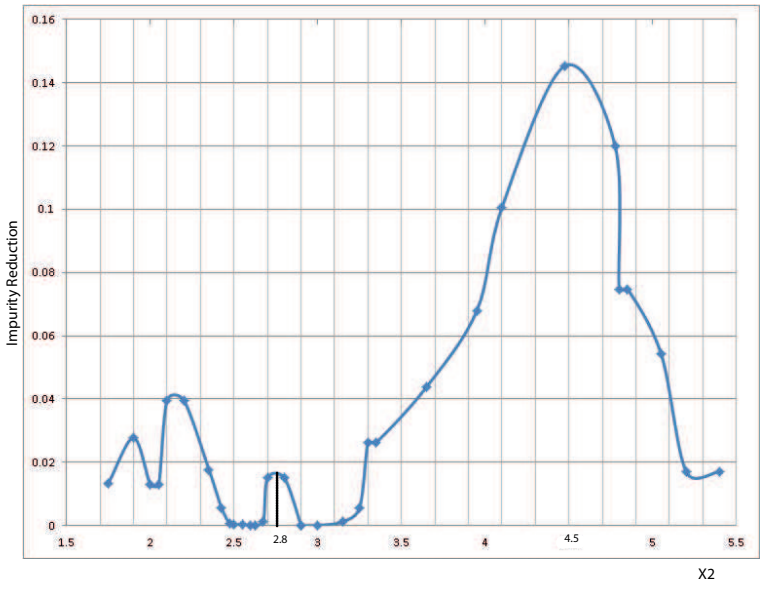


Figure 1.5: Change in impurity with respect to X2.

is selected to split the region first. The tree building process continues as mentioned in the Section 1.5 until the tree is fully grown. The final unpruned tree (GDI-tree) and the corresponding feature space partitions (GDI-Partitions) are given in Figure 1.6 and Figure 1.7 respectively. With reference to Figure 1.6 and Figure 1.7 there are five

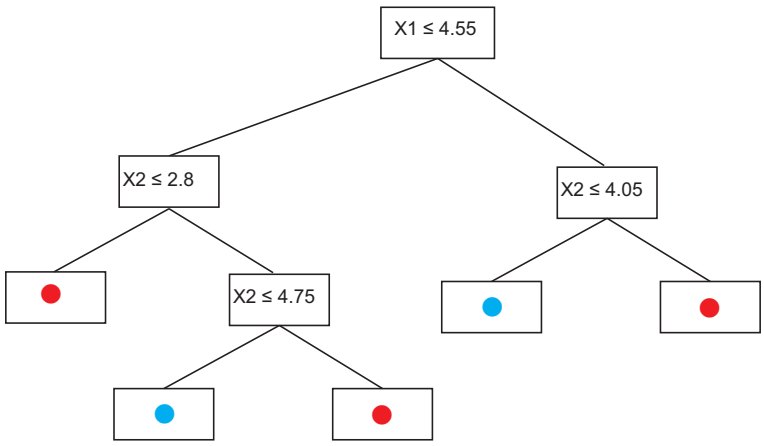


Figure 1.6: GDI DT.

terminal nodes - five homogeneous regions using axis-parallel splits. However, had the split at $X_2 = 2.8$ been chosen as the first split, where the reduction in impurity is only 0.015018 (see Figure 1.5) a smaller tree would have been obtained. The resultant DT (BA-tree) and the corresponding feature space partition (BA-partitions) are given in Figure 1.8 and Figure 1.9 respectively. The BA-tree has only 4 terminal nodes while the greedy method produces a tree with 5 terminal nodes. The BA tree also classifies examples perfectly. Therefore, the BA tree would have been the *best approximated tree* for the *best tree*. This illustrates the fact that non-optimal splits may lead to a better tree. However, to improve greedy algorithms a lookahead approach beyond one or few levels has been proposed by Sarkar, Chakrabarti, Ghose, and DeSarkar

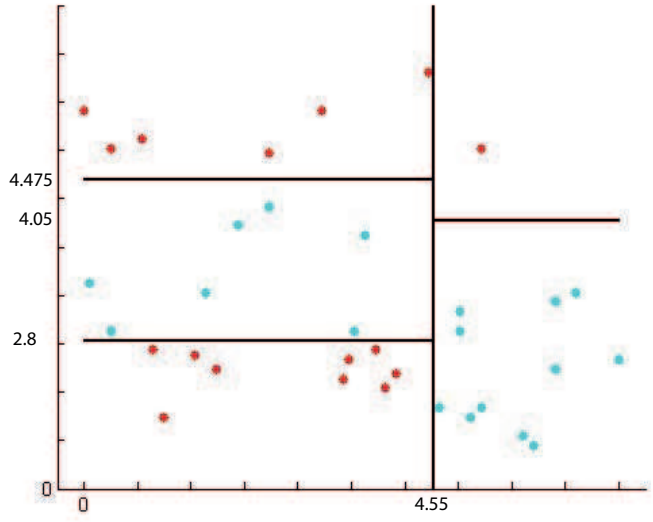


Figure 1.7: Space partition structure for the GDI DT.

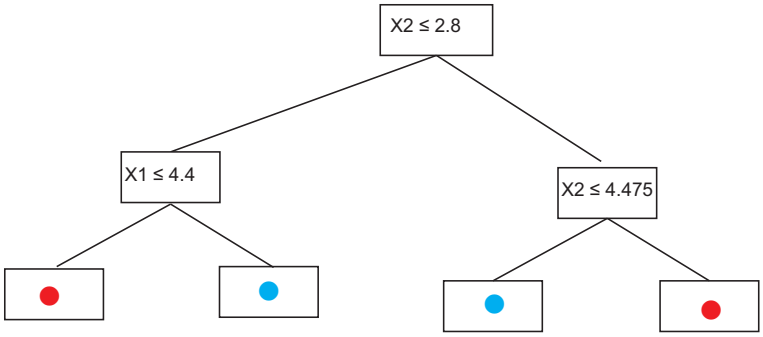


Figure 1.8: BA DT.

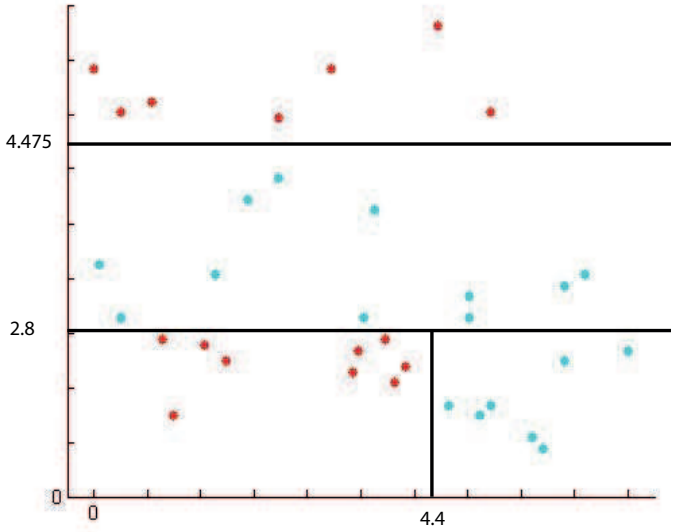


Figure 1.9: Space partition structure for the BA DT.

(1994) and Murthy and Salzberg (1995a).

1.9 Estimation of true misclassification rate, $R^*(T)$

Breiman et al. (1984) present three estimators for $R^*(T)$, the true misclassification rate of the final tree. A brief description of these methods is given below. The indicator function denoted by $I(.)$ takes value one if the argument is true, otherwise it is zero.

[1] Re-substitution estimate $R(T)$

Let the training sample of the tree be given by (X_i, Y_i), where $X_i \in \mathbb{R}^p$ and $Y_i \in \mathbb{C}$, for $i = 1 \ldots n$. Then $R(T)$ is defined as:

$$R(T) = \frac{1}{n} \sum_{i=1}^{n} I(T(X_i) \neq Y_i). \tag{1.9.1}$$

$R(T)$ is an optimistic estimator of $R^*(T)$ as it is computed using the training set. However, if there is no additional sample to use as a test sample, then $R(T)$ is used as an estimator of $R^*(T)$.

[2] Test sample estimate $R^{ts}(T)$

Let the test sample, which is independent of the learning sample but drawn from the same probability distribution as of the training sample, be given by (X_i, Y_i), where $X_i \in \mathbb{R}^p$ and $Y_i \in \mathbb{C}$, for $i = 1 \ldots n$. Then $R^{ts}(T)$ is defined as:

$$R^{ts}(T) = \frac{1}{n} \sum_{i=1}^{n} I(T(X_i) \neq Y_i). \tag{1.9.2}$$

Generally, the independent test sample is obtained by drawing a portion q of examples randomly from the learning sample. Conventionally, 10%, 20% or 30% is assigned to q. Therefore, the training sample should be large enough to guarantee that the tree is reliably trained and tested.

[3] v-Fold Cross-Validation (CV) estimate $R^{(cv)}(T)$

If the sample is not sufficiently large to split into training and testing samples, then a v-fold CV method is preferred. In this procedure, the example set is divided into v nearly equal size (n_v) disjoint partitions randomly and for each partition (hold-out partition), the remaining $v - 1$ partitions are used for the tree building while the hold-out partition, (h_v), is used for the testing. Then $R^{cv}(T)$ is defined as:

$$R^{cv}(T) = \frac{1}{n} \sum_{j=1}^{v} \sum_{(X_i, Y_i) \in h_j} I(T(X_i) \neq Y_i). \tag{1.9.3}$$

The advantage of this method is that it uses the entire sample to estimate $R^*(T)$. Conventionally, 5 or 10 is assigned to v. In the experiments of this research, $R^{(cv)}(T)$ is used to estimate the accuracy of the classifier.

Variants of CV procedure can be found elsewhere in the literature. Each method has its own unique approach to estimate the accuracy and their performances differ depending on the context. For example, Monte Carlo CV is recommended over ordinary CV to determine the number of components in a finite mixture model (Smyth, 1996). A brief description of each of the method is given below.

[1] **Ordinary CV**

As explained above, in v-fold cross validation, the example set is divided into v disjoint partitions. For each v, the model is built using $v-1$ partitions and tested on the remaining partition. The leave-one-out CV is a specific version of ordinary CV where $v = n$. However, the accuracy estimates of leave-one-out CV suffer from high variance (Smyth, 1996) and hence, $v = 5$ or $v = 10$ is the usual choice. However, in some cases leave-one-out is very useful, for example, with time series data as it may not weaker or disrupt the autocorrelation structure (Ancona et al., 2005). The experiments conducted in Chapter 3 use the ordinary CV procedure. However, Varma and Simon (2006) show that the ordinary CV procedure tends to produce optimistic estimates when it is applied to the situation where the parameter estimation[5] is performed simultaneously with the accuracy estimation. This happens as follows. For each parameter value or each combination of the parameter values, the v-fold ordinary CV procedure is used to estimate the accuracy. The optimal parameter value or the optimal combination of the parameter values is then determined by maximising the CV accuracy. Let a_{max} be the maximum accuracy. The full example set is then used to build the classifier with the optimal parameter value or the combination of the parameter values found and the estimated accuracy of the classifier is taken as a_{max}. Krstajic, Buturovic, Leahy, and Thomas (2014) state that choosing a_{max} as the estimated value for the accuracy is a common mistake

[5]parameters which required to train the classifier and they are called tuning parameters.

in classification and regression model selection and assessment. Therefore, in this research (Chapter 5 and Chapter 6) the two-stage CV procedure is used to estimate the accuracy while estimating the tuning parameters of classifiers.

[2] **Monte Carlo CV (MCCV)**

In this method, examples are partitioned into M (usually between 20-50) disjoint test and training subsets where a test set is a fraction β (usually taken as 0.5) of the overall data. The marked difference between v-fold CV and MCCV, is that the M test sets in MCCV are not disjoint. It is assumed that MCCV estimates are generally unbiased as they are computed from the average of M, generally large, estimates (Smyth, 1996).

[3] **Nested or double CV**

This procedure is particularly useful when the model selection is done simultaneously with the parameter estimation. It is noted by Varma and Simon (2006) that ordinary CV in this situation produces optimistic biased classification accuracy estimates. Thus Varma and Simon (2006) proposed nested CV while Filzmoser, Liebmann, and Varmuza (2009) proposed the repeated nested CV procedure also called the repeated double CV. The procedure for the nested CV is given in Appendix B. This procedure is used in the experiments carried out in Chapter 5.

[4] **Two-stage ordinary CV**

This procedure is also used by some researchers when the model selection is done parallel to the parameters' estimation. First, a v_1-fold ordinary CV is run to select the optimum parameter values. Then another v_2-fold CV is run, separately to the first CV, using the optimum values of the parameters to estimate the classification accuracy. This method is used by (Manwani & Sastry, 2012) to estimate the classification accuracy of their classifiers. Some of the

experiments in Chapter 5 and Chapter 6 use this CV procedure to estimate the classification accuracy.

1.10 Oblique decision trees

One of the advantages of axis-parallel DTs is that they are conceptually simple, in that, the class determination phenomenon can easily be understood. The axis-parallel splits are desirable when the class boundaries are parallel to feature axes. However, if the decision boundaries are not parallel to feature axes, axis-parallel splits can complicate the boundary structure, even in simple problems. This fact is illustrated in Figure 1.10, where two-classes are classified using the information contained in two feature variables.

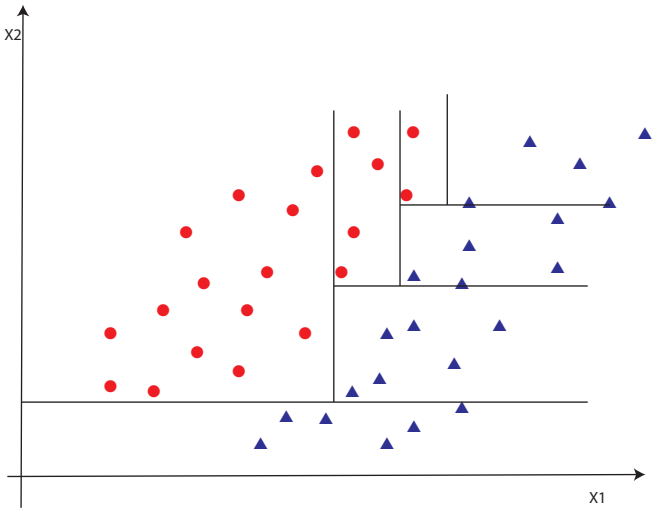


Figure 1.10: Induced axis-parallel splits for a training example set with a class boundary that is not parallel to either feature axes.

The oblique split simplifies the boundary structure (see Figure 1.11) and hence,

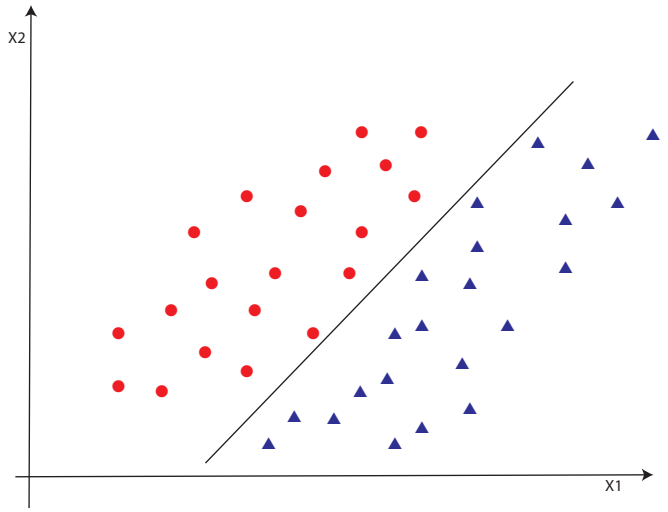


Figure 1.11: Induced oblique split for the training data in Figure 1.10.

simplifies the tree. Oblique splits also help in increasing the prediction accuracy of the tree model. However, the interpretablility of the solution can be difficult with oblique splits (X. B. Li et al., 2003). The major problem of inducing oblique trees is the computational effort required to find the best split at a node. The time complexity of axis-parallel and oblique splits is discussed in the following section.

1.10.1 Time complexity of decision tree induction

Time complexity of DT induction algorithms is one of the important characteristics when assessing DTs. Even with the recent development of computer power, some algorithms take much longer to build a DT with large training samples. Therefore, Lim, Loh, and Shih (2000) state that there may be an advantage of using quicker algorithms for large example sets even if the slower algorithms possess slightly better classification accuracy.

The complexity of DT induction algorithms has been analysed in many ways. Hyafil and Rivest (1976) studied the complexity for constructing the optimal binary tree while Murthy et al. (1994) derive the upper bound for the complexity of the best split at a node. Others, Amasyah and Ersoy (2008) for example, have considered the total training time of the DT.

Here the time complexity of finding the best axis-parallel and the best oblique split is discussed. Assume a set of n, p-dimensional examples. The best split is the split which maximises the impurity reduction given in equation (1.8.1). The best axis-parallel split can be found by exhaustively searching all possible axis-parallel splits at the node. Hence, the impurity function has to be evaluated at $(n-1)$ positions along each dimension giving a total number of evaluations of $p(n-1)$. Furthermore, at each evaluation $O(n)$ work has to be done to evaluate the impurity function. Therefore, the order of complexity for finding the best split at a node by axis-parallel method is $O(n^2p)$, if the examples are not sorted. However, Murthy et al. (1994) show that, for the same configuration, the order of complexity for finding the best oblique split at a node is $O(2^p n\binom{n}{p})$. Heath et al. (1993) show that the problem of finding an optimal oblique split is NP-complete and hence, no polynomial time algorithm is possible for reasonable values of n and p. Another way of looking at the complexity of inducing oblique DT is given below.

Searching for the *"Best axis-parallel Tree"*[6] can be interpreted as finding the globally optimal solution. Axis-parallel algorithms **approximate the global solution** through a series of local optimum solutions. Here, the local optimum solution is referred to as the optimum split at a node. However, as seen in Figure 1.4, the optimal split at the node is chosen such that it **maximises the node level impurity reduction function globally**[7].

[6] if the class boundaries are parallel to feature axes.

[7] Section 1.8 shows that this may not be an ideal split

Oblique DTs are also induced under the same mechanism where the *"Best Oblique Tree"* is **approximated** by maximising the impurity reduction function at each node. However, an exhaustive search is impossible due to the large number of possible oblique splits. Therefore, the search for the node level global maximum of the impurity reduction function is conducted using various optimisation algorithms which do not necessarily guarantee convergence to the global solution. Hence, in contrast to axis-parallel splits, the oblique split at a node is usually only an approximation to the best oblique split at the node.

1.11 Motivation for heuristic methods

Since, searching for the best oblique split at a non-terminal node is harder, inducing the best oblique tree can be extremely difficult. However, researchers have derived with various techniques to induce oblique DTs. These techniques differ in the way they search for the best split at a node. Some of the algorithms use optimisation methods[8] (Breiman et al., 1984) and (Heath et al., 1993), some use heuristic arguments (Amasyah & Ersoy, 2008) and (Manwani & Sastry, 2012), while other use genetic and evolutionary methods (Cantu-Paz & Kamath, 2003) and standard statistical techniques (Gama & Brazdil, 1999). Each of these methods has its own benefits and drawbacks and detailed explanations of the methods which are important to this research are given in Chapter 2. We define the approach of a heuristic argument as follows:

Definition 1.11.1. In the heuristic approach the structure of class boundaries is assumed before observing the example set.

[8]Hill climbing, Simulated annealing etc

If the structural assumption in Definition 1.11.1 is true, DTs based on heuristic arguments produce accurate and smaller trees. In the recent development of the oblique DT algorithm methodologies, it is evident that DTs based on heuristic arguments (Amasyah & Ersoy, 2008; Manwani & Sastry, 2012) have gained considerable attraction compared with the DTs based on optimisation techniques. Results of these methods, in general, show there is no apparent difference between the average accuracy and average tree size compared with the results of DTs induced using optimisation algorithms. However, the computer intensiveness of heuristic algorithms is usually lower than the optimisation algorithms. Moreover, Hyafil and Rivest (1976) stated that there is no efficient algorithm for constructing optimal binary trees and hence, they were motivated to find efficient heuristics for constructing near-optimal DTs. Following these recent developments, in this research, a new heuristic algorithm is proposed to induce oblique DTs.

In summary, axis-parallel splits are computationally inexpensive but often produce complicated trees. On the other hand, oblique splits can produce simple trees but are computationally expensive. In this research, our aim is to find an alternative approach to produce computationally inexpensive oblique trees that are accurate and have relatively small trees.

1.12 Thesis overview

1.12.1 Objectives of the study

[1] It is evident that it is much harder to find an oblique split than an axis-parallel split that minimises impurity function. Because the search for the optimal split is time consuming, the time taken to generate a full tree can be excessive. In this research, the first objective is to present a heuristic algorithm (HHCART)

to construct a computationally efficient oblique DT. The approach is to use axis-parallel splits, the simplest form of splits, in transformed feature spaces. These splits will be oblique in the original space. Transformed spaces are constructed using Householder matrices defined on the eigenvectors of classes' covariance matrices. This approach avoids searching for oblique splits in the original feature space.

[2] Data mining applications often come with massive example sets and inducing oblique DTs for such example sets often consumes considerable time. As the second objective, the HHCART algorithm is modified in two ways to handle massive example sets namely: (a) disk resident algorithm, and (b) parallel computing algorithm.

[3] The HHCART algorithm can use alternative vectors in place of eigenvectors of classes' covariance matrices to construct transformed feature spaces. These alternative vectors can be found in other DT methods provided in the literature. Therefore, we explore the possibilities of incorporating these alternative vectors into the HHCART algorithm and thereby investigate their performances on classification tasks.

[4] Even though top-down designs have been predominately used for DT induction, recently a Bottom-Up method has been proposed. This method uses cluster analysis followed by Support Vector Machine (SVM) to determine separating hyperplanes. We explore this methodology by replacing the SVM approach with the hyperplane generation method in the HHCART algorithm.

1.12.2 Structure of the thesis

Chapter 2 of the thesis is dedicated to the literature review on DT induction algorithms. A detailed description is given on algorithms, some of which are used in this thesis to compare the proposed methods. A brief summary of other DT methods are presented to fulfil our literature survey.

Chapter 3 presents the proposed methodology. The complete description of the proposed heuristic argument and the detailed algorithm are given. This chapter also includes the derivation of the time and space complexities of the proposed methodology. Finally, the performance of the method is compared to some benchmark DT methods in the literature.

Chapter 4 shows how the proposed method can be modified to work with massive example sets. First, the disk resident version of the HHCART algorithm is introduced and then the parallelised version of HHCART is introduced to work in a parallel computing environment.

Use of alternative vectors in the proposed method is given in Chapter 5. Some of the alternative vectors are obtained from existing DT methods. Hence brief descriptions of those methods are also given in Chapter 5. Classification results on real data sets are obtained and compared with some existing DTs. Moreover, some alternative methods require tuning parameters to be estimated prior to the tree building. In this study, two CV methods are used for estimation and the properties of two estimators are discussed.

Chapter 6 introduces the bottom-up tree construction of the proposed method. Brief descriptions of model based clustering methods and the Expectation-Maximisation (EM) algorithm are given for proper understanding of the existing bottom-up tree induction approach. The existing Bottom-Up Tree Induction Framework (BUTIF) uses Support Vector Machine (SVM) to find separating hyperplanes. Hence, SVM is also introduced.

Chapter 7 reviews the thesis with a discussion and a conclusion.

1.13 Thesis outcomes

[1] The novel DT algorithm presented in the Chapter 3 was submitted to the Special Issues of the Journal of Computational Statistics and Data Analysis under the title of "HHCART: An Oblique DT" and now it is under revision.

[2] The outcome of Chapter 5 is submitted to the journal of IEEE Transactions on Cybernetics Part B under the title of "HHCART with alternative vectors".

[3] A manuscript based on the results of the bottom-up method, which is presented in Chapter 6, is in preparation.

[4] Based on the preliminary results obtained, an abstract was presented at the Canterbury Statistics Open Day 2012 under the title of "Oblique DTs using HouseHolder Reflection".

[5] An abstract was presented on the partial results of Chapter 3 at the Joint Conference of the NZ Statistical Association and Operations Research Society of NZ 2014 under the title of "Oblique DT induction with HHCART". The presentation was mostly focused on highlighting the ability of HHCART to handle both qualitative and quantitative features in the same oblique split.

[6] Based on this study's literature review, an abstract was presented at the Canterbury Statistics Day-Research 2014 under the title of "The Use of DTs in Statistical Data Classification".

Chapter 2

Literature review - oblique decision tree induction algorithms

2.1 Introduction

In this chapter, a concise survey of top-down oblique DT induction methods is presented. The decision tree algorithm presented in this thesis follows the feature space partitioning concept of the CARTopt algorithms (Robertson, Price, & Reale, 2013, 2014) which were specifically designed to find a minimiser of a non-smooth function. A detailed description of partitioning strategy used in CARTopt is given in this Chapter and Chapter 3. Though the other early methods are not directly relevant to the principal methodology proposed in this thesis, some of them are briefly illustrated in this Chapter because:

[1] The results of those methods are used to compare the performance of the proposed methodology.

[2] Some early methods have shown some resemblance to the proposed methodology because in both cases artificial features are created.

[3] It is interesting to show the current trend in the DT induction methodologies.

The common top-down DT induction approach works in two stages. The first stage builds a fully grown tree and the second stage prunes the tree to avoid the over-fitting of the tree. However, some tree induction algorithms use pre-pruning techniques so that pruning is done while the tree is being built. The main task of the tree growing stage is to search for the best split. Choosing a search method for the best oblique split is crucial in DT induction because the time efficiency of the tree building process depends heavily on it. Therefore, in this research the oblique DT induction methods are categorised using the best split search method. Three categories are defined: induction algorithms that use optimisation techniques, standard statistical techniques and those that use heuristical arguments.

2.2 Tree induction methods based on optimisation techniques

CART-LC (Breiman et al., 1984) uses a deterministic hill climbing algorithm to search for the best split at a non-terminal node. At each non-terminal node, the algorithm perturbs each coefficient of the hyperplane until the algorithm finds a split that produces the maximum impurity reduction. To reduce the risk of a local minimum being found, each perturbation starts from three different pre-specified locations. A backward feature elimination process is also carried out to delete irrelevant features from the split. The CART-LC algorithm, implemented in OC1 system (Murthy & Salzberg, n.d.), is used in Chapter 3 for comparison purposes.

Heath et al. (1993) introduced a randomisation approach called Simulated Annealing DT (SADT) which uses the simulated annealing optimisation algorithm to search for the best split. At each non-terminal node, an initial hyperplane is set such that it is not parallel to any feature axis. Next, the algorithm picks one coefficient

at a time randomly and adds a random quantity $\delta : \delta \sim U(-0.5, 0.5)$. The resulting hyperplane is then tested using an impurity measure and if the impurity reduction[1] ΔI is negative, then the new hyperplane split is always accepted. If ΔI is positive, then the new hyperplane is accepted with a probability $e^{-\Delta I/T}$, where $T > 0$. Initially T is set large so that when ΔI is small compared with T, the probability of accepting a worse hyperplane is approximately equal to 1. However, T is gradually decreased and hence, the probability of choosing a worse hyperplane tends to zero. This process is repeated until there is no further reduction in impurity. A distinct feature of this algorithm is that a series of locally optimal decisions is not necessarily made. Accepting a worse split from time to time can potentially lead to a globally optimal tree. The main disadvantage of the algorithm is the time taken to find the best split. In some cases it may require the evaluation of the tens of thousands of hyperplanes before finding an optimal split (Murthy et al., 1994).

Murthy et al. (1994) combine the concept of CART-LC and SADT to introduce a new oblique DT methodology called OC1. First, it uses a deterministic hill climbing algorithm to perturb the hyperplane until a local minimum of an impurity function is found. The hyperplane is then perturbed randomly to potentially leave the local minimum. These two steps are performed several times. Each time, the algorithm starts with a different initial guess. One of the initial guesses is the best axis-parallel split. Random hyperplanes are also used as initial guesses. Since each initial guess potentially converges to a different hyperplane, the one that maximises the impurity reduction is taken as the splitting hyperplane. Murthy et al. (1994) show that the time complexity at each non-terminal node for OC1 in the worst case scenario is $O\left(pn^2 \log n\right)$ provided that Max-Minority or Sum-Minority impurity measures are used. For other functions, obtaining a similar upper bound is an open question

[1] $\Delta I = I(\text{New-Hyperplane}) - I(\text{Old-Hyperplane})$

```
Define ΔI                          ▷ Impurity reduction (see Section 1.8.1) ;
Define a_i: The i^th coefficient of the hyperplane. ;
Define h_0: The best axis-parallel hyperplane at node t. ;
Δ(I) = Impurity reduction due to h_0.;
for J=1:20 do                      ▷ Restart loop, OC1 default is 20 restarts: Loop4
    repeat                                                          ▷ Loop3
        while ΔI > 0 do                                             ▷ Loop2
            for i=1:p do                                            ▷ Loop1
                Perturb a_i using the hill climbing algorithm. ;
            end
            Compute ΔI of the split found from the hill climbing algorithm.;
        end
        Let h_j1 be the hyperplane given by the hill climbing algorithm. ;
        Perturb h_j1 to a random direction, say h_j2. ;
        Compute ΔI = ΔI(h_j2) − ΔI(h_j1).
    until ΔI < 0;
    h_j = The best hyperplane found at the j^th iteration.;
    ΔI_j= Impurity reduction of h_j.;
end
h_t = h_{argmax_j ΔI_j}
```

Algorithm 2: Overview of the OC1 algorithm at a single node

(Murthy et al., 1994). Furthermore, the amount of work that OC1 does at a non-terminal node is mostly depends on the number of times that OC1 evaluates the impurity function to find the best split at a non-terminal node. The number of impurity function evaluations made by OC1 can be derived as follows. Consider the basic split finding algorithm of OC1 given in Algorithm 2.

[1] OC1 starts with the best axis-parallel split. Hence, the number of impurity function evaluations to find the best axis-parallel split is np.

[2] Loop1 (the inner-most loop) in Algorithm 2: To perturb one coefficient, the impurity function has to be evaluated n times[2] hence, for p coefficients the total number of evaluations is np.

[2] see Murthy et al. (1994)

[3] Loop2 in Algorithm 2: Let γ be the number of times that OC1 executes Loop2. Hence, the total number of impurity function evaluations at the exit of Loop2 is γnp.

[4] Loop3 in Algorithm 2: Let β be the number of times that OC1 executes Loop3. Hence, the total number of impurity function evaluations at the exit of Loop3 is $\beta\gamma np$.

[5] Loop4 in Algorithm 2: OC1 executes Loop1-Loop3 20 times which is the default value. Hence, the total number of impurity function evaluations at the exit of Loop4 is $20\beta\gamma np$.

Therefore, the total number of function evaluation of OC1 is $20\beta\gamma np$ in the worst case scenario. Moreover, Murthy et al. (1994) state that the number of hyperplane evaluations at most will be n if Max-Minority or Sum-Minority impurity measures are used and this will be equivalent to $20\beta\gamma$. Hence, the total number of hyperplane evaluations required by OC1 in the worst case scenario is n^2p if Max-Minority or Sum-Minority impurity measures are used. The OC1 algorithm is used in the experiments of this thesis.

One feature of both the SADT and OC1 algorithms is that they can construct different DTs on different runs using the same learning sample. Therefore, it is possible to run these algorithms multiple times and pick a tree which produces the minimum misclassification rate. However, realising this advantage is tough when the learning sample contains a large number of examples and features.

2.3 Tree induction methods based on standard statistical techniques

Various oblique DT induction algorithms have been explored using standard statistical techniques. Most of the algorithms use various forms of Fisher's linear discriminant function to find the separating hyperplane. However, some algorithms induce non-binary DTs (Hu, Deng, & Sui, 2009; Kim & Loh, 2001, 2003; Loh & Vanichsetakul, 1988).

An oblique DT called Ltree is introduced by Gama and Brazdil (1999) which combines the splitting mechanism of axis-parallel DTs (C4.5, Quinlan (1993)) with Linear Discriminant Functions (LDF) to induce oblique DTs. At each non-terminal node, LDF[3], a new set of features are constructed from the features available at that node. Axis-parallel splits are then searched along the original and the new features. If the split found involves one of the new features, then the decision boundary is oblique in the original feature space. One of the distinct features of this algorithm is that new features are propagated down through the tree to be considered for splitting lower nodes. The argument is as follows. Each LDF generates one new feature and is capable of discriminating only one class. If one of the new features is selected at a particular node, then it discriminates only one of the classes. The other new features which are capable of discriminating other classes could be useful in lower nodes, therefore, new features are propagated down the tree. This approach can be viewed as an attempt to better explore the feature space by considering more splitting directions.

The Linear Discriminant and Tabu Search (LDTS) algorithm (X. B. Li et al., 2003) builds an oblique DT using linear discriminant functions as splitting hyperplanes. At each non-terminal node, a set of new feature variables is constructed

[3]number of LDF = (number of classes at the node) - 1

based on linear discriminant functions that are computed using all, or a subset, of the feature variables. Then, the best discriminant function (the one which reduces the impurity function the most) is selected as the splitting hyperplane at the node. Since the number of subsets of feature variables grows exponentially with the number of features, Tabu search (Glover, 1986) is used to select the best combination. The distinct feature of LDTS is that it may use the full set of the features or subset of them to construct discriminant functions whereas other methods use only the full set of features.

Kolakowska and Malina (2005) use Fisher's Linear Discriminant Function (FLDF) to construct oblique DTs. Three different DT induction algorithms were proposed which use FLDF in various forms. The algorithms differ in the way they separate the classes at each non-terminal node. For example, one algorithm separates one class from the rest while another algorithm divides classes into two distinct groups. The third is a general case of the second one as two groups are no longer distinct and hence, one class may be in the both groups.

Quick Unbiased Efficient Statistical Tree (QUEST) (Loh & Shih, 1997) uses Linear Discriminant Analysis (LDA) to find the best split at each node and hence there is no requirement for searching for the best split. QUEST's axis-parallel tree begins by performing an ANOVA test at each non-terminal node to select the best feature. LDA is then applied to the selected feature to find the best splitting point. QUEST's oblique DT simply applies LDA on all the features to find the best splitting hyperplane. For multi-class problems, QUEST groups the classes into two super-classes using the k-means clustering algorithm. Furthermore, QUEST is able to find oblique splits which are a linear combination of qualitative and quantitative features. First, each qualitative feature is transformed into a quantitative feature and then LDA is applied to the full set of features to find the separating hyperplane. The method for the transformation is used in this research to handle both types of features in the

same oblique split.

Very recently, López-Chau, Cervantes, López-García, and Lamont (2013) proposed Fisher's DT (FDT) for two-class classification problems. At each non-terminal node, all the examples are projected onto a vector ω obtained by the Fisher's linear discriminant analysis and an axis-parallel search is carried out along ω. The DT induction is efficient because the search for the best split is limited to one dimension at each non-terminal node.

2.4 Tree induction methods based on heuristics

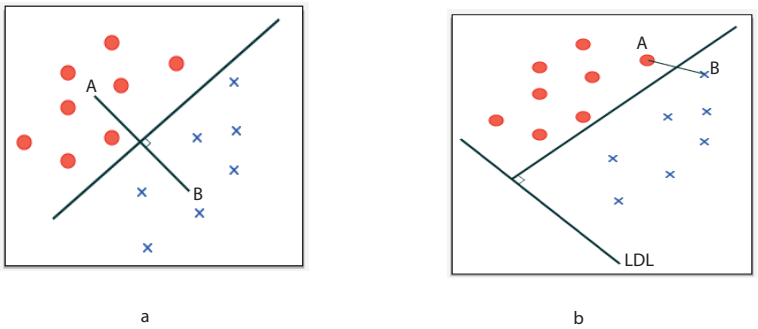
Henrichon and Fu (1969) introduced a tree type classifier for multivariate multi-class classification problems. For each dimension, the example set is divided into K disjoint regions, where K is a pre-specified value. Each region is given a class label such that it minimises the misclassification cost. Then the adjacent regions with similar classes are joined. The boundaries of the regions are then perturbed and joined to adjacent regions where the improvement of classification accuracy is obtained. Finally the empirical classification statistic (*Score*), a function of the number of misclassified examples, is computed to evaluate the goodness of the partitions obtained. From the partitions obtained along each dimension, the one which minimises the *Score* statistic is selected to split the feature space. The same procedure is then repeated in each sub-region of the partition until no further partitioning is possible. Each split in the algorithm is axis-parallel and the tree is non-binary. Apart from original feature variables, the authors suggest using new feature variables called transgenerated features for partitioning the feature space. One of the suggestions from the authors for a transgenerated feature is to construct a new feature of the form $\mathbf{x}^T\mathbf{d}_i$ where $\mathbf{d}_i$ is the eigenvector corresponding to the maximum eigenvalue associated with the covariance matrix of the i^{th} class and $\mathbf{x}$ is the feature vector. These splits are oblique in the

original feature space.

Another heuristic family of algorithms developed by Amasyah and Ersoy (2008), called Cline, were originally developed for 2-class problems. The Cline family has a number of different heuristic methods to determine separating hyperplanes. Two of the methods are highlighted here. Other methods can be found in Amasyah and Ersoy (2008).

[1] The splitting hyperplane is one which passes through the mid point of the line (AB) joining the mean of two classes and perpendicular to AB. This is given in Figure 2.1a.

[2] The line AB is one joining the two nearest points of the two different classes (see Figure 2.1b). The splitting hyperplane is one which goes through the mid point of AB and is perpendicular to the Linear Discriminant Line of the two classes.



a b

Figure 2.1: Two heuristics used in the Cline algorithm

According to the authors the best version, called ClineMix, tries several methods to find the best splitting hyperplane at each node. However, extending ClineMix to multi-class problems is time consuming. ClineMix uses a one-versus-one method where it constructs several classifiers each to distinguish one class from another class.

More specifically, if there are C classes then the Cline algorithm constructs $\binom{C}{2}$ classifiers. Because of the way that the heuristics work, the requirement for evaluating an impurity function vanishes.

Geometric DT (GDT) is another oblique DT that exploits the geometric structure of the data (Manwani & Sastry, 2012). For a two-class classification problem, the algorithm generates two clustering hyperplanes, one for each class. Loosely speaking, each clustering hyperplane tries to minimise the distance to examples in one class while maximising the distance to examples in the other class. The separating hyperplane is found by calculating the angular bisector of the two clustering hyperplanes. Since there are two angular bisectors, the one that minimizes an impurity measure is chosen as the splitting hyperplane. For a multi-class classification problem, the authors suggest forming two super-classes where one super-class contains the class which has the most examples and the remaining examples from the other classes are grouped into the other super-class. GDT does not require a search procedure to select splitting hyperplanes. At each non-terminal node it only requires two evaluations of an impurity function. An algorithm proposed in this research is used to improve the performance of GDT. A detailed description of GDT is therefore given in Chapter 5.

The CARTopt algorithm introduced by (Robertson et al., 2013), uses a two-class oblique DT to find a minimiser of a non-smooth function $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^n$. Initially, the examples in $\mathbb{R}^n$ are labelled (or classified) into two classes: "high" and "low" depending on their value of $f(\mathbf{x})$. One of the main tasks of the CARTopt algorithm is to identify a rectangular region which contains the most "low" points. The authors use axis-parallel partitions to identify the rectangular region. However, if the orientation of the "low" points is not aligned with the coordinate axes, the axis-parallel partitions will approximate the entire rectangular region by a series of small rectangular regions. This is illustrated in Figure 2.2 which is extracted from Robertson et al. (2013). To simplify the partition structure, Robertson et al. (2013) use a transformation, by

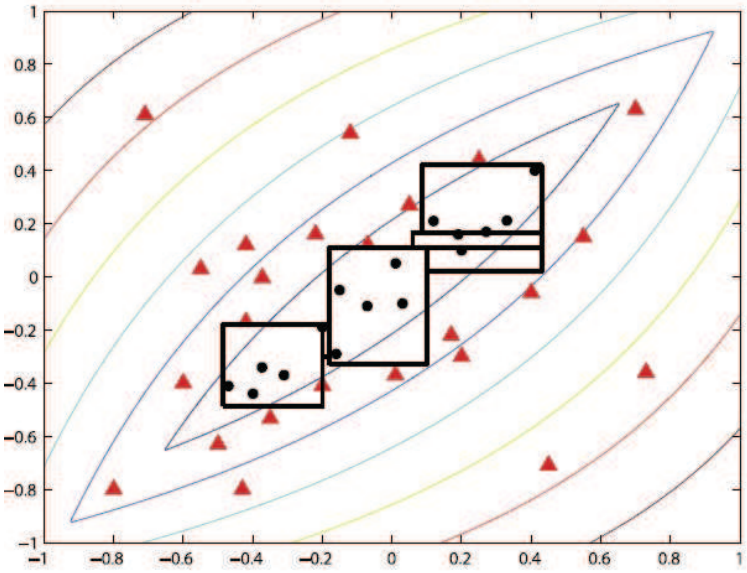


Figure 2.2: Orientation of the "low" points (depicted by black colour) in the original space. "High" points are depicted in red.

which the orientation of the "low" points becomes parallel to one of the coordinate axes in the transformed space. The axis-parallel splits can then be searched in the transformed space to find the rectangular partition structure which contains the "low" points. This is illustrated in Figure 2.3 (Robertson et al., 2013). The transformation is done using the Householder matrix and further details of the Householder matrix and the transformation are discussed in Chapter 3. In this study, the concept used in the CARTopt algorithm to create partitions is extended in a number of ways to develop a complete oblique DT for statistical data classification.

2.5 Other tree building methods

In addition to the above methods, numerous oblique DT induction algorithms have been proposed with various split selection methods. Some of the algorithms use

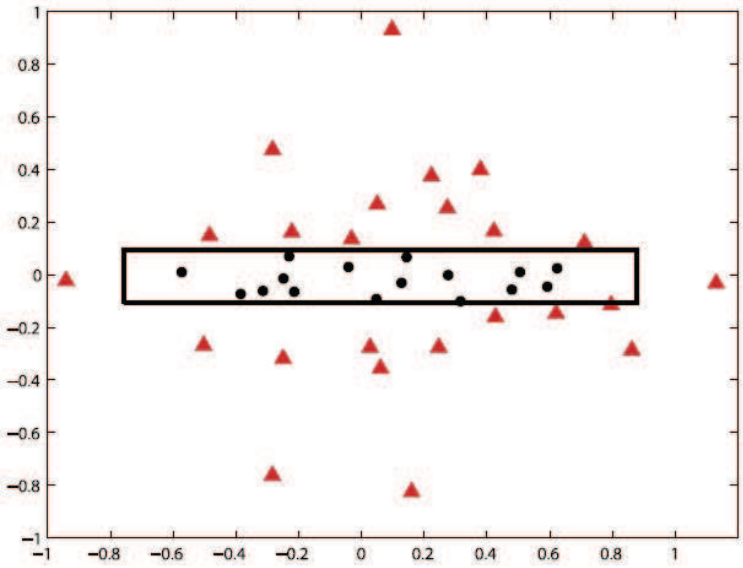


Figure 2.3: Orientation of the "low" points in the transformed space.

Support Vector Machines to partition the feature space (N. Li, Zhao, Chen, Meng, & Zhang, 2009). Neural Network algorithms are also used in DT inductions (Yildiz & Alpaydin, 2000) whereas fuzzy logic is used in Olaru and Wehenkel (2003). DTs based on evolutionary and genetic algorithms have been proposed in (Cantu-Paz & Kamath, 2003) and (Kretowski, 2004). Use of Linear Programming in constructing DTs was explored by (Bennett, 1992).

2.6 Discussion

DT induction algorithms have been explored since the late 1960's. However, the first major oblique DT was proposed by Breiman et al. (1984). Early oblique tree induction methods are mostly based on optimisation algorithms to find the best split. The DT induction methodology was also influenced by the development of artificial neural networks, evolutionary and genetic algorithms. Heuristic methods and standard

statistical techniques have been continuously explored by researchers especially over the last three decades. The results obtained show that there is no apparent difference between heuristic and non-heuristic methods with respect to the performance measures such as accuracy and tree size. Although standard statistical methods are computationally cheap, they rely on assumptions which may not be valid or justified. LDA assumes equal covariance matrices for example. On the other hand, heuristic methods are gaining popularity. They usually do not require statistical assumptions and are fast. Some of the algorithms do not require the evaluation of an impurity function. Moreover, Section 1.8.1 demonstrates that locally optimal solutions do not always lead to globally optimal solutions. Hence, spending more time searching for the best split at a node in general may not be beneficial (Iyengar, 1999) or possible. In fact, finding the best oblique split is a NP-complete problem (Heath et al., 1993). Therefore, building oblique DTs using heuristic methods is justifiable when trading off accuracy of the tree with time complexity. This fact leads the author to propose a new oblique DT induction methodology based on a heuristic method which is fully explained in the next chapter.

Chapter 3

HHCART: An oblique decision tree

3.1 Introduction

This chapter presents a detailed description of a novel proposed method, which is a comprehensive extension of the work of Robertson et al. (2013). In this study, the method used in the CARTopt algorithm to construct feature space partitions is extended in a number of ways to develop a complete oblique DT called HHCART (HouseHolder Classification and Regression Tree). First, CARTopt is designed to classify two classes whereas HHCART can handle multi-class classification problems. Second, CARTopt reflects the training examples only at the root node whereas HHCART performs reflections at each non-terminal node during tree construction. This is an important part of the proposed algorithm particularly for multi-class data classification. Finally, CARTopt deals only with quantitative features whereas HHCART is capable of finding oblique splits which can be linear combinations of both quantitative and qualitative features. This step enables HHCART to be applied in an any feature space and hence, broadens the applicability of the algorithm.

3.2 Householder reflection for a two-class problem

Chapter 2 introduced the concept used in the CARTopt algorithm to create a partition for a two-class problem. In this section, the same concept is explained from the oblique DT point of view. Consider the two dimensional, two-class classification problem shown in Figure 3.1. Here the direction of the separating hyperplane can be taken as the most stretched direction of either class. The most stretched direction of each class is given by the dominant eigenvector of its class covariance matrix S defined as:

Definition 3.2.1. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be p dimensional feature vectors where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$. The estimated covariance matrix is then given by:

$$\mathrm{S} = \frac{1}{(n-1)} \sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T, \tag{3.2.1}$$

where $\bar{\mathbf{x}} = (\mathbf{x_1} + \mathbf{x_2} + \cdots + \mathbf{x_n})/n$ is the mean vector.

Since there are two classes, two dominant eigenvectors can be found (called $\mathbf{d}^1$ and $\mathbf{d}^2$), one for each class. In the illustration, a hyperplane which is parallel to either $\mathbf{d}^1$ or $\mathbf{d}^2$ can be a candidate direction for a separating hyperplane of the classes. Hence, one of these eigenvectors ($\mathbf{d}^1$ say) is reflected such that it becomes parallel to one of the coordinate axes (for example, $\mathbf{e}_1$). Consequently, the orientation of the separating hyperplane also becomes parallel to $\mathbf{e}_1$ in the reflected space. The scatter of examples in the reflected space is shown in Figure 3.2. Now the separating hyperplane can be found by performing axis-parallel splits along the $\mathbf{e}_2$ direction in the reflected space.

3.3 Householder matrix

The reflection of a set of examples is the key idea in the HHCART algorithm. The examples are reflected using a Householder matrix, which is used by Robertson et

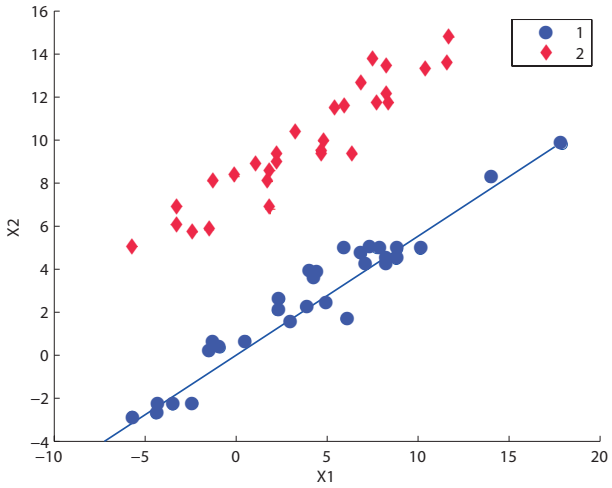


Figure 3.1: Scatter of examples in the original feature space.

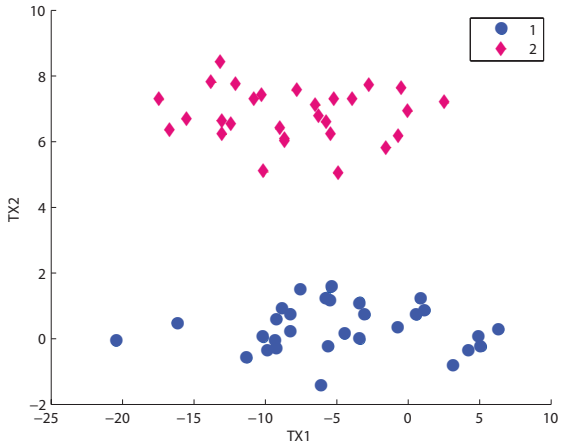


Figure 3.2: Scatter of examples in the transformed space after the Householder Reflection.

al. (2013) to induce space partitions. It can be defined for a p-dimensional space as follows. Let $\mathbf{d}^1$ be the normalized dominant eigenvector of class 1 examples and $\mathbf{e}_{1_{p\times 1}} = (1, 0, \ldots, 0)^T$ be a basis vector. Hence, both the vectors have the same norm. There exists an orthogonal symmetric matrix $\mathrm{H}_{p\times p}$ (where p is the number of features) such that

$$\begin{aligned} \mathbf{e}_1 &= \mathrm{H}\mathbf{d}^1 \text{ where} \\ \mathrm{H} &= \mathrm{I} - 2\mathbf{u}\mathbf{u}^T \text{ where } \mathbf{u} = \frac{\mathbf{e}_1 - \mathbf{d}^1}{\|\mathbf{e}_1 - \mathbf{d}^1\|_2}. \end{aligned} \tag{3.3.1}$$

3.3.1 Construction of the Householder matrix

In this section, the construction of the Householder matrix is explained. Let $\mathbf{d}^1$ be the vector to be reflected using the matrix H such that $\mathbf{e}_1 = \mathrm{H}\mathbf{d}^1$. Vectors $\mathbf{d}^1$ and $\mathbf{e}_1$ are shown in Figure 3.3. Vector $\mathbf{e}_1$ can be written as an addition of two vectors, $\mathbf{e}_1 = \mathbf{d}^1 + \tilde{\mathbf{u}}$, where $\tilde{\mathbf{u}} = \mathbf{e}_1 - \mathbf{d}^1$. According to Figure 3.3, $AB = \|\mathbf{d}^1\| \cos\theta$ and $AC = 2AB$ because $\mathbf{e}_1$ is the reflection of $\mathbf{d}^1$. Therefore,

$$\tilde{\mathbf{u}} = 2\left\|\mathbf{d}^1\right\| \frac{\tilde{\mathbf{u}}}{\|\tilde{\mathbf{u}}\|} \cos\theta.$$

Hence,

$$\mathbf{e}_1 = \mathbf{d}^1 + 2\left\|\mathbf{d}^1\right\| \frac{\tilde{\mathbf{u}}}{\|\tilde{\mathbf{u}}\|} \cos\theta.$$

Since, $\mathbf{d}^1 \cdot \tilde{\mathbf{u}} = \|\mathbf{d}^1\| \, \|\tilde{\mathbf{u}}\| \cos(\pi - \theta)$, we have:

$$\cos\theta = -\frac{\mathbf{d}^1 \cdot \tilde{\mathbf{u}}}{\|\mathbf{d}^1\| \, \|\tilde{\mathbf{u}}\|} \text{ and,}$$

$$\mathbf{e}_1 = \mathbf{d}^1 - 2\left\|\mathbf{d}^1\right\| \frac{\tilde{\mathbf{u}}}{\|\tilde{\mathbf{u}}\|} \frac{\mathbf{d}^1 \cdot \tilde{\mathbf{u}}}{\|\mathbf{d}^1\| \, \|\tilde{\mathbf{u}}\|}$$

$$\mathbf{e}_1 = \mathbf{d}^1 - 2\tilde{\mathbf{u}} \frac{\tilde{\mathbf{u}}^T \mathbf{d}^1}{\|\tilde{\mathbf{u}}\|^2}. \tag{3.3.2a}$$

Since $\tilde{\mathbf{u}} = \mathbf{e}_1 - \mathbf{d}^1$, equation (3.3.1) gives $\tilde{\mathbf{u}} = \|\tilde{\mathbf{u}}\| \, \mathbf{u}$. Therefore, $\tilde{\mathbf{u}}$ in equation (3.3.2a)

is substituted by $\|\tilde{\mathbf{u}}\|\,\mathbf{u}$ to give:

$$\mathbf{e}_1 = \left(\mathrm{I} - 2\mathbf{u}\mathbf{u}^T\right)\mathbf{d}^1.$$

That is, $\mathrm{H} = \left(\mathrm{I} - 2\mathbf{u}\mathbf{u}^T\right)$. □

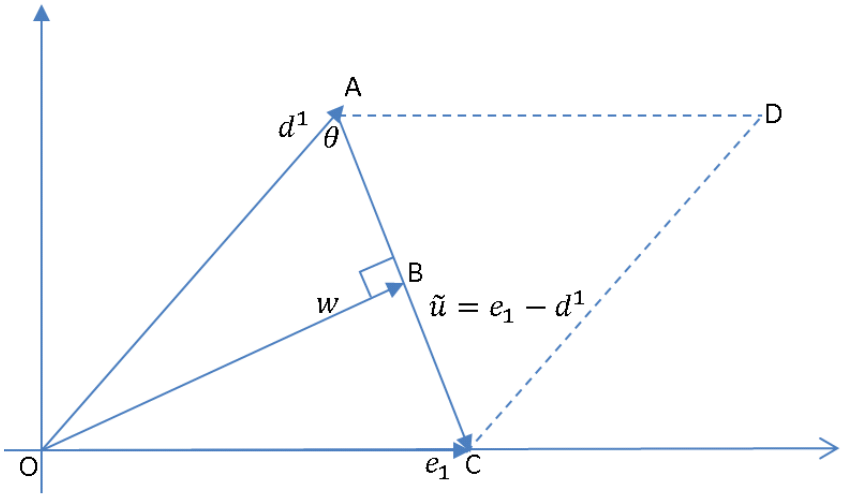


Figure 3.3: Geometry of Householder Reflection.

3.3.2 Some properties of the Householder matrix

Matrix H is symmetric as:

$$\mathrm{H}^T = (\mathrm{I} - 2\mathbf{u}\mathbf{u}^T)^T = \mathrm{I} - 2\mathbf{u}\mathbf{u}^T = \mathrm{H}. \tag{3.3.3}$$

Matrix H is orthogonal as:

$$\begin{aligned} \mathrm{H}^T\mathrm{H} &= (\mathrm{I} - 2\mathbf{u}\mathbf{u}^T)^T(\mathrm{I} - 2\mathbf{u}\mathbf{u}^T) \\ &= \mathrm{I} - 2\mathbf{u}\mathbf{u}^T - 2\mathbf{u}\mathbf{u}^T + 4(\mathbf{u}\mathbf{u}^T)^T\mathbf{u}\mathbf{u}^T \\ &= \mathrm{I}. \end{aligned} \tag{3.3.4}$$

Since, the Householder matrix is symmetric and orthogonal, a point in the transformed space can be mapped back to the original space at a minimal cost. Let $\mathbf{x}$ be a point in the original space. Define the transformed point $\hat{\mathbf{x}} = \mathrm{H}\mathbf{x}$. Multiplying both sides by H gives $\mathrm{H}\hat{\mathbf{x}} = \mathrm{HH}\mathbf{x}$. Since H is orthogonal and symmetric, $\mathrm{H}^T = \mathrm{H}^{-1}$. Therefore, HH=I where I is an identity matrix. Therefore, $\mathrm{H}\hat{\mathbf{x}} = \mathbf{x}$.

Let $\mathfrak{D}_{n\times p}$ be the training example set. The reflected example set $\hat{\mathfrak{D}}_{n\times p}$ is obtained using $\hat{\mathfrak{D}} = \mathfrak{D}\mathrm{H}$. The mechanism of the Householder reflection is that it makes a vector $\mathbf{d}^1$ parallel to $\mathbf{e}_1$ by a reflection through the plane perpendicular to vector $\mathbf{e}_1 - \mathbf{d}^1$. The resultant Householder matrix is given by:

$$\mathrm{H} = \begin{pmatrix} d_1^1 & d_2^1 & d_3^1 & \dots & d_p^1 \\ d_2^1 & 1 - \frac{(d_2^1)^2}{1-d_1^1} & \frac{-d_2^1 d_3^1}{1-d_1^1} & \dots & \frac{-d_2^1 d_p^1}{1-d_1^1} \\ d_3^1 & \frac{-d_3^1 d_2^1}{1-d_1^1} & 1 - \frac{(d_3^1)^2}{1-d_1^1} & \dots & \frac{-d_3^1 d_p^1}{1-d_1^1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_p^1 & \frac{-d_p^1 d_2^1}{1-d_1^1} & \frac{-d_p^1 d_3^1}{1-d_1^1} & \dots & 1 - \frac{(d_p^1)^2}{1-d_1^1} \end{pmatrix}$$

where d_i^1, $i = 1, \dots, p$ is i^{th} component of $\mathbf{d}^1$.

Each column of H represents the direction of a coordinate axis in the reflected space. axis-parallel splits are searched along these axes and the best split found is oblique in the original space. The resultant oblique split found for the problem illustrated in Figure 3.1 is shown in Figure 3.4. It is found, by a simulation study, not only parallelising the dominant eigenvectors to $\mathbf{e}_1$, but also parallelising non-dominant eigenvectors to $\mathbf{e}_1$ increases the performances of the tree. Consequently, it creates new reflected spaces to search, which may contain better splits. Hence, the axis-parallel search space is enhanced by using all possible eigenvectors for reflections. For a p-dimensional classification problem with C classes there are Cp eigenvectors to be considered for the Householder reflection. However, this increases the time complexity of tree induction, but gives an opportunity to produce more accurate and compact trees. In some instances, the orientation of an eigenvector may be parallel to

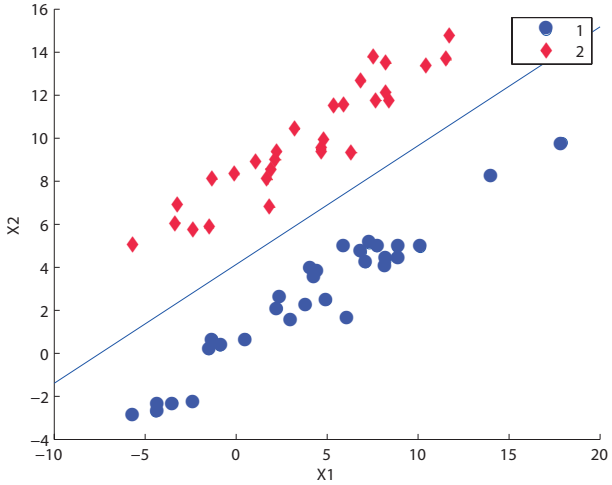


Figure 3.4: Split in the Original Space.

a feature axis in the original feature space. When this happens, the transformation is not required and hence, the best separating hyperplane is found by performing axis-parallel splits in the original space.

3.4 Householder reflection for a multi-class problem

In this section, an illustration is given to show how the proposed method partitions the two dimensional feature space for a multi-class classification problem. A two dimensional five-class example set, which can be linearly separated, was generated by the author. All possible eigenvectors are computed. For instance, at the root node 10 eigenvectors can be found (2 for each class). For each eigenvector, a Householder matrix is computed and the reflection is performed. axis-parallel splits are then carried out in each of the reflected spaces. Therefore, it is possible to find the best

split in a reflected space which is created from a non-dominant eigenvector. The scatter plot of the data is given in Figure 3.5. The following figures (Figure 3.6 -

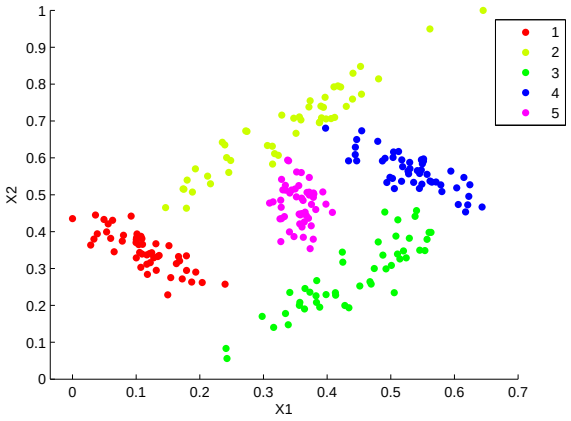


Figure 3.5: Scatter plot of examples belonging to five classes.

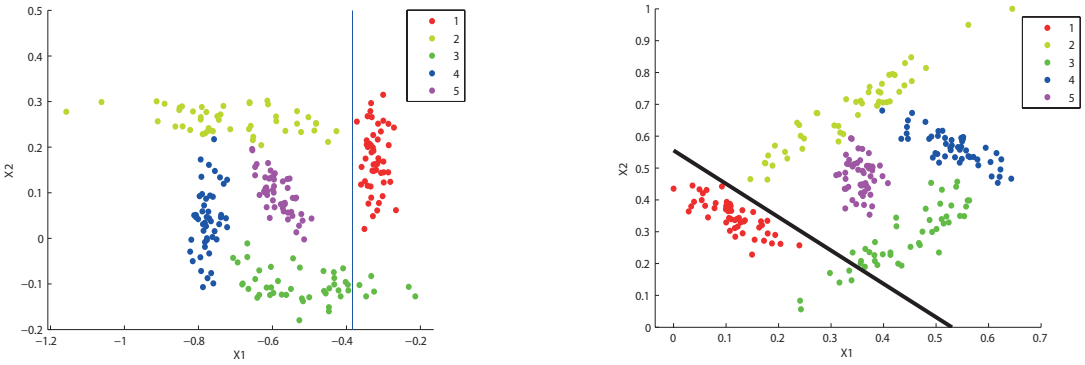
Figure 3.10) illustrate how multi-class classification is performed by the algorithm. The information provided below for each figure, contains:

[1] Node: Node in which the split is carried out.

[2] Space: The space which gives the best split.

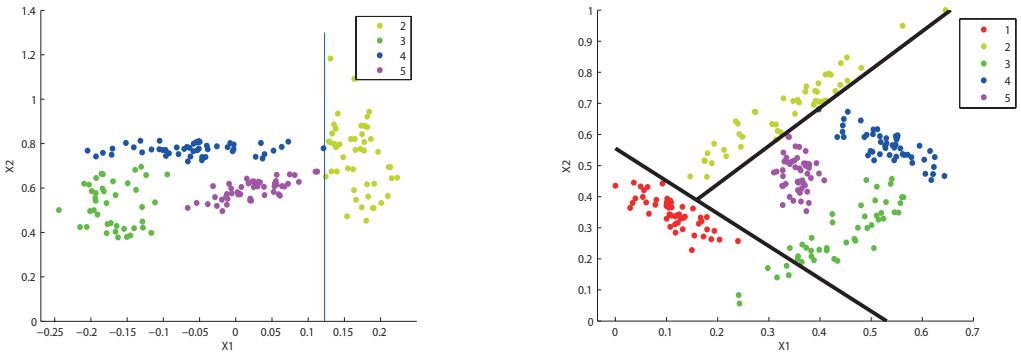
[3] Child nodes: Child nodes of the split. The status (terminal or non-terminal) of the node is given in the parentheses.

[4] Hyperplane: The equation of the hyperplane in the original space.



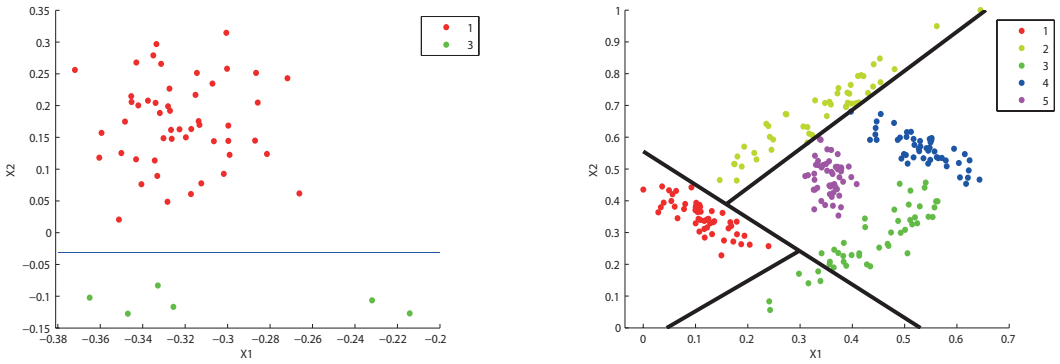
a Reflected Space b Original Space

Figure 3.6: Partition Structure at the root node
Space: Defined by the non-dominant eigenvector of group one examples.
Child Nodes: 2 (Non-Terminal) and 3 (Non-Terminal)
Hyperplane: $-0.7231X_1 - 0.6907X_2 + 0.3835 = 0$



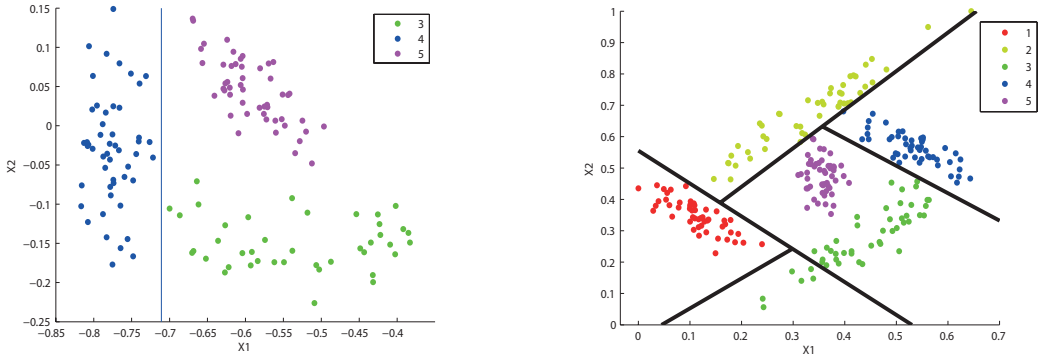
a Reflected Space b Original Space

Figure 3.7: Partition Structure at node 2
Space: Defined by the non-dominant eigenvector of group two examples.
Child Nodes: 4 (Non-Terminal) and 5 (Terminal)
Hyperplane: $-0.7757X_1 + 0.6371X_2 - 0.1227 = 0$



a Reflected Space b Original Space

Figure 3.8: Partition Structure at node 3
Space: Defined by the dominant eigenvector of group one examples.
Child Nodes: 6 (Terminal) and 7 (Terminal)
Hyperplane: $-0.6907X_1 + 0.7321X_2 + 0.0311 = 0$



a Reflected Space b Original Space

Figure 3.9: Partition Structure at node 4
Space: Defined by the non-dominant eigenvector of group four examples.
Child Nodes: 8 (Terminal) and 9 (Non-Terminal)
Hyperplane: $-0.6584X_1 - 0.7527X_2 + 0.7108 = 0$

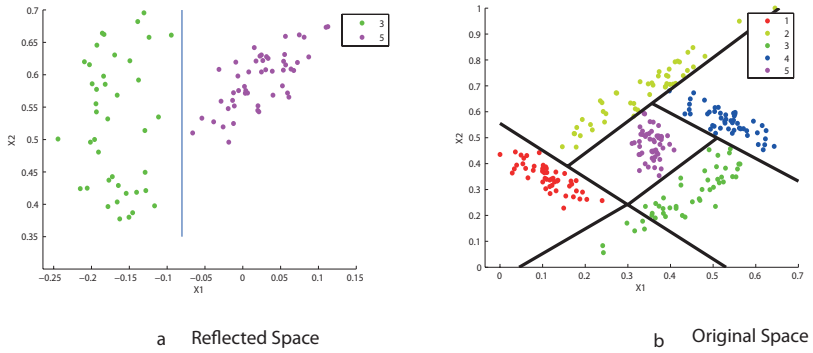


Figure 3.10: Partition Structure at node 9
Space: Defined by the non-dominant eigenvector of group three examples.
Child Nodes: 18 (Terminal) and 19 (Terminal)
Hyperplane: $-0.7757X_1 + 0.6311X_2 + 0.0804 = 0$

3.5 Proposed algorithm

Here the complete algorithm of HHCART is explained. Two versions of HHCART are proposed: (a) HHCART(A) is based on all possible eigenvectors of all classes, and (b) HHCART(D) is based on only the dominant eigenvector of each class. For any given non-terminal node t, let $\mathfrak{D}_t$ and C_t be the set of examples and classes available at that node respectively. At node t, HHCART(A) finds all eigenvectors of the estimated covariance matrix for each class whereas HHCART(D) finds only the dominant eigenvector of each class. A Householder matrix is constructed for each eigenvector. Then $\mathfrak{D}_t$ is reflected using each Householder matrix and axis-parallel splits are performed along each coordinate axis in the reflected space. The best axis-parallel split is chosen as the separating hyperplane at node t. However, if an eigenvector is already parallel to any of the feature axes, no reflection is done and hence, axis-parallel splits are searched in the original space. The hyperplane found by the search divides node t into two child nodes. The algorithm is recursively run on all child nodes until each child node satisfies either:

[1] The misclassification rate at the child node is not greater than a user specified threshold (MisRate); or

[2] The number of examples in the node is less than or equal to a user specified threshold (MinParent).

In both algorithms, the reflection is done if an eigenvector is not parallel to any feature axis. However, there may be a situation where the eigenvector is almost parallel to a feature axis and hence, the reflection may not be beneficial. Therefore, another parameter τ is introduced to the algorithms which can be used as a threshold to determine the parallelism between eigenvector and a feature axis. That is if $\|\mathbf{e} - \mathbf{d}\| \leq \tau$, where $\mathbf{e}$ and $\mathbf{d}$ are a basis vector and an eigenvector respectively, no reflection is done and axis-parallel splits are searched in the original space. In the experiments τ was set to 0.05 arbitrarily. However, the user can choose any small positive value or use a separate CV procedure to estimate the optimal value for τ.

An overview of HHCART(A) algorithm at node t is given in Algorithm 3. The time complexity at a node for HHCART(A) in the worst case is $O(Cp^2(p + n \log n))$ (see Section 3.5.1 for the derivation). However, if HHCART(D) is used then the time complexity reduces to $O(Cnp(p + \log n))$.

Data: Input: Examples at node t, called $\mathfrak{D}_t$, *Minparent, MisRate*, and $\tau >= 0$.
initialization;
Define N_t = Number of examples in $\mathfrak{D}_t$;
Define mp_t = misclassification rate at node t;
Define C_t = number of classes at node t;
Define p = number of features;
$\Delta(I_{max}) = 0$;
$h_t = empty$;
if *($N_t >$ Minparent) and (MisRate $< mp_t$)* **then**
  for *i=1:C_t* **do**
    Extract the examples that belong to the i^{th} class in $\mathfrak{D}_t$, called D_i;
    Compute the normalized eigenvectors and eigenvalues of the estimated covariance matrix for D_i;

$$((d^{1i}, \lambda^{1i}), \ldots (d^{pi}, \lambda^{pi}))$$

    for *j=1:p* **do**
      if $\lambda^{ji} \neq 0$ **then**
        if $\|e_1 - d^{ji}\| \leq \tau$ *or* $\|e_2 - d^{ji}\| \leq \tau$ *or* $\ldots$ *or* $\|e_p - d^{ji}\| \leq \tau$ **then**
          $H_t^{ji} = I$, the Identity matrix;
        else
          Construct the Householder matrix H_t^{ji} using d^{ji} ;
        end
        Reflect $\mathfrak{D}_t : \hat{\mathfrak{D}}_t = \mathfrak{D}_t * H_t^{ji}$;
        Find the best axis-parallel hyperplane split, called h_t^{ji};
        if *impurity reduction of* $h_t^{ji} > \Delta(I_{max})$ **then**
          Replace h_t with h_t^{ji}, the best hyperplane found so far;
          Replace $\Delta(I_{max})$ with the impurity reduction of h_t^{ji}
        end
      end
    end
  end
end

Algorithm 3: Overview of HHCART(A) algorithm at a single node

3.5.1 Time complexity of HHCART

The maximal time complexity at a node of HHCART(A) and HHCART(D) is derived. Assume there are n examples with p quantitative features and C classes at the node.

[1] **HHCART(A)** and **HHCART(D)** - Complexity for constructing estimated covariance matrix for one class of examples is $O(np^2)$. For C classes the complexity is $O(Cnp^2)$.

[2] **HHCART(A)** - Complexity of the complete eigenanalysis for one class of examples is $O(p^3)$. For C classes the complexity is $O(Cp^3)$.
HHCART(D) - Complexity for finding the dominant eigenvector for one class of examples is $O(p^2)$. For C classes the complexity is $O(Cp^2)$.

[3] **HHCART(A)** - Complexity for the reflection of n examples using one Householder matrix is $O(np)$. Since, there are Cp Householder matrices the Complexity is $O(Cnp^2)$.
HHCART(D) - Complexity for the reflection of n examples using one Householder matrix is $O(np)$. For C Householder matrices the complexity is $O(Cnp)$.

[4] **HHCART(A)** - Complexity of finding the best axis-parallel splits for one reflected space is $O(p(n + n\log n)$. That is, along one dimension, sorting the examples takes $O(n\log n)$ time and impurity function evaluation takes maximum of $O(n)$ time. Hence, the total time along p dimension (one reflected space) is $O(p(n+n\log n))$. Since, there are Cp reflected spaces the Complexity is $O(cp^2(n + n\log n)) = O(cp^2n(1 + \log n)) = O(cp^2n\log n)$.
HHCART(D) - Complexity of finding the best axis-parallel splits for one reflected space is $O(p(n + n\log n)$. For C classes the complexity is $O(Cp(n + n\log n) = O(pn(1 + \log n) = O(Cpn\log n)$.

[5] **HHCART(A)** - The maximal time complexity at a node is therefore: $O(Cnp^2)+O(Cp^3) + O(Cnp^2) + O(Cp^2 n \log n) = O(Cp^2(p + n \log n))$.
HHCART(D) - The maximal time complexity at a node is therefore: $O(Cnp^2)+ O(Cp^2) + O(Cnp) + O(Cpn \log n) = O(Cnp(p + \log n))$.

The number of impurity function evaluations at a non-terminal node is Cnp^2 for HHCART(A) whereas for HHCART(D) it is Cnp. These figures are significantly smaller than the number of impurity function evaluations required by OC1, n^2p, in the worst case scenario if Max-Minority or Sum-Minority impurity measures are used.

3.5.2 Space complexity of HHCART

The maximal space complexity of HHCART is derived. Both algorithms, HHCART(A) and HHCART(D), have the same space complexity. Here too, n, p and C have the same meaning as in Subsection 3.5.1.

[1] The space required for storing the entire example set is $O(np)$.

[2] The space required for the examples in one transformed space $O(np)$. There are Cp transformed spaces. However, axis-parallel splits are performed sequentially. Therefore, once the search of one transformed space is completed examples in that space can be deleted. Hence, the space complexity remains at $O(np)$.

[3] The final decision tree holds some information at each node. The largest tree has n nodes and each node holds: (a) the class label, (b) the class distribution vector (C-dimensional), and (c) status of the node: whether terminal or non-terminal node. All this information requires the maximal space complexity of $O(nC)$. Each non-terminal node holds a p-dimensional coefficient vector of the separating hyperplane and there is a maximum of $n - 2$ non-terminal nodes[1].

[1] for binary trees with minimum node size is 2.

Hence, the space requirement for holding the hyperplanes is $p(n-2)$.

[4] Therefore, the total space complexity of HHCART is: $O(np) + O(np) + O(nC) + p(n-2) = O(np) + O(nC) = O(np)$.

3.6 Small samples

As the tree grows, the number of examples at each node usually becomes small. This raises two questions to be answered: (a) Is it worthwhile searching for an oblique split rather than an axis-parallel split? (b) Covariance matrices tend to be singular with small samples.

The first problem is common for any oblique decision tree. In the OC1 algorithm, Murthy et al. (1994) suggest using oblique splits if the number of examples at that node is greater than twice the number of feature variables. The second problem is specific to those algorithms which use matrix operations, for example eigen analysis or the inverse of matrix, to find oblique splits. Manwani and Sastry (2012) introduced a different computation procedure for small samples and it is given in Subsection 5.1.2 in Chapter 5.

The effect of a small sample for HHCART is as follows:

[1] Lack of information in eigenvectors with zero eigenvalues from a singular covariance matrix.

[2] Eigenvectors are not informative for those classes having only one example or several examples with the same feature vector.

The first problem can be solved without modifying the method because the reflection is done using available eigenvectors. For the second problem, those classes are disregarded from eigenanalysis. However, if all the classes suffer from this problem axis-parallel splits are performed.

3.7 Qualitative feature variables

Many practical data classification problems often contain a mixture of quantitative and qualitative feature variables. Since the class discriminatory information may be contained in both types of feature variables, an effective classifier should be able to handle both types of features in the classification process. For a qualitative feature variable X, the form of the split is given by $X \in A$ where A is a non-empty subset of values taken by X. If a qualitative feature has M non-empty levels, then $2^{M-1} - 1$ splits are possible. axis-parallel algorithms which consider qualitative splits can be found in Quinlan (1986).

Incorporating qualitative features in oblique splits has not been thoroughly explored. The QUEST algorithm (Loh & Shih, 1997) is capable of finding oblique splits with both qualitative and quantitative features. QUEST transforms each unordered qualitative feature variable into a new ordered quantitative feature, called CRIMCOORD. Each level of an unordered qualitative feature is mapped to an ordered value called a CRIMCOORD value. The CRIMCOORD algorithm is briefly explained below and the exact algorithm can be found in (Loh & Shih, 1997).

Let X be a qualitative feature taking values in the set $\{1, 2, \ldots L\}$ and the class variable $Y \in \mathbb{C} = \{1, 2, 3, ..., \mathrm{C}\}$. Each level of X is first transformed into an L-dimensional dummy vector $\mathbf{v} = \{v_1, v_2, \ldots, v_L\}$ as follows:

$$\text{For} \quad i = 1, \ldots, L, \qquad v_i = \begin{cases} 1 & X = i \\ 0 & \text{otherwise.} \end{cases} \tag{3.7.1}$$

Let $\mathbf{v}_i^j$ be the i^{th} observed value of $\mathbf{v}$ in the j^{th} response class and define the mean vector of class j and the grand mean vector as $\bar{\mathbf{v}}^j = N_j^{-1} \sum_{i=1}^{N_j} \mathbf{v}_i^j$ and $\bar{\mathbf{v}} = N^{-1} \sum_{j=1}^{C} \sum_{i=1}^{N_j} \mathbf{v}_i^j$ respectively. Then the between sum of squares matrix B and the

within sum of squares matrix W can be computed as:

$$\mathrm{B} = \sum_{j=1}^{C} N_j(\bar{\mathbf{v}}^j - \bar{\mathbf{v}})(\bar{\mathbf{v}}^j - \bar{\mathbf{v}})^T, \tag{3.7.2a}$$

$$\mathrm{W} = \sum_{j=1}^{C}\sum_{i=1}^{N_j}(\mathbf{v}_i^j - \bar{\mathbf{v}}^j)(\mathbf{v}_i^j - \bar{\mathbf{v}}^j)^T, \tag{3.7.2b}$$

respectvely and the total sum of square matrix T can be computed as T= B + W, specifically:

$$\mathrm{T} = \sum_{j=1}^{C}\sum_{i=1}^{N_j}(\mathbf{v}_i^j - \bar{\mathbf{v}})(\mathbf{v}_i^j - \bar{\mathbf{v}})^T. \tag{3.7.2c}$$

The idea is to find a vector ($\mathbf{a}$ say) to project $\mathbf{v}$ which maximises the between sum of squares of projected scores ($\mathbf{a^T}\mathrm{B}\mathbf{a}$) while minimising the within sum of squares of projected scores ($\mathbf{a^T}\mathrm{W}\mathbf{a}$). More precisely, the aim is to find:

$$\mathbf{a}^* = \arg\max_{\mathbf{a}} \frac{\mathbf{a}^T\mathrm{B}\mathbf{a}}{\mathbf{a}^T\mathrm{W}\mathbf{a}}. \tag{3.7.3}$$

The solution to the optimisation problem given in equation (3.7.3) is the eigenvector corresponding to the maximum eigenvalue of $\mathrm{W}^{-1}\mathrm{B}$ when W is a full rank matrix. The solution to equation (3.7.3) is also given by $\mathrm{T}^{-1}\mathrm{B}$ when W is a full rank matrix (Loh & Shih, 1997). However, in this setting T and W are not in full rank. Therfore, a special computational method is needed to compute $\mathbf{a}^*$. The precise algorithm to compute $\mathbf{a}^*$ is given in Loh and Shih (1997). Let the vector $\mathbf{a}^*$ be the solution to equation (3.7.3) and hence, the largest discriminate coordinate, CRIMCOORD. Finally, each level of the qualitative feature is mapped to a real value by $\mathbf{a}^{*T}\mathbf{v_i}$. Figure 3.11 depicts the mechanism of the transformation. Assume a qualitative feature has two levels and hence, two dummy vectors are constructed: Level 1 - $(1,0)^T$, Level 2 - $(0,1)^T$. Also, assume that there are two classes so that two mean vectors can be computed and denoted by $\bar{\mathbf{v}}_1$ and $\bar{\mathbf{v}}_2$. The grand mean vector is denoted by $\bar{\mathbf{v}}$. The distances showed in the figure are defined as $D_1 = \mathbf{a}^{*T}\mathrm{B}\mathbf{a}^*$

and $d_{11} + d_{12} + d_{21} + d_{22} = \mathbf{a}^{*T}\mathrm{W}\mathbf{a}^{*}$. The vector $\mathbf{a}^{*}$ tries to find the direction which maximises the between sum of squares of projected dummy vectors while minimising the within sum of squares of projected dummy vectors. The projected values of $\mathbf{v}_1$ and $\mathbf{v}_2$ along the $\mathbf{a}^{*}$ vector are given by $\mathbf{v}_1^{'}$ and $\mathbf{v}_2^{'}$ respectively. Therefore Level 1 and Level 2 dummy vectors are replaced by $\mathbf{v}_1^{'}$ and $\mathbf{v}_2^{'}$ respectively.

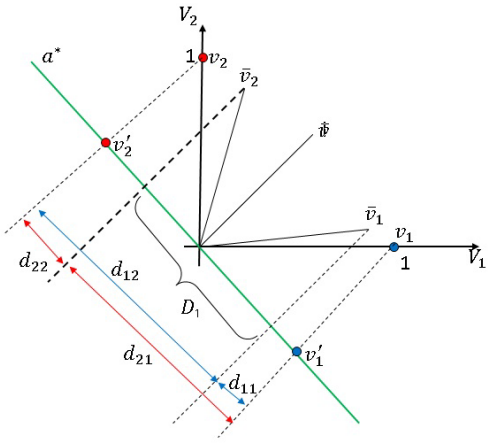


Figure 3.11: Mechanism of the transformation of the qualitative feature to qualitative feature.

The size of the example set at lower nodes decreases as the tree grows further. For smaller example sets, a qualitative feature may contain only one level. In these situations, the qualitative feature does not contain class discriminative information and hence, in the HHCART algorithm, the level of the qualitative feature is mapped to the value zero.

A significant property of the method is that it assigns zero for a level of a qualitative feature even if it is not in the learning sample. To realise this, the set of all possible values of the qualitative feature needs to be known beforehand. However, this enables the handling of qualitative levels that will appear in future examples but which are absent from the learning examples.

The same CRIMCOORD algorithm (proposed in Loh and Shih (1997)) is implemented in the HHCART algorithm to induce oblique splits which contain both qualitative and quantitative features. At each node, a new quantitative feature is constructed for each qualitative feature by mapping the levels to its CRIMCOORD. These new quantitative features are then amalgamated with the existing quantitative features in the example set. The HHCART algorithm can then be applied to find the best oblique split. At each node the CRIMCOORD value corresponding to each level of each qualitative feature is stored. When predicting, the level of each qualitative feature of an unclassified observation is replaced by the corresponding CRIMCOORD attached to each node along its path.

3.8 Importance of HHCART

HHCART possesses some important features which are useful in effective data classifications. These are summarised below:

[1] As discussed in Chapter 2, optimization algorithms based DTs take considerable time to find the best split at a node. For example, SADT and OC1 algorithms iterate many times until a locally optimal point of $\Delta(I)$ is found. However, HHCART uses axis-parallel splits and hence, an exhaustive search for the global minimum can be performed. Therefore, HHCART is a good alternative to those algorithms which uses optimization techniques to find the best split at a non-terminal node.

[2] The other advantage of HHCART is the ease in implementation of the algorithm. In the HHCART algorithm, only the axis-parallel splits are to be implemented. Optimisation algorithm based DTs use optimisation techniques, such as hill-climbing, simulated annealing. The implementation of these algorithms

is relatively harder compared with HHCART.

[3] With the ever increasing size of example sets serial computers face challenges such as: (a) memory limitations, and (b) inability to induce DTs in a reasonable time. Therefore, DT induction in a parallel computing architecture has been explored. Three induction strategies are proposed in parallel computing and it is shown that axis-parallel splits can be implemented using all of the strategies. Therefore, HHCART can also be parallelised using these three strategies because, axis-parallel splits are searched in the transformed spaces. However, Optimization algorithms based DT induction methods and even some DTs based on heuristic arguments are difficult to parallelise due to the inherent nature of the tree building algorithms. Hence, HHCART is a flexible tree building method which can be implemented under any computing architecture. Parallel computing architecture and the parallel version of HHCART is discussed in Chapter 4.

[4] As mentioned in Section 3.7, HHCART can handle both qualitative and quantitative features in the same oblique split which makes it useful in a diverse range of applications.

3.9 Experiments on real life example sets

Three sets of experiments are performed. In the first experiment, the performances of HHCART on real data sets, given in Appendix A, are compared with benchmark DTs. Five-fold cross validations (CVs) are used to estimate the classification accuracy. For each fold, 10% of the training set was used exclusively for pruning. Then 10, 5-fold CVs are used to estimate the accuracy and the size of the tree (number of terminal nodes). Therefore, to estimate accuracy and tree size the average over ten

runs was used. The 5-fold CV samples are obtained using Simple Random Sampling (SRS). Therefore, the original class distributions in the example set may not be properly represented in the test and training sets. Thus, in the second experiment, the classification accuracy of HHCART based on Simple Random Sampling CV (SRS CV) is compared with Stratified Random Sampling CV (STRS CV). Finally, in the last experiment, HHCART is tested on example sets in which the feature space is comprised of both qualitative and quantitative features. For all experiments, data sets are downloaded from UCI repository (Lichman (2013), *http://archive.ics.uci.edu/ml*) and are given in Table A.1 in Appendix A.

3.9.1 Comparison of performances of HHCART with other DTs

In this section, the HHCART algorithm is compared with OC1, OC1-LC (OC1's version of Breiman's linear combination methods) and OC1-AP (OC1 version of axis-parallel splits). All of these methods are available in the OC1 system which is freely available at *http://salzberg-lab.org/software*. However, the backward feature elimination process of Breiman's CART-LC method is not included in OC1-LC and hence, is somewhat different from the original method. For the HHCART algorithm, MinParent, MisRate and τ was set to 2, 0 and 0.05 respectively. For OC1, OC1-LC and OC1-AP MinParent was set to 2. All algorithms used the Twoing rule as the measure of impurity (Breiman et al., 1984) and cost complexity pruning (Breiman et al., 1984) with zero standard error. For OC1, the number of restarts and jumps were set to 20 and 5 (default values) respectively.

The Shuttle example set comes with its own training set containing 43500 examples and a test set with 14500 examples. Therefore, instead of performing a cross validation experiment, 10 trees were induced, each using 90% of training examples for induction

and the remaining 10% for pruning. The accuracy of all the trees was estimated using the Shuttle data test set. Since, approximately 80% of the examples belong to class 1, the aim is to achieve an accuracy between $99-99.9\%$ (Lichman, 2013). All experiment results are reported in Table 3.1 along with respective standard deviations.

Table 3.1: Results of HHCART and other DT methods.

example set	DT	Avg. Acc.	Avg. Size	example set	DT	Avg. Acc.	Avg. Size
BS	HHCART(A)	**93.7** ± 1.3	7.9 ± 1.7	PIND	HHCART(A)	72.2 ± 2.0	9.1 ± 5.1
	HHCART(D)	88.3 ± 1.7	12.2 ± 3.5		HHCART(D)	72.9 ± 1.3	10.8 ± 4.4
	OC1	91.9 ± 0.9	8.7 ± 3.4		OC1	73.4 ± 1.0	9.2 ± 5.4
	OC1-AP	78.2 ± 1.3	37.5 ± 16.8		OC1-AP	**73.6** ± 1.4	15.9 ± 8.7
	OC1-LC	84.3 ± 1.5	12.6 ± 6.5		OC1-LC	72.8 ± 1.8	11.4 ± 9.6
BH	HHCART(A)	**83.3** ± 0.9	6.5 ± 2.1	SHUT	HHCART(A)	99.94 ± 0.02	25.4 ± 5.9
	HHCART(D)	83.0 ± 0.7	9.9 ± 2.6		HHCART(D)	99.94 ± 0.05	26.1 ± 4.9
	OC1	82.2 ± 1.2	9.3 ± 3.4		OC1	99.95 ± 0.03	32.6 ± 7.71
	OC1-AP	82.0 ± 0.7	13.0 ± 5.3		OC1-AP	**99.97** ± 0.02	26.5 ± 5.6
	OC1-LC	81.5 ± 1.3	10.6 ± 6.0		OC1-LC	88.4 ± 7.07	44.7 ± 42.4
BC	HHCART(A)	**97.0** ± 0.3	2.4 ± 0.6	WINE	HHCART(A)	**91.3** ± 1.6	3.4 ± 0.3
	HHCART(D)	**97.0** ± 0.3	2.6 ± 1.1		HHCART(D)	88.7 ± 3.1	4.5 ± 0.6
	OC1	95.4 ± 0.5	3.3 ± 1.4		OC1	89.2 ± 2.1	3.5 ± 0.3
	OC1-AP	94.0 ± 0.8	8.3 ± 3.3		OC1-AP	89.2 ± 4.6	4.6 ± 0.6
	OC1-LC	95.5 ± 0.6	3.4 ± 1.6		OC1-LC	89.4 ± 2.7	3.8 ± 0.6
BUPA	HHCART(A)	64.1 ± 2.6	6.5 ± 1.5	LET	HHCART(A)	82.1 ± 0.3	759.2 ± 88.1
	HHCART(D)	62.4 ± 2.5	8.6 ± 3.1		HHCART(D)	83.1 ± 0.3	1135.9 ± 122
	OC1	**66.9** ± 2.2	8.9 ± 6.1		OC1	83.6 ± 0.4	1197.2 ± 88.9
	OC1-AP	64.7 ± 2.5	13.2 ± 10.5		OC1-AP	**86.3** ± 0.3	1611.7 ± 60.0
	OC1-LC	64.4 ± 2.4	8.9 ± 3.6		OC1-LC	84.5 ± 0.2	1332.6 ± 146.3
GLS	HHCART(A)	60.3 ± 3.0	8.5 ± 3.0	SUR	HHCART(A)	**73.5** ± 1.5	5.3 ± 2.7
	HHCART(D)	61.9 ± 3.0	10.1 ± 2.3		HHCART(D)	72.8 ± 1.0	5.0 ± 2.4
	OC1	61.1 ± 3.5	10.8 ± 4.3		OC1	71.0 ± 2.1	6.4 ± 3.5
	OC1-AP	64.6 ± 3.9	14.6 ± 8.7		OC1-AP	71.9 ± 1.5	10.7 ± 6.5
	OC1-LC	**67.4** ± 2.0	12.0 ± 3.6		OC1-LC	70.2 ± 2.4	8.1 ± 4.4
HRT	HHCART(A)	74.1 ± 2.9	4.5 ± 1.7	CLI	HHCART(A)	91.7 ± 1.0	2.4 ± 0.9
	HHCART(D)	75.8 ± 2.8	7.8 ± 2.6		HHCART(D)	91.8 ± 0.7	3.6 ± 1.1
	OC1	**77.1** ± 2.5	3.6 ± 1.0		OC1	91.5 ± 0.9	3.1 ± 0.9
	OC1-AP	76.3 ± 2.3	6.7 ± 2.4		OC1-AP	91.5 ± 0.9	4.0 ± 1.6
	OC1-LC	76.3 ± 2.5	4.0 ± 1.1		OC1-LC	**92.9** ± 0.7	4.1 ± 1.5
BNK	HHCART(A)	**99.4** ± 0.2	3.0 ± 0.3	SEED	HHCART(A)	90.4 ± 1.4	3.9 ± 0.8
	HHCART(D)	99.1 ± 0.3	3.6 ± 0.5		HHCART(D)	89.7 ± 2.8	3.9 ± 0.8
	OC1	98.6 ± 0.4	6.3 ± 1.2		OC1	**92.9** ± 1.8	3.6 ± 0.6
	OC1-AP	97.4 ± 1.0	14.7 ± 2.1		OC1-AP	88.8 ± 1.1	3.8 ± 0.9
	OC1-LC	97.9 ± 2.1	6.6 ± 1.5		OC1-LC	88.4 ± 1.1	3.8 ± 0.7

Table 3.1 shows the average accuracies and the average tree sizes of 10, 5-fold CVs along with the respective standard deviation. The oblique splits reduced the average tree size for most of the example sets while increasing the accuracy. First, the HHCART(A) algorithm is compared with the other DTs except HHCART(D). The average accuracy of HHCART(A) is significantly (more than 2 standard deviations) higher than all the other methods tested for the BC example set except for OC1-LC. For all other example sets, except for LET, the average accuracy of HHCART(A) is not significantly different from that of OC1.

The average tree sizes of HHCART(A) are consistently smaller than the average tree sizes of the other methods except for the HRT, SUR and SEED example sets. Therefore, the performance of HHCART(A) with respect to accuracy and tree size is better than the other methods for the BS, BH, BC, WINE and BNK datasets.

Nine of the 14 example sets have at least eight features. For six of these relatively high dimensional data sets, the performance of HHCART(A) is comparable with OC1 and OC1-LC. Hence, we can conclude that the proposed method works well in relatively high dimensional feature spaces provided that $p < n$.

For all the datasets except BS and WINE, HHCART(D) performs as well as HHCART(A) in terms of average accuracy. Also, the tree sizes of HHCART(D) are comparable with those produced by HHCART(A) except for the BS, BH, BUPA, HRT, WINE and LET example sets. The performance of HHCART(D) is similar to OC1 with respect to both the accuracy and tree size for all the datasets except the BS, HRT and BUPA datasets.

The time complexity of HHCART(A) is higher than that of HHCART(D) by a factor of $O(p)$. Results show that HHCART(D) produces DTs with similar accuracies and sizes as HHCART(A) and OC1 for most of the datasets. Hence, HHCART(D) would be a more efficient method to use for higher dimensional problems.

3.9.2 Effect of different sampling schemes

The HHCART algorithm is run on 10×5-fold CV samples which are created using SRS and Stratified Random Sampling (STRS) schemes. In the STRS CV setting, the training, test and a 10% pruning set is created. Estimated classification accuracies due to both sampling schemes are given in Table 3.2. The OC1 classifiers cannot be used in this situation as they are designed to work only on CV based on SRS. The SHUT example set comes with its own training set and it is found that the class proportions of test set and training set are almost the same.

Moreover, using the same 10×5-fold CV samples created above, the effect of the two different schemes on the class-wise accuracy is investigated and the results are given in Table 3.3.

Table 3.2: Classification accuracies for SRS and STRS sampling schemes.

	SRS CV		**STRS CV**	
example set	**Accuracy**	**Tree Size**	**Accuracy**	**Tree Size**
BS	92.8 ± 1.3	7.4 ± 1.3	93.2 ± 1.4	8.4 ± 2.2
BH	83.4 ± 1.2	7 ± 2.9	83.1 ± 1.2	6.7 ± 3.1
BC	96.8 ± 0.9	2.3 ± 0.4	96.9 ± .3	2.7 ± 0.7
BUPA	63.8 ± 2.6	7.6 ± 1.5	66.1 ± 2.6	8.0 ± 3.6
GLS	60.9 ± 3.8	8.6 ± 3.2	63.3 ± 2.8	9.8 ± 2.3
HRT	74.3 ± 3.5	5.3 ± 2.1	75.2 ± 1.7	5.3 ± 1.2
PIND	72.6 ± 1.9	11.7 ± 6.6	73.5 ± 1.0	9.1 ± 4.7
SHUT	99.93 ± 0.02	26.2 ± 1.7	99.93 ± .02	28.3 ± 1.3
WINE	91.4 ± 1.8	3.4 ± 0.3	90.6 ± 1.5	3.3 ± 0.3
LET	82.9 ± 0.1	771.8 ± 92.1	82.8 ± 0.3	763.6 ± 90.2
SUR	73.1 ± 1.2	3.3 ± 1.9	72.6 ± 1.9	5.9 ± 2.7
CLI	91.7 ± 1.0	2.9 ± 1.7	91.8 ± 0.8	2.9 ± 1.1
BNK	99.1 ± 0.2	2.96 ± 0.3	99.4 ± 0.2	3.0 ± 0.3
SEED	90.4 ± 1.3	3.96 ± 0.8	91.6 ± 1.2	4.0 ± 0.6

It is clear from the results given in Table 3.2 that the average accuracies are more or less the same in both sampling schemes. Although STRS CV results show a slight

increase in the accuracy, it is not significant. However, in most of the cases, the average tree size is larger for the STRS CV based tree construction. Although the increments are not statistically significant[2], there is a notable increase in average tree size for BS, GLS and SUR example sets compared to the increments of the other example sets. It can be seen from Table 3.3 that the class distributions for each of these example sets are highly imbalanced. Therefore, the CV samples, based on SRS, may under-represent the minority classes in the training set and consequently, the number of terminal nodes (pure regions in the feature space) required to classify those classes may not be found by the tree. In contrast, CV samples based on STRS have proper representation in the training sets and hence, these trees can have additional nodes for minority classes than the trees based on SRS.

Table 3.3 shows class-wise accuracies for both the sampling schemes. The SRS and STRS columns show the percentage of correct classifications for each class for CVs based on SRS and STRS respectively. In the STRS column, the parenthesised data shows the increase in the accuracy for the minority class. Note that this information is recorded only for those example sets for which class imbalance is present. Most of the example sets have imbalanced class distributions except for BH, LET, BNT, and SED. For those class imbalance example sets, CV STRS based trees have a higher classification accuracy for the minority classes than that of SRS CV trees. However, minority classes always have lower accuracy than other classes irrespective of the sampling scheme. When considering increments of classification accuracies for each minority class, almost all classes have gained at least 4% accuracy and in some cases it has risen up to 10% except for the CLI example set. Overall, there is an approximately 4% increment in the classification accuracy for minority classes due to the STRS CV scheme. Here, the increments of 12.3%, 17.8 % (both in the GLS example set) and 25.0% (in the SHUT example set) are omitted due to too few

[2]under 5% level of significance, if normality assumed

Table 3.3: Class-wise classification accuracies for SRS and STRS sampling schemes.

example set	SRS	STRS	Class size	example set	SRS	STRS	Class size
BS	66.1	71.0 (4.9)	49	**LET**	91.0	90.1	789
	95.1	95.6	288		76.1	76.2	766
	94.9	94.7	288		84.6	84.2	736
BH	83.0	85.2	246		79.5	79.9	805
	83.8	81.0	260		79.5	79.5	768
BC	96.3	96.5	444		80.2	79.5	775
	97.7	97.6	239		77.9	76.9	773
BUPA	44.8	49.6 (4.8)	145		70.9	70.1	734
	77.6	78.1	200		87.6	87.9	755
GLS	72.1	72.7	70		85.9	85.6	747
	64.1	63.7	76		77.0	76.8	739
	7.6	9.4 (1.8)	17		88.4	88.0	761
	32.3	44.6 (12.3)	13		90.4	91.1	792
	30.0	47.8 (17.8)	9		85.3	85.2	783
	79.0	84.1 (5.1)	29		77.7	77.8	753
HRT	80.1	78.2	150		84.1	84.3	803
	67.1	71.4 (4.3)	120		81.8	82.5	783
PIND	83.8	83.9	500		75.8	75.5	758
	52.4	54.0 (1.6)	268		77.2	77.6	748
SHUT	100.0	100.0	11478		85.0	84.9	796
	85.0	90.0 (5.0)	13		86.8	87.1	813
	97.0	99.0 (2.0)	39		87.5	87.6	764
	100.0	100.0	2155		91.0	90.8	752
	100.0	100.0	809		80.3	80.9	787
	45.0	70.0 (25.0)	4		85.5	85.6	786
	100.0	100.0	2		86.0	86.0	734
WINE	93.0	91.9	59	**BNK**	99.4	99.3	762
	87.7	87.0	71		99.4	99.7	610
	94.8	94.4	48				
				SED	86.7	88.0	70
SUR	95.8	91.9	225		95.7	96.4	70
	10.0	19.1 (9.1)	81		88.7	90.3	70
CLI	28.7	27.2 (-1.5)	46				
	97.6	97.9	494				

examples in the respective classes.

3.9.3 HHCART performances on example sets having mixed feature types

Experiments were performed to study the performance of the HHCART methods when the training example set contains both qualitative and quantitative features. QUEST (Loh & Shih, 1997) was used for comparison purposes since OC1, OC1-AP, and OC1-LC were not designed to handle oblique splits containing both types of features. Ten, 5-fold cross validations were used in the experiments and the average accuracies and tree sizes (over ten cross validations) are reported in Table 3.4. The Income example set comes with its own training and testing set of 30162 and 15060 examples respectively. Ten trees were induced, each using 90% of the training examples and the remaining 10% for pruning. The accuracy of all the trees were estimated using the same test set.

QUEST uses the following parameter setting: estimated prior, unit misclassification cost, zero standard error for pruning, linear splits, linear discriminant analysis for the split point and the minimum node size for splitting is 2. The HHCART algorithms were implemented as given in Subsection 3.9.1.

Table 3.4: Results of HHCART and QUEST.

example set	Decision Tree	Avg. Acc.	Avg. Size
Income	HHCART(A)	85.1 ± 0.2	32.7 ± 12.9
	HHCART(D)	$\mathbf{85.5} \pm 0.2$	59.5 ± 19.7
	QUEST	83.9 ± 0.2	68.0 ± 23.1
Bank	HHCART(A)	90.2 ± 0.12	22.58 ± 11.94
	HHCART(D)	$\mathbf{90.4} \pm 0.07$	44.4 ± 14.19
	QUEST	90.1 ± 0.1	27.0 ± 15.2
StatLog	HHCART(A)	85.1 ± 0.9	5.6 ± 1.9
	HHCART(D)	$\mathbf{85.8} \pm 0.7$	6.5 ± 3.0
	QUEST	85.65 ± 0.92	6.08 ± 3.6

For the Income example set, HHCART(A)'s performance is significantly (more

than 2 standard deviations) better than QUEST both in terms of the average accuracy and average tree size. For the other two datasets, HHCART(A) produces comparable accuracies with smaller trees. These results also suggest that the HHCART algorithms perform well in relatively high dimensions. Though HHCART(D) produces larger trees compared with HHCART(A), its classification accuracy is comparable with HHCART(A).

3.10 Conclusions and discussion

This chapter presents a novel algorithm, HHCART, for data classification. The proposed algorithm captures the orientation of examples belonging to each class by means of eigenvectors. Each eigenvector is then made parallel to the $\mathbf{e}_1$ direction using a Householder matrix and axis-parallel splits are conducted in the reflected space. These splits are oblique in the original feature space. The proposed method can induce an oblique split with less computational effort than some existing benchmark methods such as CART-LC (Breiman et al., 1984), OC1 (Murthy et al., 1994) without losing the accuracy and simplicity of the tree. Two versions of HHCART have been presented: HHCART(A) uses all possible eigenvectors of the estimated covariance matrices of respective classes whereas HHCART(D) uses only the dominant eigenvector of each class. The empirical results show that HHCART induces better trees, in terms of accuracy and the tree size, than that of other DT algorithms for most of the problem domains. The algorithm is designed to convert qualitative features into quantitative features and thereby HHCART is capable of handling both qualitative and quantitative features in the same oblique split. This enables HHCART to work in a wide range of real life data classification problems. Moreover, the effects of the imbalance classes on final tree accuracy and class-wise accuracy were empirically studied. The classification accuracy for the minority class always stays low compared

to that of the majority classes. However, the use of CV based on stratified random sampling can raise the classification accuracy of the minority classes on average by 4%.

In HHCART, the creation of new feature (or artificial feature) spaces using the Householder reflection can be viewed as an attempt to expand the search space. In the literature (see Chapter 2), it can be found that many DT methods try to expand the feature space to find better splits. For example, the methods based on statistical techniques try to explore new search spaces by creating artificial features. However, the way these new features are created is different from what is implemented in the HHCART algorithm. For example, Henrichon and Fu (1969) propose using the dominant eigenvector of each class as artificial features. This differs from HHCART as follows: (a) HHCART uses all possible eigenvectors of each class, and (b) HHCART creates a new feature space for each eigenvector. Moreover, OC1 uses the hill climbing algorithm and a randomisation procedure in turn to find a better split. Furthermore, it uses different initial locations to start the hill climbing algorithm. All these strategies help OC1 to expand the search space. However, the feature space expansion and the way HHCART finds splits is computationally cheaper than that of OC1. Therefore, the empirical results and the reduction in time complexity of HHCART show that the proposed algorithm is a good alternative for optimisation based tree building algorithms.

Chapter 4

HHCART with massive example sets

The DT algorithms discussed so far, including the work of this thesis, are based on a serial computing memory resident approach. In order to build a decision tree, these algorithms need the full example set to be entirely loaded into the computer memory, (hence memory resident), and the algorithm executes each instruction sequentially, (hence serial). However, this approach may not be feasible when such an algorithm is applied to massive example sets. Two solutions are proposed in the literature: (a) Disk resident decision tree algorithms, and (b) Parallel implementation of decision tree algorithms. This chapter briefly presents some early work in this regard and shows how HHCART can be modified accordingly.

4.1 Early attempts of decision tree induction for large example sets

Data mining applications often come with massive data sets (Srivastava A, Han, Kumar, & Singh, 2002). Inducing oblique decision trees for such data sets using conventional methods, for example CART, OC1, SADT, can consume considerable

time. Consequently various attempts have been made to increase the efficiency of tree growing algorithms including the work of this thesis. The conventional and new methods are based on a memory resident, serial computing approach. However, it has come to a point where none of these methods are effective enough to cater to the ever increasing size of data. Decision tree induction algorithms face two challenges in the presence of massive example sets. First, the size of the example set often exceeds the capacity of the memory and hence, memory resident algorithms become impractical to implement. Thus, disk resident algorithms have been proposed. Second, as the number of examples increases, even searching for the best axis-parallel split takes excessive time. However, with the invention of parallel computing architecture, parallel decision tree algorithms have been introduced and are widely used to solve large scale classification problems.

4.1.1 Disk resident decision tree algorithms

Mehta, Agrawal, and Rissanen (1996) propose an axis-parallel decision tree algorithm, SLIQ, for disk resident data. However, SLIQ still requires that some information resides in memory, so it assumes that the capacity of the memory is large enough to store the data. The results show that the algorithm scales well with a large number of examples and features. SPRINT (Shafer, Agrawal, & Methta, 1996), is an axis-parallel algorithm which removes all memory restrictions of SLIQ. For each feature, SPRINT initially creates a separate structure called a feature list which comprises the feature values, their class labels and the example numbers (or record numbers). Table 4.1 shows the hypothetical example set and Table 4.2 shows the two feature lists constructed. Feature lists are maintained in the disk and they are sorted according to the feature values. The initial feature lists correspond to the root node of the decision tree. All feature lists are read sequentially from the disk and for each feature

Table 4.1: Hypothetical example set.

Example No.	Y	X1	X2
1	1	10	5
2	2	6	8
3	2	7	7
4	1	15	10
5	1	9	12
6	1	12	14
7	2	11	16
8	2	4	10
9	1	15	1
10	1	14	15

list, one row at a time is read into the memory. A separate node level frequency table for each feature list is maintained in the memory and is updated as each row is read into the memory from the feature list being processed. Frequency tables are created when the node is created. For example, Table 4.3a shows the initial frequency table of feature X_1 (before the first record is read from the feature list given in Table 4.2a) and Table 4.3b shows the updated frequency table after $X_1 = 10$ record is read. At each update of the frequency table, the Gini index, a measure of impurity, is calculated. At the end of the reading, the best split for the feature is saved and the frequency table is deleted before processing the next feature list. Once all the features are processed, the best split at the node can be found. The above steps are given in Algorithm 4.

Each feature list is then partitioned according to the best split found. The feature which gives the best split can easily be partitioned by reading the feature values again into the memory and applying the split test. This creates two child feature lists corresponding to the feature being processed, one for the left node and the other for the right node and are stored in the disk. At the same time the example number and the node (left or right), to which the example has been sent by the test, are written in a separate table called a hash table. The hash table is used to split the other

Table 4.2: Constructed feature lists for the example set in Table 4.1.

(a) Feature list for X_1.

Example No.	Y	X_1
8	2	4
2	2	6
3	2	7
5	1	9
1	1	10
7	2	11
6	1	12
10	1	14
4	1	15
9	1	15

(b) Feature list for X_2.

Example No.	Y	X_2
9	1	1
1	1	5
3	2	7
2	2	8
4	1	10
8	2	10
5	1	12
6	1	14
10	1	15
7	2	16

Table 4.3: Frequency table created when reading X_1 feature list into the memory.

(a) Frequency table of X_1 at the beginning.

	Y=1	Y=2
$X_1 < 4$	0	0
$X_1 > 4$	6	4

(b) Frequency table after reading $X_1 = 10$.

	Y=1	Y=2
$X_1 \leq 10$	2	3
$X_1 > 10$	4	1

feature lists. Assume that $X_1 = 8$ gives the best split for the node. The hash table created is given in Table 4.4 where 2 and 3 are the left and right nodes respectively. Algorithm 5 outlines these steps. The rest of the feature lists are split as follows: (a) for each feature list, records are read into the memory one at a time; (b) for each record, its observation number is mapped to the hash table and the corresponding node is obtained; (c) if the node is the left node, then the record is written to the left child feature list or otherwise to the right child feature list of that feature; and (d) at each new node, a new frequency table for each feature list is created and initialised as given in Table 4.3a. These steps are given in Algorithm 6.

These three algorithms are executed sequentially at each node until the node becomes homogeneous. The SPRINT algorithm is modified in this research to develop

Table 4.4: Hash table.

Example No.	Node
8	2
2	2
3	2
5	3
1	3
7	3
6	3
10	3
4	3
9	3

the disk resident version of HHCART and is given in Subsection 4.1.1.1.

4.1.1.1 The disk resident implementation of HHCART

The methodology for disk resident version of HHCART is proposed. Therefore, even if there is no parallel computing facility available, HHCART can still be used with massive example sets. Let $\mathfrak{D}_t$ be the example set at a non-terminal node t and define the class variable $Y \in \{1, 2, \ldots, C\}$. The steps of the basic HHCART algorithm at node t are given below.

[1] Convert all qualitative features to respective CRIMCOORDs.

[2] Partition $\mathfrak{D}_\mathfrak{t}$ based on the class labels. Let $\mathfrak{D}_\mathfrak{t}^\mathfrak{i}$ be the example set belonging to i^{th} class at node t. Then, $\mathfrak{D}_\mathfrak{t} = \bigcup_{i=1}^{C} \mathfrak{D}_\mathfrak{t}^\mathfrak{i}$.

[3] Compute covariance matrix for each $\mathfrak{D}_\mathfrak{t}^\mathfrak{i}$, where $i = 1, 2, \ldots, C$.

[4] Perform eigen analysis on each covariance matrix (there are C covariance matrices).

```
Define PV = ∞, Δ(I) = 0;
p = number of features;
for i=1:p do
    PV = ∞ a temporary variable;
    RC = 1 the record counter;
    open feature list(FL_i) corresponding to feature_i;
    Read RC^th record of i^th FL_i;
    while NOT End Of File do
        x_{i,RC} = RC^th feature value;
        Update the frequency table;
        Calculate Gini index and the impurity reduction ;
        if impurity reduction > Δ(I) then
            BSP = (x_iRC + PV)/2 ;
            SF = feature_i ;
            Δ(I) = impurity reduction;
        end
        PV = x_{i,RC};
        RC = RC + 1;
        Read RC^th record of i^th feature list ;
    end
    Delete frequency table from the memory;
end
```

Algorithm 4: The procedure used by SPRINT to find the best split.

[5] For each eigenvector (there are Cp eigenvectors altogether and p is the number of features), construct a Householder matrix. Let H_t^{ij} be the Householder matrix defined by the j^{th} eigenvector of i^{th} class at node t.

[6] Transform $\mathfrak{D}_t$ based on each Householder matrix and construct transformed spaces. Let $\hat{\mathfrak{D}}_t^{ij}$ be the transformed example set based on H_t^{ij} matrix.

[7] Perform axis-parallel searches in all transformed spaces. There are Cp transformed spaces.

[8] Find the best split at the node and divide $\mathfrak{D}_t$ according to the best split found.

```
Open SF list, RC = 1
Read RC^th record of SF list ;
while NOT End Of File do
    x_{i,RC} = RC^th feature value;
    if x_{i,RC} ≤ BSP then
        write x_{i,RC} to the left node SF_{left} list;
        append the hash table with the observation number and the left node
        number;
    else
        write x_{i,RC} to the right node SF_{right} list;
        append the hash table with the observation number and the right node
        number;
    end
end
```

Algorithm 5: The procedure used by SPRINT to split the feature giving the best split. SF and BSP are defined in Algorithm 6.

[9] Recursively apply step 1 to step 8 until each child node meets one of the stopping criteria specified in Section 3.5.

Since the HHCART algorithm uses axis-parallel splits in the transformed spaces, the split finding mechanism of the SPRINT algorithm can easily be utilised to find the best split in HHCART. However, HHCART has some additional work to do (step 1 to 6 in the above list) before it proceeds to axis-parallel splits. Let M_m and M_d be the memory capacity of the machine and the memory requirement of $\mathfrak{D}_t$ respectively. Assume that $M_d >> M_m$ and hence, $\mathfrak{D}_\mathfrak{t}$ cannot be loaded into the memory as a whole. Therefore, $\mathfrak{D}_\mathfrak{t}$ is read into the memory in blocks and thus, the number of blocks K, for $\mathfrak{D}_\mathfrak{t}$, is defined as

$$K = \left\lceil \frac{M_d}{M_m} \right\rceil \tag{4.1.1}$$

One of the important characteristics of HHCART is its ability to handle both qualitative and quantitative features in the same oblique split. Qualitative features are

```
for i=1:p do
    Create left frequency table (LFREQ) and right frequency table (RFREQ).
    if Feature_i ≠ SF then
        Read RC^th record of Feature_i list ;
        while NOT End Of File do
            Get the Node Number corresponding to the observation number
            from the hash table
            if Node Number is a left node then
                Write RC^th record of Feature_i list to left node Feature_i list;
                update LFREQ;
            else
                Write RC^th record of Feature_i list to right node Feature_i list;
                update RFREQ;
            end
        end
    end
end
```

Algorithm 6: The procedure used by SPRINT to split features.

first transformed into new variables called CRIMCOORDs and they are then amalgamated with existing quantitative features before constructing covariance matrices. The CRIMCOORD transformation methodology, based on Loh and Shih (1997), is given in Section 3.7 and requires the entire feature to be loaded into the memory. Therefore, it cannot be used in the disk resident version of HHCART. A new methodology is proposed to compute CRIMCOORDs, which does not require the feature to be fully loaded into the memory as a whole but which guarantees the same CRIMCOORDs as those given in Loh and Shih (1997). The proposed CRIMCOORD construction methodology is given below:

Let X be a qualitative feature taking values in the set $\{1, 2, \ldots, L\}$ and the class variable $Y \in \mathbb{C} = \{1, 2, 3, ..., \mathrm{C}\}$. Assume the number of levels of X is known. In the most general case, we assume that the entire X feature cannot be read into the memory as a whole and hence, it is proposed to read X in blocks. Let the number of

blocks be K_q (usually $K_q << K$, where K is defined in equation (4.1.1)). Assume that the m^{th} block of X and Y are read into the memory where $m = 1, \ldots, K_q$. Each level of X is first transformed into an L-dimensional dummy vector $\mathbf{v}^i_{mj} = \{\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_L\}$ as:

$$\text{For} \quad l = 1, \ldots, L, \qquad \tilde{v}_l = \begin{cases} 1 & \text{if } X = l \\ 0 & \text{otherwise.} \end{cases} \tag{4.1.2}$$

where $\mathbf{v}^i_{mj}$ is the j^{th} dummy vector of the i^{th} response class of the m^{th} block. The CRIMCOORD construction procedure requires computation of matrices B, W and T which are defined in equations (3.7.2a), (3.7.2b) and (3.7.2c) respectively. However, these definitions cannot be used for the disk resident version as some of the quantities in these equations are not available until the entire feature is read into the memory. Therefore, a new formulation of the same matrices is defined and they are given below.

Let $\bar{\mathbf{v}}^i$ and $\bar{\mathbf{v}}^i_m$ be the mean dummy vector of the i^{th} response class and mean dummy vector of the i^{th} response class in the m^{th} block respectively. Also, let n^i_m be the number of examples of i^{th} response class in the m^{th} block. Then:

$$\begin{aligned} \mathrm{W} &= \sum_{i=1}^{C} \sum_{m=1}^{K_q} \sum_{j=1}^{n^i_m} (\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i)(\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i)^T \\ &= \sum_{i=1}^{C} \sum_{m=1}^{K_q} \sum_{j=1}^{n^i_m} \left[(\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i_m) - (\bar{\mathbf{v}}^i - \bar{\mathbf{v}}^i_m)\right] \left[(\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i_m) - (\bar{\mathbf{v}}^i - \bar{\mathbf{v}}^i_m)\right]^T \\ &= \sum_{i=1}^{C} \sum_{m=1}^{K_q} \sum_{j=1}^{n^i_m} \left[(\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i_m)(\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i_m)^T + (\bar{\mathbf{v}}^i - \bar{\mathbf{v}}^i_m)(\bar{\mathbf{v}}^i - \bar{\mathbf{v}}^i_m)^T\right] \end{aligned} \tag{4.1.3a}$$

Therefore,

$$W = \sum_{i=1}^{C} \sum_{m=1}^{K_q} \gamma^i_m + \sum_{i=1}^{C} \sum_{m=1}^{K_q} n^i_m (\bar{\mathbf{v}}^i - \bar{\mathbf{v}}^i_m)(\bar{\mathbf{v}}^i - \bar{\mathbf{v}}^i_m)^T \tag{4.1.3b}$$

where $\gamma^i_m = \sum_{j=1}^{n^i_m} (\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i_m)(\mathbf{v}^i_{mj} - \bar{\mathbf{v}}^i_m)^T$.

Also:

$$\begin{aligned} \mathrm{B} &= \sum_{i=1}^{C}\sum_{m=1}^{K_q}\sum_{j=1}^{n_m^i}(\bar{\mathbf{v}}^i - \bar{\mathbf{v}})(\bar{\mathbf{v}}^i - \bar{\mathbf{v}})^T \\ &= \sum_{i=1}^{C} n^i(\bar{\mathbf{v}}^i - \bar{\mathbf{v}})(\bar{\mathbf{v}}^i - \bar{\mathbf{v}})^T \end{aligned} \tag{4.1.4a}$$

and

$$\mathrm{T} = \mathrm{B} + \mathrm{W} \tag{4.1.4b}$$

where n^i is the number of examples in the i^{th} response class.

CRIMCOORD is the eigenvector, $\mathbf{a}^*$, corresponding to the maximum eigenvalue of $\mathrm{T}^{-}\mathrm{B}$, where B and T are defined in equations (4.1.4a), (4.1.4b) respectively and T^{-} is the generalised inverse of T. What follows is the proposed disk resident version of HHCART at node t:

[1] Convert all qualitative features to respective CRIMCOORD features: The following steps explain the construction of the CRIMCOORD for one qualitative feature X for the disk resident version of HHCART.

[i] Read m^{th} block of X and Y, where $m = 1, \ldots, K_q$.

[ii] Each value (actually a level of the qualitative variable) of m^{th} block of X, is transformed into an L-dimensional dummy vector, $\mathbf{v}_{mj}^i$, as given in function 4.1.2.

[iii] Calculate γ_m^i, (see equation (4.1.3b)) and $\bar{\mathbf{v}}_m^i$, defined as $\bar{\mathbf{v}}_m^i = \frac{1}{n_m^i}\sum_{j=1}^{n_m^i}\mathbf{v}_{mj}^i$, for each response class of the m^{th} block and store it in the memory along with n_m^i.

[iv] Once all K_q blocks have been read, compute the following quantities.

$$\bar{\mathbf{v}}^i = \frac{1}{n_i}\sum_{m=1}^{K_q} n_m^i \bar{\mathbf{v}}_m^i, \qquad \text{where } n^i = \sum_{m=1}^{K_q} n_m^i$$

$$\bar{\mathbf{v}} = \frac{1}{n}\sum_{i=1}^{C} n^i \bar{\mathbf{v}}^i, \qquad \text{where } n = \sum_{i=1}^{C} n^i$$

Hence, W, B and T as given in equations (4.1.3b), (4.1.4a) and (4.1.4b) can be computed respectively.

[v] Compute $\mathbf{a}^*$ as the eigenvector corresponding to the maximum eigenvalue of $\mathrm{T}^{-}\mathrm{B}$, where T^{-} is the generalised inverse of T.

[vi] Now the CRIMCOORD value of $\mathbf{v}_{mj}^i$ can be computed by the dot product of two vectors, $\mathbf{v}_{mj}^i \cdot \mathbf{a}^*$, and each CRIMCOORD value is written to $\mathfrak{D}_t$ against the corresponding example.

[vii] CRIMCOORD value corresponding to the each level of X is written and maintained in a separate file. When predicting, the level of X of an unclassified observation is replaced by the corresponding CRIMCOORD, which is stored in the file.

[viii] Repeat steps [i] - [vii] to construct CRIMCOORD for each qualitative feature at node t.

[2] After step 1, $\mathfrak{D}_\mathfrak{t}$ contains quantitative features and CRIMCOORD for each qualitative feature. Partition $\mathfrak{D}_\mathfrak{t}$ based on the class label: This step can easily be implemented by reading K example blocks into the memory one at a time and then writing the examples belonging to each class into a separate file, $\texttt{CLASS\_FILE}_i$, where $i = 1, 2, \ldots, C$, in the disk.

[3] Read each $\texttt{CLASS\_FILE}_i$, where $i = 1, 2, \ldots, C$ into the memory and compute covariance matrix for each class. We take the most general case by assuming

the CLASS_FILE$_i$ is too large to fit into the memory. Therefore, read K example blocks one at a time into the memory and at each time compute; $\bar{\mathbf{x}}_k^i$, the mean vector of the k^{th} block of the i^{th} class:

$$\bar{\mathbf{x}}_k^i = \frac{\sum_{j=1}^{n_k^i} \mathbf{x}_{kj}^i}{n_k^i} \text{ and}$$

$$\mathrm{W}_k^i = \sum_{j=1}^{n_k^i} (\mathbf{x}_{kj}^i - \bar{\mathbf{x}}_k^i)(\mathbf{x}_{kj}^i - \bar{\mathbf{x}}_k^i)^T$$

where $\mathbf{x}_{kj}^i$ is j^{th} feature vector of i^{th} response class of the k^{th} block and n_k^i is the number of examples in the k^{th} block in the i^{th} class. Once all K blocks have been read, compute the quantities:

$$\bar{\mathbf{x}}^i = \sum_{k=1}^{K} \frac{n_k^i \bar{\mathbf{x}}_k^i}{n^i}, \qquad \text{where } n^i = \sum_{k=1}^{K} n_k^i$$

To give

$$\mathrm{B}^i = \sum_{k=1}^{K} n_k^i (\bar{\mathbf{x}}_k^i - \bar{\mathbf{x}}^i)(\bar{\mathbf{x}}_k^i - \bar{\mathbf{x}}^i)^T$$

$$\mathrm{W}^i = \sum_{k=1}^{K} \mathrm{W}_k^i$$

The covariance matrix for i^{th} class can be written:

$$\mathrm{COV}(X^i) = \frac{\mathrm{T}^i}{(n-1)}$$

$$\text{where, } \mathrm{T}^i = \sum_{k=1}^{K} \sum_{j=1}^{n_k^i} (\mathbf{x}_{kj}^i - \bar{\mathbf{x}}^i)(\mathbf{x}_{kj}^i - \bar{\mathbf{x}}^i)^T \text{ and } n = \sum_{i=1}^{\mathrm{C}} n_i$$

It can be shown that $\mathrm{T}^i = \mathrm{B}^i + \mathrm{W}^i$ (Johnson & Wichern, 2002) and hence, the covariance matrix of the complete example set of class i can be reproduced as,

$$\mathrm{Cov}(X^i) = \frac{(\mathrm{B}^i + \mathrm{W}^i)}{(n-1)}$$

[4] Perform eigen analysis on each covariance matrix, $\mathrm{COV}(X^i)$.

[5] For each eigenvector, construct a Householder matrix, H_t^{ij}.

[6] Transform $\mathfrak{D}_t$ based on each Householder matrix and construct transformed spaces. The transformed example set, ij^{th} transformed space, using H_t^{ij}, is defined by $\hat{\mathfrak{D}}_t^{ij} = \mathfrak{D}_t \mathrm{H}_t^{ij}$.

Since the entire $\mathfrak{D}_t$ cannot be read into the memory, it is proposed that again reading blockwise and let $\mathfrak{D}_{bt}$ be the b^{th} block of $\mathfrak{D}_t$. Once $\mathfrak{D}_{bt}$ read into the memory, the transformed values, $\hat{\mathfrak{D}}_{bt}^{ij}$ can be obtained by $\mathfrak{D}_{bt}\mathrm{H}_t^{ij}$ and $\hat{\mathfrak{D}}_{bt}^{ij}$ is appended into a separate file, $\texttt{TRANSDATA\_FILE}_{ij}$. After the b^{th} block is processed, $\texttt{TRANSDATA\_FILE}_{ij}$ contains, $\hat{\mathfrak{D}}_t^{ij}$, the entire transformed example set at node t corresponding to H_t^{ij}.

[7] In order to perform axis-parallel search, apply the split finding method of the SPRINT algorithm on $\texttt{TRANSDATA\_FILE}_{ij}$ to find the best split in the ij^{th} transformed space. Delete $\texttt{TRANSDATA\_FILE}_{ij}$ once the best split is found. Repeat steps 5 to 6, for each H_t^{ij} to find the best split at node t and apply the SPRINT splitting mechanism to partition the node based on the best split found.

[8] At each non-terminal node, perform steps 1-7 repeatedly until one of stopping criteria specified in Section 3.5 is met.

The major drawback of the proposed algorithm is that it requires frequent access to the disk for writing and reading. For example, feature lists have to be read from the disk during splitting and have to be write at the creation of new nodes. Furthermore, new set of files have to be created at each node. For example, C_t (the number of classes at the node t) number of files are needed to store the examples belonging to each class at node t whereas feature lists are created at the creation of new nodes.

The top-down DT induction algorithm builds a tree in two steps. First, it builds the tree until each node becomes homogeneous (or near homogeneous) with respect

to a particular class and second, it prunes the tree upward to reduce over-fitting. In the second stage, no matter how large the dataset is, it is reasonable to assume that the induced tree can be loaded into the memory for pruning. Therefore, in this work the tree pruning for massive example sets is not discussed.

4.1.2 Parallel computing architecture

Serial computing has been shown to be inefficient for solving large scale data mining problems Kufrin (1997); V, Grama, Gupta, and Karypis (1994). Therefore, to avoid this situation, some efforts were made to utilize parallel computing architecture to handle challenging data mining applications including decision tree algorithms (Ben-Haim & Tom-Tov, 2010; Joshi, Karypis, & Kumar, 1998). In parallel computing, multiple processors are used to solve the computational task. Figure 4.1 illustrates how a problem is solved in a parallel computing environment. Initially the prob-

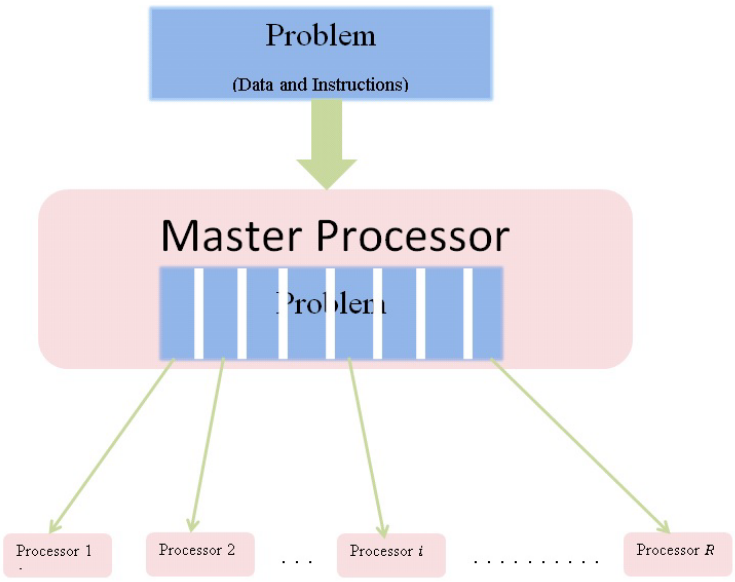


Figure 4.1: Schematic of Parallel computing Architecture.

lem is divided into disjoint parts that can be solved simultaneously. Each part is then processed by an individual processor (slave processor) simultaneously. Parallel computing architecture can be classified in following ways:

[1] Single Instruction, Multiple Data (SIMD)
All processing units execute the same instruction at any given clock cycle and each processing unit can operate on a different data element.

[2] Multiple Instruction, Single Data (MISD)
Each processing unit operates on the data independently via separate instruction streams and a single data stream is fed into multiple processing units.

[3] Multiple Instruction, Multiple Data (MIMD)
Every processor can execute a different instruction stream and every processor can work with a different data stream.

Further discussion of parallel computing architecture can be found in V et al. (1994). The following section briefly introduces the early attempts of DT construction in a parallel computing environment. The main processor is called a master processor and subordinate processors are called slave processors or processors. The master processor maintains the status of all the slave processors in the system and distributes the work to all the slave processors. Moreover, the operating system runs only on the master processor.

4.1.3 Parallel implementation of decision tree algorithms

Amado, Gama, and Silva (2001) present three strategies for implementing parallel decision trees, namely: (a) data parallelism, (b) task parallelism, and (c) hybrid parallelism. In data parallelism, the example set is partitioned into disjoint subsets and each subset is assigned to a separate processor. The partitioning of the example

set is done in two ways: (a) horizontal partitioning, and (b) vertical partitioning. In the horizontal partitioning the training set is partitioned evenly into R disjoint subsets, where R is equal to the number of processors, and each subset is assigned to one of the processors. Possible splits at each processor are sent to other processors for evaluation and processors then communicate to find the best split at the node. However, this method can suffer from high communication load specially for the nodes having less number of examples. In the vertical partitioning, the example set is partitioned on features. Each feature or a set of features is assigned to a processor and the processor-level best split is found for each processor. The best split for the node is then found by communicating the processor-level best splits among processors. This strategy may suffer from load load balancing as nodes responsible for continuous features have higher work load than the nodes working on qualitative features.

In the task parallelism, the entire dataset is assigned to a single processor to find the best split at the root node. From there onwards, each partition (or node) of the example set is assigned to a separate processor until the number of nodes equal the number of processors. When this happens, each processor proceeds with the construction of the decision sub-trees rooted at the node of its assignment. The major drawback of this strategy is unequal load balancing in processors. It is common in some problems for terminal nodes to appear in the early stages of the tree and that the processors responsible for these nodes remain idle afterwards. In all of these strategies, basically the example set is partitioned and distributed among the processors. At each processor, the same set of instructions are carried out. Therefore, each of these strategies can be implemented under Single Instruction Multiple Data architecture.

Hybrid parallelism combines both data and task parallelism strategies to overcome the drawbacks of each strategy. For the nodes having large number of examples are split according to the data parallelism while the nodes having fewer examples are

processed according to the task parallelism. More specifically, a node having fewer examples is assigned to a processor to build a sub tree rooted at the node.

Various algorithms have been proposed to construct decision trees in parallel computing architecture using above three strategies. Examples for horizontal data partitioning parallel algorithms can be found in: Amado et al. (2001); Joshi et al. (1998); Kufrin (1997); Shafer et al. (1996); Srivastava A et al. (2002); Yıldız and Dikmen (2007). The vertical data partitioning strategy is used by Yıldız and Dikmen (2007) while DTs based on task parallelism can be found in Srivastava A et al. (2002); Yıldız and Dikmen (2007). Robertson et al. (2014); Sreenivas, Alsabti, and Ranka (1999); Srivastava A et al. (2002) use hybrid parallelism, the combination of task and data parallelism.

In parallelising HHCART, we follow the SPRINT algorithm given in Shafer et al. (1996). Shafer et al. (1996) show that SPRINT scales well with the size of the example set and Joshi et al. (1998) state that the parallel formulation of continuous features in SPRINT is efficient. HHCART can handle both qualitative and quantitative features in the same oblique split. However, HHCART converts all qualitative features into quantitative features (CRIMCOORDS) therefore, it actually works only with quantitative features. Hence, HHCART is parallelised using SPRINT parallel algorithm which is explained below.

The parallel version of SPRINT (Shafer et al., 1996) uses a horizontal data partitioning approach where it assumes that each processor has its own private memory and disk. Initially, the training example set is distributed evenly over slave processors. Each processor then constructs a local feature list (feature lists are introduced in Subsection 4.1.1) for each feature. The key point is that SPRINT requires each local feature list to be a contiguous sorted section of the entire (global) feature list. That is, the first processor has the lowest values of the feature, the second has the next set of lower values of the feature and the last processor contains the highest

values of the feature. For this, SPRINT uses the parallel sorting algorithm given in DeWitt, Naughton, and Schneider (1991). As an example, the distributed feature lists for the features given in Table 4.2, over two processors, are given in Table 4.5 and Table 4.6. Each processor has a contiguous sorted section of each feature list.

Table 4.5: Segments of X_1 and X_2 features assigned to processor P1.

P1 Processor					
Example No.	Y	X1	Example No.	Y	X2
8	2	4	9	1	1
2	2	6	1	1	5
3	2	7	3	2	7
5	1	9	2	2	8
1	1	10	4	1	10

Table 4.6: Segments of X_1 and X_2 features assigned to processor P2.

P2 Processor					
Example No.	Y	X1	Example No.	Y	X2
7	2	11	8	2	10
6	1	12	5	1	12
10	1	14	6	1	14
4	1	15	10	1	15
9	1	15	7	2	16

Also, the processor level frequency tables (frequency tables are introduced in Subsection 4.1.1) are constructed for each feature list at the time of the parallel sorting. It is important to note that, for each feature, the frequencies in the table reflect the global count rather than the local count. This is accomplished by creating the frequency table when the node is created. As an example, the initial frequency table for feature X1 in processor P1 is given in Table 4.7. In each processor, the feature lists are read and the frequency tables are updated accordingly as in the SPRINT serial version. At each update of each frequency table the corresponding impurity is

Table 4.7: Initial frequency table of feature X_1 at processor P1.

	Y=1	Y=2
$X_1 \leq 4$	0	1
$X_1 > 4$	6	3

computed. At the end of the reading, each processor has the local best split for each feature and they are communicated among processors to find the best split at the node. Once the best split is found, the splitting is done in the following way. Each processor is responsible for splitting its feature lists. The feature giving the best split (say *winning-feature*) is split first and at the same time the information (example number and the node) needed to create the hash table (see Table 4.4) is gathered. Assume that the best split occurs at $X_1 = 8$. The information collected to construct the hash table is given in Table 4.8a and Table 4.8b where node 2 and 3 denotes the left and right child nodes respectively. Thus, after splitting *winning-feature*, the information is exchanged with all other processors. After the exchange, each processor constructs its own full hash table (see Table 4.9) and then uses it to partition the remaining feature lists in the processor.

Table 4.8: Processor level information to construct the hash table.

(a) Information collected in processor P1.

Example No.	Node
8	2
2	2
3	2
5	3
1	3

(b) Information collected in processor P2.

Example No.	Node
7	3
6	3
10	3
4	3
9	3

In summary, most of parallel decision tree algorithms are based on the data parallelism approach. Moreover, most of the parallel algorithms construct axis-parallel

Table 4.9: Hash table for X_1.

Example No.	Node
8	2
2	2
3	2
5	3
1	3
7	3
6	3
10	3
4	3
9	3

decision trees. Since the search along each axis is independent of each other, axis-parallel trees can easily be implemented in a parallel environment. To the best of author's knowledge, the information on oblique decision trees in parallel computing environment is limited. In fact Cantu-Paz and Kamath (2003) state that the parallelising existing oblique decision trees is difficult. However, Yıldız and Dikmen (2007) parallelise the Linear Discriminant Tree of Yildiz and Alpaydin (2000), which constructs oblique DT. Cantu-Paz and Kamath (2003) refer to Cantu-Paz (2000) and state that parallel decision trees based on evaluation algorithms can be implemented. The common feature of DTs, whether the split is axis-parallel or oblique, is search for the best split at a node independent of the other nodes. Hence, oblique decision tree algorithms can be implemented in the task parallelism approach, where each node is assigned to a processor. However, the data parallelism for oblique trees is difficult to achieve as the split finding mechanism embedded in most algorithms require entire example sets. In this regard, the HHCART methodology has an advantage as it can be parallelised using task, data and hence, hybrid strategies.

4.1.3.1 Parallel implementation of HHCART

Though HHCART is designed to produce oblique splits, it uses axis-parallel splits in a transformed space. Therefore, all the methods discussed in Section 4.1.3 can be used to parallelise the HHCART algorithm including data and task parallelism, which is a desirable characteristic of the HHCART approach. However, the task parallelism procedure is obvious, the work load of each node is assigned to a processor, thus it is not discussed in this thesis. The HHCART implementation under data parallelism is discussed. The main focus of this section is to briefly explain how HHCART can be implemented in a parallel environment. Therefore, the other issues, for example load balancing and communication load, are not discussed. Data parallism can be divided into two categories: (a) horizontal partitioning, and (b) vertical partitioning. The following section describes one option of how HHCART could assign work. Other choices exist which transform some or nearly all of the master processor's work to slave processors, which are briefly discussed in Section 4.2.

Vertical Partitioning

Case 1:

Resources Required:

[1] Cp^2 number of processors.

[2] The memory capacity of the master processor should be large enough to hold the entire dataset.

[3] The memory capacity of each slave processor should be large enough to hold the entire set of feature values coming to the processor.

Work flow

At each non-terminal node, the master processor is responsible for:

[1] Calculating the impurity at the node.

[2] Converting qualitative features to quantitative features.

[3] Finding classes' covariance matrices.

[4] Performing eigenanalysis on each covariance matrix.

[5] Constructing of Householder matrices using all the eigenvectors found in step [4].

[6] Reflecting the example set using each Householder matrix. Each Householder reflection creates p dimensional space so altogether there are Cp^2 axes (or new features) to search.

[7] Distributing each new feature (including with the class variable and impurity at the node) to each processor to search for the best split.

[8] Receiving the best split found and impurity reduction from each processor.

[9] Comparing impurity reductions and selecting the best split for the node.

[10] Splitting the node based on the best split.

At each non-terminal node, each slave processor is responsible for:

[1] Receiving the feature which is sent by the master processor.

[2] Performing axis-parallel splits.

[3] Sending the best split found and the impurity reduction to the master processor.

If the number of processors is less than Cp^2, then the work flow of case 1 can be slightly modified to accomplish the task. Assume the number of processors is $R < Cp^2$. Multiple features can then be assigned to each processor and the processor

is responsible for finding the best split for those features coming to the node. The fewer the number of processors the higher the number of features that can arrive at each processor.

Horizontal partitioning

In horizontal partitioning, the example set is divided evenly into R number of blocks, where R is the number of processors.

Resources Required:

[1] R number of processors.

[2] The memory capacity of the master processor should be enough to hold the entire dataset.

[3] The memory capacity of each slave processor should be enough to hold the feature values coming to the processor.

Work flow

At each non-terminal node, the master processor is responsible for:

[1] Calculating the impurity at the node.

[2] Converting qualitative features to quantitative features.

[3] Finding classes' covariance matrices.

[4] Performing eigenanalysis on each covariance matrix.

[5] Constructing of Householder matrices on all eigenvectors found in step 4.

[6] Reflecting the example set using each Householder matrix.

[7] Partitioning each transformed example set into R blocks.

[8] Sending each block to each processor.

[9] Receiving the best split found and impurity after the split from each slave processor.

[10] Calculating the impurity reduction for the best split for each feature.

[11] Comparing impurity reductions and selecting the best split for the node.

[12] Splitting the node based on the best split.

Each slave processor is responsible for:

[1] Receiving a block of the example set from the master processor.

[2] Searching for the best axis-parallel split while communicating other R processors. Here it is proposed to follow the mechanism used in the SPRINT parallel algorithm (Shafer et al., 1996) to find the best split.

[3] Sending the best split found and the corresponding impurity to the master processor.

4.2 Other possible work flow distributions

Other than options described above, some of the operations carried out at the master processor level (in vertical and horizontal partition methods) can be parallelised in many different ways. For example, instead of computing class covariance matrices at the master processor, the example set belonging to each class can be sent to a slave processor to compute the covariance matrix of that class and perform the eigenanalysis. It is also possible to construct the Householder matrix for each eigenvector at the same processor. Then all Householder matrices are sent to the master processor to reflect the dataset before the transformed example sets sent to slave processors for splitting.

Computation of CRIMCOORDs can also be assigned to slave processors. For each qualitative feature, HHCART computes a CRIMCOORD. Therefore, each qualitative feature can be assigned to a slave processor to compute its CRIMCOORD.

4.3 Discussion

This chapter presents several modified versions of HHCART for massive data classification problems. Two problems are considered: (a) how to induce DTs on example sets which are too large to fit into the memory, and (b) how to deal with the excessive induction time in the face of massive example sets. Various attempts have been proposed in the literature and most methods are designed for axis-parallel DTs. Since HHCART uses axis-parallel splits in a transformed space, the available methods can easily be applied to HHCART to handle massive example sets. For the problem given in (a), we present the modified serial implementation of the HHCART algorithm as a solution to example sets which are too large to fit into the memory. Problem (b) is solved in the parallel computing environment. HHCART is an easily parallelisable algorithm, which leads to the proposal for several implementations of HHCART to work under different (based on resources) parallel computing environments. However, implementation of the proposed versions of HHCART is left for future work.

Chapter 5

Alternative vectors for the Householder reflection

As presented in Chapter 3, the HHCART algorithm captures the orientation of a class by the principal eigenvector $\mathbf{d^1}$ of that class. The proposed heuristic argument assumes that $\mathbf{d^1}$ is parallel to the separating hyperplane of that class. Hence, the HHCART algorithm makes $\mathbf{d^1}$ parallel to $\mathbf{e_1}$ through a Householder reflection and an axis-parallel search is performed in the reflected space to find the best separating hyperplane. This chapter describes how HHCART can be used with other vectors to improve the classification results. Two cases are considered. In the first case, a normal vector to the angular bisector, introduced in the GDT algorithm (Manwani & Sastry, 2012), is used to construct the Householder reflection matrix. In the second case, *Class Representative Vectors* are introduced and used to construct Householder reflection matrices. For both cases, empirical evidence is provided to show the effectiveness of the Householder reflection.

The GDT classifier has a tuning parameter which should be estimated before tree building. Two estimation procedures are considered, namely: (a) two-stage ordinary CV, and (b) nested CV. The effect of these procedures on the accuracy and the tree size is thoroughly studied and recommendations are given based on the empirical evidence obtained.

5.1 Application of the Householder reflection to GDT

In this section, the methodological development of GDT is briefly explained. The interested reader may refer to Manwani and Sastry (2012) for a detailed description. For a two class problem, GDT finds a class separating hyperplane by means of an angular bisector of two hyperplanes called clustering hyperplanes. The computation of the angular bisector is given in the following section. As the main focus of this section is to obtain a separating hyperplane of the two classes, the discussion is limited to explain the method of finding the class separating hyperplane adopted in GDT. The complete tree growing algorithm of GDT is not explained and can be found in Manwani and Sastry (2012).

5.1.1 Finding the class separating hyperplane of the GDT algorithm

Here the method of obtaining the angular bisector of GDT (Manwani & Sastry, 2012) is briefly explained. The GDT algorithm is specifically designed to find separating hyperplanes for two-class problems. However, when dealing with a multi-class problem, GDT converts it into a two-class problem by forming two super-classes[1]. GDT tries to find two clustering hyperplanes, each one closest to examples in one class and furthest away from examples in the other class. Let $\mathbf{w}_1^T\mathbf{x} + b_1 = 0$ be the clustering hyperplane of one class denoted by C_+ and $\mathbf{w}_2^T\mathbf{x} + b_2 = 0$ be the clustering hyperplane of the other class denoted by C_-. Here distance is refereed to the Euclidean distance.

The distance of a set of points to a hyperplane is defined as an average of squared Euclidean distances. The average squared Euclidean distances of points of class C_+

[1] A supper-class is a class which contains one or more original classes.

from a hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ is:

$$D_+(\mathbf{w}, b) = \frac{1}{n_+\|\mathbf{w}\|^2}\acute{\mathbf{w}}^T\mathrm{A}\acute{\mathbf{w}} \tag{5.1.1}$$

where $\acute{\mathbf{w}} = [\mathbf{w}^T \quad b]^T$, $\mathrm{A}_{(p+1)\times(p+1)} = \sum_{\mathbf{x}_i \in C_+} \acute{\mathbf{x}}_i\acute{\mathbf{x}}_i{}^T$, $\acute{\mathbf{x}} = [\mathbf{x} \quad 1]^T$ and $n_+ = |C_+|$. Similarly, the average squared distances of points of class C_- from the hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ is given by:

$$D_-(\mathbf{w}, b) = \frac{1}{n_-\|\mathbf{w}\|^2}\acute{\mathbf{w}}^T\mathrm{B}\acute{\mathbf{w}} \tag{5.1.2}$$

where $\acute{\mathbf{w}} = [\mathbf{w}^T \quad b]^T$, $\mathrm{B}_{(p+1)\times(p+1)} = \sum_{\mathbf{x}_i \in C_-} \acute{\mathbf{x}}_i\acute{\mathbf{x}}_i{}^T$, $\acute{\mathbf{x}} = [\mathbf{x} \ \ 1]^T$ and $n_- = |C_-|$.

The algorithm tries to find a clustering hyperplane by maximising one of $D_+(\mathbf{w}, b)$ or $D_-(\mathbf{w}, b)$ while minimising the other. Since both criteria cannot be satisfied together, the ratio between $D_+(\mathbf{w}, b)$ and $D_-(\mathbf{w}, b)$ is maximised (or minimised). Hence, the problem can be re-expressed as:

$$\acute{\mathbf{w}}_1 = \mathrm{argmax}_{\acute{\mathbf{w}}\neq 0}\frac{\acute{\mathbf{w}}^T\mathrm{B}\acute{\mathbf{w}}}{\acute{\mathbf{w}}^T\mathrm{A}\acute{\mathbf{w}}} \tag{5.1.3}$$

where $\acute{\mathbf{w}}_1 = [\mathbf{w}_1^T \quad b_1]^T$ is the clustering hyperplane of class C_+ and similarly:

$$\acute{\mathbf{w}}_2 = \mathrm{argmin}_{\acute{\mathbf{w}}\neq 0}\frac{\acute{\mathbf{w}}^T\mathrm{B}\acute{\mathbf{w}}}{\acute{\mathbf{w}}^T\mathrm{A}\acute{\mathbf{w}}} \tag{5.1.4}$$

where $\acute{\mathbf{w}}_2 = [\mathbf{w}_2^T \quad b_2]^T$ is the clustering hyperplane of class C_-.
If A is in full rank, then the solutions to problems (5.1.3) and (5.1.4) are the solutions of the generalised eigenvalue problem given by:

$$\mathrm{B}\acute{\mathbf{w}} = \lambda\mathrm{A}\acute{\mathbf{w}}. \tag{5.1.5}$$

Specifically, the solutions $\acute{\mathbf{w}}_1$ and $\acute{\mathbf{w}}_2$ are eigenvectors corresponding to the maximum and minimum eigenvalues of $\mathrm{A}^{-1}\mathrm{B}$ respectively. If A suffers from rank deficiency, particularly for small samples, then the computation of A^{-1} becomes complex and difficult (Chen, Liao, Ko, Lin, & Yu, 2000). However, Manwani and Sastry (2012) use

the method given in Chen et al. (2000), which is based on the original work of Liu, Cheng, Yang, and Liu (1992), to find $\acute{\mathbf{w}}_1$ and $\acute{\mathbf{w}}_2$. The following section summarises the development of the method.

5.1.2 Computing $\acute{\mathbf{w}}_1$ when matrix A suffers from rank deficiency

Liu et al. (1992) show that equation (5.1.3) is functionally equivalent to:

$$\acute{\mathbf{w}}_1 = \text{argmax}_{\acute{\mathbf{w}}\neq 0}\frac{\acute{\mathbf{w}}^T\text{B}\acute{\mathbf{w}}}{\acute{\mathbf{w}}^T\text{A}\acute{\mathbf{w}} + \acute{\mathbf{w}}^T\text{B}\acute{\mathbf{w}}} = \text{argmax}_{\acute{\mathbf{w}}\neq 0}\frac{\acute{\mathbf{w}}^T\text{B}\acute{\mathbf{w}}}{\acute{\mathbf{w}}^T(\text{A+B})\acute{\mathbf{w}}}. \tag{5.1.6}$$

Hence, instead of solving equation (5.1.3), Liu et al. (1992) solve problem (5.1.6) and $\acute{\mathbf{w}}_1$ will be the eigenvector corresponding to the maximum eigenvalue of $(\text{A+B})^{-1}\text{B}$. However, if (A+B) suffers from rank deficiency Liu et al. (1992) solve the problem in the complementary subspace of the null space of (A+B), that is, in the range space of (A+B). Chen et al. (2000) identify a major drawback in the method suggested by Liu et al. (1992) as follows:

$$\text{Let } F(\acute{\mathbf{w}}) = \frac{\acute{\mathbf{w}}^T\text{B}\acute{\mathbf{w}}}{\acute{\mathbf{w}}^T\text{A}\acute{\mathbf{w}} + \acute{\mathbf{w}}^T\text{B}\acute{\mathbf{w}}}. \tag{5.1.7}$$

Let the ideal solution to problem (5.1.6) be given by the vector $\mathbf{w_s}$. That is, when $\acute{\mathbf{w}} = \mathbf{w_s}$, $F(\mathbf{w}_s) = 1$ (the maximum value) while maximising $\mathbf{w}_s^T\text{B}\mathbf{w}_s$ and minimising $\mathbf{w}_s^T\text{A}\mathbf{w}_s$. However, there may be an arbitrary vector $\mathbf{q}$ which will also give $F(\mathbf{q}) = 1$, the maximum value of $F(\mathbf{q})$, if $\mathbf{q}^T\text{A}\mathbf{q} = 0$ and $\mathbf{q}^T\text{B}\mathbf{q} \neq 0$. That is, $\mathbf{q}$ minimises $\mathbf{q}^T\text{A}\mathbf{q}$ but may not maximise $\mathbf{q}^T\text{B}\mathbf{q}$. Under this circumstance, $\mathbf{q}$ will not be the ideal solution for problem (5.1.6). Furthermore, the former case $\mathbf{w}_s$ will not be distinguished with the latter case $\mathbf{q}$ as in both cases $F(.)$ attains to its maximum.

Hence, if A suffers from rank deficiency, Chen et al. (2000) project B onto the null space of A, denoted by $\mathcal{N}(A)$, and find the eigenvector corresponding to the

largest eigenvalue of the projected B as a solution to problem (5.1.6). However, it is noticed that this procedure fails when $\mathcal{N}(A) \subseteq \mathcal{N}(B)$. If $\mathcal{N}(A) \subseteq \mathcal{N}(B)$, then projecting B onto the $\mathcal{N}(A)$ results in zero and hence problem (5.1.6) suffers from the *zero divided by zero* problem. Hence, the following procedure is proposed by the author to find a clustering hyperplane when $\mathcal{N}(A) \subseteq \mathcal{N}(B)$ is true.

Let $\mathcal{R}(A)$ be the range space of A. Since A is a symmetric matrix, $\acute{\mathbf{w}} = \mathbf{u_1} + \mathbf{v_1}$ where $\mathbf{u_1} \in \mathcal{R}(A)$ and $\mathbf{v_1} \in \mathcal{N}(A)$. Then $\acute{\mathbf{w}}^T \mathrm{B} \acute{\mathbf{w}} = (\mathbf{u_1} + \mathbf{v_1})^T \mathrm{B}(\mathbf{u_1} + \mathbf{v_1}) = \mathbf{u_1}^T \mathrm{B} \mathbf{u_1}$ since $\mathbf{v_1}^T \mathrm{B} = \mathrm{B}\mathbf{v_1} = 0$ as $\mathbf{v_1} \in \mathcal{N}(A) \subseteq \mathcal{N}(B)$. Similarly, it can be shown that $\acute{\mathbf{w}}^T \mathrm{A} \acute{\mathbf{w}} = \mathbf{u_1}^T \mathrm{A} \mathbf{u_1}$. Therefore:

$$\frac{\acute{\mathbf{w}}^T \mathrm{B} \acute{\mathbf{w}}}{\acute{\mathbf{w}}^T \mathrm{A} \acute{\mathbf{w}}} = \frac{\mathbf{u_1}^T \mathrm{B} \mathbf{u_1}}{\mathbf{u_1}^T \mathrm{A} \mathbf{u_1}} \quad \text{when} \quad \mathcal{N}(A) \subseteq \mathcal{N}(B). \tag{5.1.8}$$

Hence, according to problem (5.1.3):

$$\mathbf{w}_{R_A} = \mathrm{argmax}_{\acute{\mathbf{w}} \neq 0} \frac{\acute{\mathbf{w}}^T \mathrm{B} \acute{\mathbf{w}}}{\acute{\mathbf{w}}^T \mathrm{A} \acute{\mathbf{w}}} = \mathrm{argmax}_{\mathbf{u_1} \neq 0} \frac{\mathbf{u_1}^T \mathrm{B} \mathbf{u_1}}{\mathbf{u_1}^T \mathrm{A} \mathbf{u_1}} \tag{5.1.9}$$

where values of $\acute{\mathbf{w}}$ giving $\acute{\mathbf{w}}^T \mathrm{A} \acute{\mathbf{w}} = 0$ are avoided.

Similarly for problem (5.1.4), $\acute{\mathbf{w}} = \mathbf{u_2} + \mathbf{v_2}$ where $\mathbf{u_2} \in \mathcal{R}(B)$ and $\mathbf{v_2} \in \mathcal{N}(B)$ and the solution can be written as:

$$\mathbf{w}_{R_B} = \mathrm{argmax}_{\acute{\mathbf{w}} \neq 0} \frac{\acute{\mathbf{w}}^T \mathrm{A} \acute{\mathbf{w}}}{\acute{\mathbf{w}}^T \mathrm{B} \acute{\mathbf{w}}} = \mathrm{argmax}_{\mathbf{u_2} \neq 0} \frac{\mathbf{u_2}^T \mathrm{A} \mathbf{u_2}}{\mathbf{u_2}^T \mathrm{B} \mathbf{u_2}} \quad \text{when} \quad \mathcal{N}(B) \subseteq \mathcal{N}(A) \tag{5.1.10}$$

where values of $\acute{\mathbf{w}}$ giving $\acute{\mathbf{w}}^T \mathrm{B} \acute{\mathbf{w}} = 0$ are avoided .

In summary, problem (5.1.3) is solved as follows:

[1] If A is in full rank, $\acute{\mathbf{w}}_1$ will be the eigenvector corresponding to the maximum eigenvalue of $\mathrm{A}^{-1}\mathrm{B}$.

[2] if A is not in full rank, but $\mathcal{N}(A) \not\subseteq \mathcal{N}(B)$, then project B onto $\mathcal{N}(A)$ and $\acute{\mathbf{w}}_1$ will be the eigenvector corresponding to the maximum eigenvalue of projected B. These two steps are discussed in Manwani and Sastry (2012).

[3] If A is not in full rank and $\mathcal{N}(A) \subseteq \mathcal{N}(B)$, project A and B onto $\mathcal{R}(A)$ (say A_{R_A} and B_{R_B} respectively) and, according to problem (5.1.9), $\mathbf{w}_{R_A}$ will be the eigenvector corresponding to the maximum eigenvalue of $A_{R_A}^{-1} B_{R_A}$.

The same procedure is used to solve problem (5.1.4) by replacing matrices suitably.

The situation listed in case [3] is illustrated below using a hypothetical set of examples given in Figure 5.1. Assume a two dimensional classification problem having the following example set and the scatter plot of the examples is given in Figure 5.1.

Table 5.1: A hypothetical example set.

Y	X_1	X_2
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
2	6	6
2	7	7
2	8	8
2	9	9

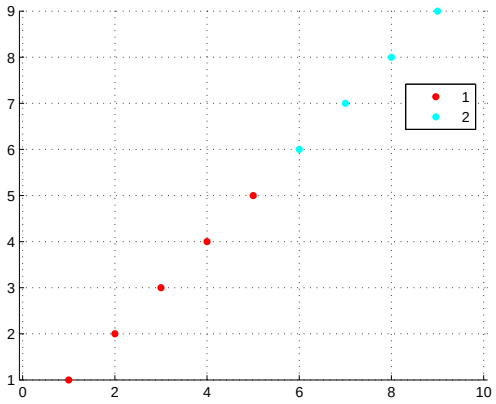


Figure 5.1: Scatter plot of the hypothetical data.

The computed A and B matrices are as follows:

$$\mathrm{A} = \begin{pmatrix} 0.1145 & 0.1145 & 0.0382 \\ 0.1145 & 0.1145 & 0.0382 \\ 0.0382 & 0.0382 & 0.0153 \end{pmatrix}$$

and

$$\mathrm{B} = \begin{pmatrix} 0.5800 & 0.5800 & 0.0867 \\ 0.5800 & 0.5800 & 0.0867 \\ 0.0867 & 0.0867 & 0.0133 \end{pmatrix}$$

The null spaces of both the matrices are the same: $\mathcal{N}(A) = \mathcal{N}(B) = [0.7071, -0.70710]^T$ and hence, the projected B onto the $\mathcal{N}(A)$ becomes zero. Therefore, the clustering hyperplane for the class 1 examples is found by projecting B and A onto $\mathcal{R}(A)$. Similarly, the clustering hyperplane for class 2 examples is found by projecting B and A onto $\mathcal{R}(B)$. Both clustering hyperplanes found are shown in Figure 5.2.

Once the clustering hyperplanes of the classes are found, the angular bisectors are computed if the clustering hyperplanes are not parallel. There are two angular

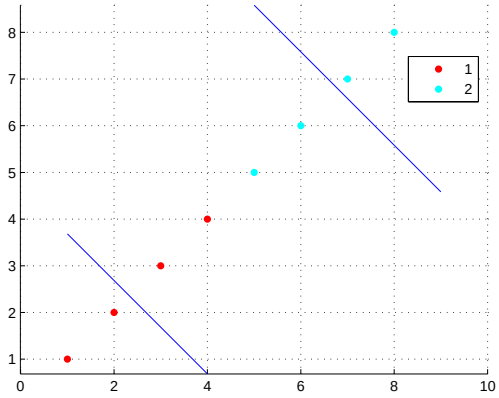


Figure 5.2: Class 1 and Class 2 clustering hyperplanes.

bisectors and one of the angular bisectors is chosen as the separating hyperplane based on the evaluation of an impurity function. If the clustering hyperplanes are parallel to each other, then the class separating hyperplane is parallel and halfway between them. The following proposed methodology aims to improve the classification results of GDT using the Householder reflection.

Let the normal vector of the selected hyperplane be $\acute{\mathbf{z}}$. Then the Householder reflection defined in equation (3.3.1) is used to make $\acute{\mathbf{z}}$ parallel to one of the feature axes so that axis-parallel splits can be searched in the reflected space. With this reflection, a new search space is constructed and oblique splits can be searched with minimal cost (using axis-parallel splits). Up to the computation of $\acute{\mathbf{z}}$, the multi-class problem is considered as a two-class problem. However, when searching for the best axis-parallel split in the reflected space, the proposed algorithm uses the full set of classes. The algorithm is called HHGDT (HouseHolder Geometric Decision Tree) and it works as follows: At each non-terminal node t, classes are grouped into two super-classes as suggested in Manwani and Sastry (2012). Then for each class, a clustering hyperplane is found and the angular bisectors are computed. The best

angular bisector is chosen using an impurity function. The normal vector of the chosen angular bisector (say $\acute{\mathbf{z}}$) is used to construct the Householder matrix which makes $\acute{\mathbf{z}}$ parallel to the $\mathbf{e}_1$ axis. The example set available at node t is then reflected using the Householder matrix and axis-parallel splits are searched in the reflected space. The best axis-parallel split is then chosen by evaluating an impurity function. All the classes available at the node are considered when evaluating the impurity function. The hyperplane found divides node t into two child nodes. The algorithm is recursively applied to all child nodes until the misclassification rate, (MisRate), at the node is not greater than a user specified threshold ϵ. The optimal value of ϵ is estimated using a separate cross validation procedure (Manwani & Sastry, 2012). An overview of the split finding method of the HHGDT algorithm at node t is given in Algorithm 7.

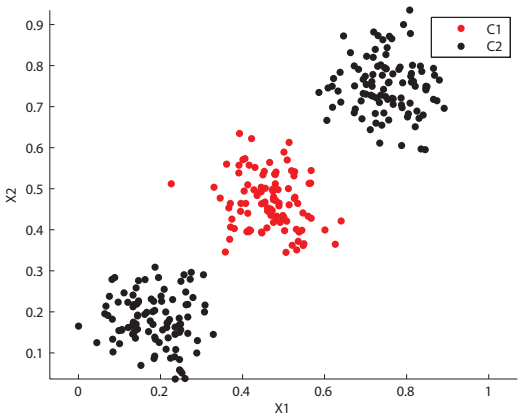
5.1.3 GDT vs HHGDT

In this section, two illustrations are given to show the effectiveness of the Householder reflection to improve the classification result of the GDT algorithm. Two artificially generated two dimensional example sets are classified using GDT and HHGDT. In the first illustration a two-class problem is considered and in the second illustration a five-class problem is considered.

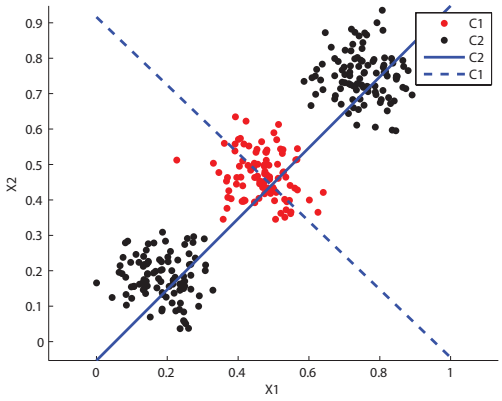
[1] Consider the two dimensional two-class classification problem given in Figure 5.3a. Class 2 (in black) is either side of Class 1 (in red). Figure 5.3b shows the clustering hyperplanes found at the root node by GDT. In the figure, the clustering hyperplane for class 2, depicted by the solid line, goes through the three data clouds. the dashed line depicts the clustering hyperplane for class 1 examples. The split at the root node is the selected angular bisector of these two hyperplanes and the full partition structure produced by GDT (lines

Data: Input: Examples at node t, $\mathfrak{D}_t$
initialisation;
$h_t = empty$;
Construct A and B matrices.
if *A is full rank* **then**
  $\acute{\mathbf{w}}_1$ = eigenvector corresponding to the maximum eigenvalue of $\mathrm{A}^{-1}\mathrm{B}$;
  else if $\mathcal{N}(A) \not\subseteq \mathcal{N}(B)$ **then**
    B_{N_A} = projection of B onto $\mathcal{N}(A)$; (see Manwani and Sastry (2012))
    $\acute{\mathbf{w}}_1$ = eigenvector corresponding to the maximum eigenvalue of B_{N_A} ;
    else if $\mathcal{N}(A) \subseteq \mathcal{N}(B)$ **then**
      Let $\mathrm{Q}_{r(A)}$ be the matrix whose columns are an orthonormal basis of $\mathcal{R}(A)$;
      $\mathrm{B}_{R_A} = \mathrm{Q}_{r(A)}^T \mathrm{B} \mathrm{Q}_{r(A)}$;
      $\mathrm{A}_{R_A} = \mathrm{Q}_{r(A)}^T \mathrm{A} \mathrm{Q}_{r(A)}$;
      $\mathbf{w}_{R_A}$ = eigenvector corresponding to the maximum eigenvalue of $\mathrm{A}_{R_A}^{-1}\mathrm{B}_{R_A}$;
      $\acute{\mathbf{w}}_1 = \mathrm{Q}_{r(A)}\mathbf{w}_{R_A}$;
    end
  end
end
if *B is full rank* **then**
  $\acute{\mathbf{w}}_2$ = eigenvector corresponding to the maximum eigenvalue of $\mathrm{B}^{-1}\mathrm{A}$;
  else if $\mathcal{N}(B) \not\subseteq \mathcal{N}(A)$ **then**
    A_{N_B} = projection of A onto $\mathcal{N}(B)$; (see Manwani and Sastry (2012))
    $\acute{\mathbf{w}}_2$ = eigenvector corresponding to the maximum eigenvalue of A_{N_B} ;
    else if $\mathcal{N}(B) \subseteq \mathcal{N}(A)$ **then**
      Let $\mathrm{Q}_{r(B)}$ be the matrix whose columns are an orthonormal basis of $\mathcal{R}(B)$;
      $\mathrm{A}_{R_B} = \mathrm{Q}_{r(B)}^T \mathrm{A} \mathrm{Q}_{r(B)}$;
      $\mathrm{B}_{R_B} = \mathrm{Q}_{r(B)}^T \mathrm{B} \mathrm{Q}_{r(B)}$;
      $\mathbf{w}_{R_B}$ = eigenvector corresponding to the maximum eigenvalue of $\mathrm{B}_{R_B}^{-1}\mathrm{A}_{R_B}$;
      $\acute{\mathbf{w}}_2 = \mathrm{Q}_{r(B)}\mathbf{w}_{R_B}$;
    end
  end
end
$\acute{\mathbf{z}}$ = the normal vector of the selected angular bisector of $\acute{\mathbf{w}}_1$ and $\acute{\mathbf{w}}_2$
Call Algorithm 8 with $\mathbf{d} = \acute{\mathbf{z}}$ $MinParent = 2$, $MisRate = \epsilon$ and $\tau = 0$

Algorithm 7: Overview of HHGDT algorithm at a single node.



(a) Scatter plot of two classes.



(b) Clustering hyperplanes of GDT at the root node. Solid line-clustring hyperplane of the black class, dashed line: clustering hyperplane for the red class.

Figure 5.3: GDT in two-class classification.

Data: Input: Examples at node t, $\mathfrak{D}_t$; permissible misclassification rate at a node, $MisRate$; direction vector for the Householder matrix, $\mathbf{d}$, and τ.
initialization;
Define N_t = Number of examples in $\mathfrak{D}_t$;
Define mp_t = misclassification rate at node t;
$h_t = empty$;
if *($N_t > Minparent$) and ($MisRate < mp_t$)* **then**
  if $\|\boldsymbol{e}_1 - \boldsymbol{d}\| \leq \tau$ **then**
    $H_t = I$, the Identity matrix;
  else
    Construct the Householder matrix H_t using $\mathbf{d}$ as shown in equation (3.3.1);
  end
  Reflect $\mathfrak{D}_t : \hat{\mathfrak{D}}_t = \mathfrak{D}_t \mathrm{H}_t$;
  Find the best axis-parallel hyperplane split, called h_t;
  Return h_t
end

Algorithm 8: Basic algorithm of HHCART at a single node.

in green) is given in Figure 5.4. The figure shows that although the classes are classified accurately, the tree size (or the number of partitions) is unnecessarily large. However, the resultant partition structure of the HHGDT algorithm for the same example set is given in Figure 5.5. HHGDT makes the normal vector of the angular bisector parallel to $\mathbf{e_1}$ using Householder reflection. This creates a new two dimensional space and the axis-parallel splits in the reflected space enables better splits to be found and hence, simplifies the tree.

[2] In the second illustration, a two dimensional five-class classification problem is considered. The scatter plot of the example set is given in Figure 5.6.

The GDT algorithm first groups five classes into two super-classes: Class 1 contains class A examples while class 2 contains the rest of the examples belonging to the other classes. Then the clustering hyperplane for the two super-classes is found and thereby computes the angular bisectors. The best angular bisector is selected as

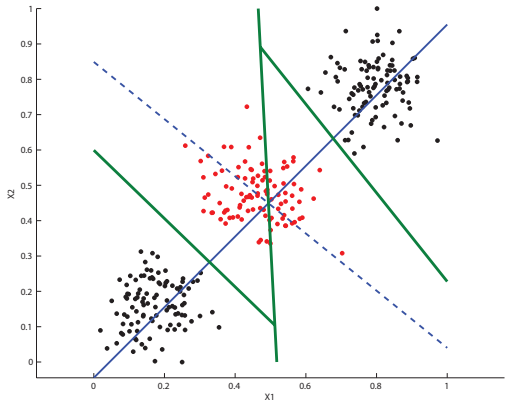


Figure 5.4: The final partition structure of GDT in solid green lines.

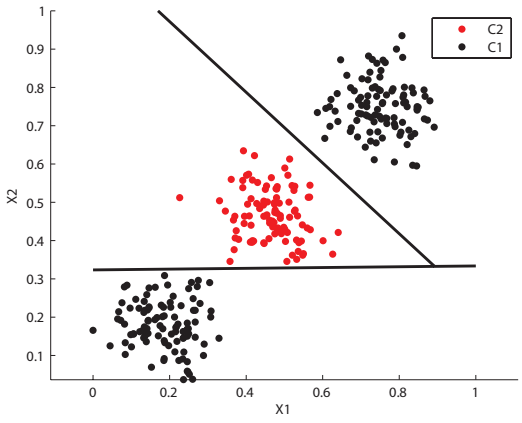


Figure 5.5: The final partition structure of HHGDT in solid black lines.

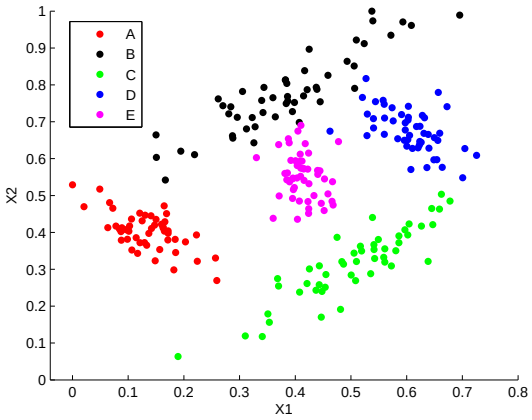


Figure 5.6: Scatter plot of examples belonging to five classes.

the first split and is shown in Figure 5.7. The HHGDT algorithm takes the normal vector of this angular bisector and makes it parallel to $\mathbf{e_1}$ using the Householder reflection. The reflected dataset is given in Figure 5.8a. Then the axis-parallel splits are searched along the axes of the reflected space and the best split found, in the original space, is given in Figure 5.8b.

The final unpruned partition structures of GDT and HHGDT are given in Figure 5.9 and Figure 5.10 respectively. It is clearly evident that HHGDT produces a more simplified tree than GDT for this dataset. The main reason for this is that the ability of HHGDT to expand the search space and hence, there is an opportunity to search for a better split with a minimal cost.

5.2 Experiments on real life datasets

The GDT algorithm developed by the author is somewhat different from the original algorithm as it contains a new procedure of finding clustering hyperplanes under the

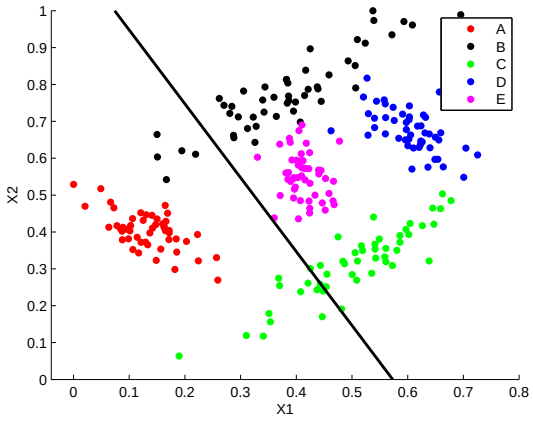
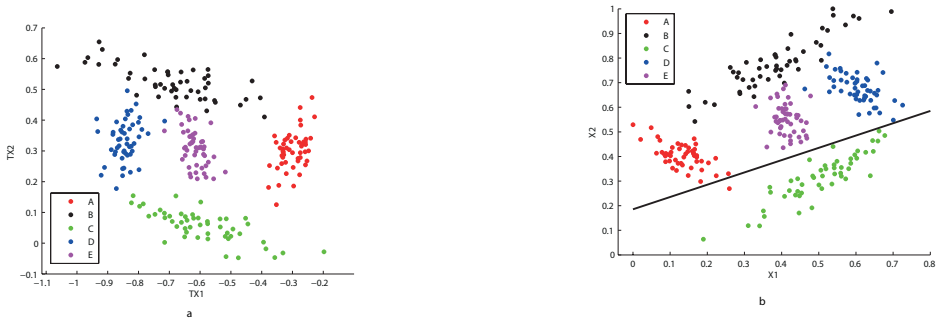


Figure 5.7: Selected angular bisector at the root node.



(a) Scatter plot of the reflected space.

(b) Best split found (in the original space) by the HHGDT.

Figure 5.8: HHGDT in multiclass classification.

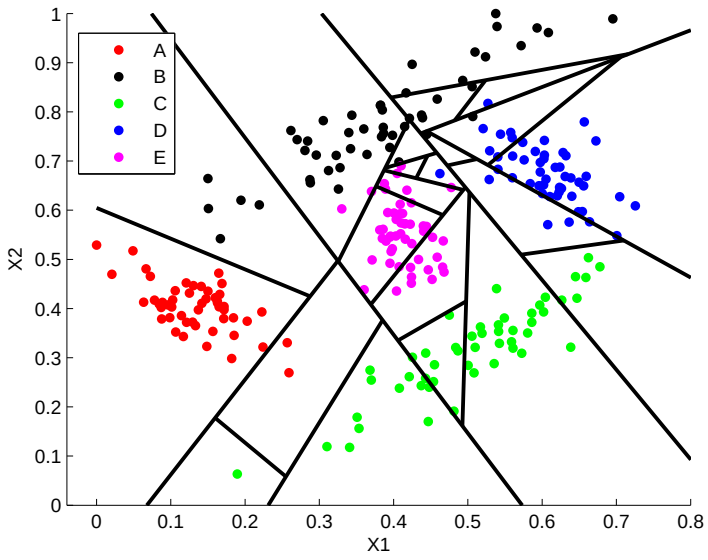


Figure 5.9: Final unpruned partition structure of GDT.

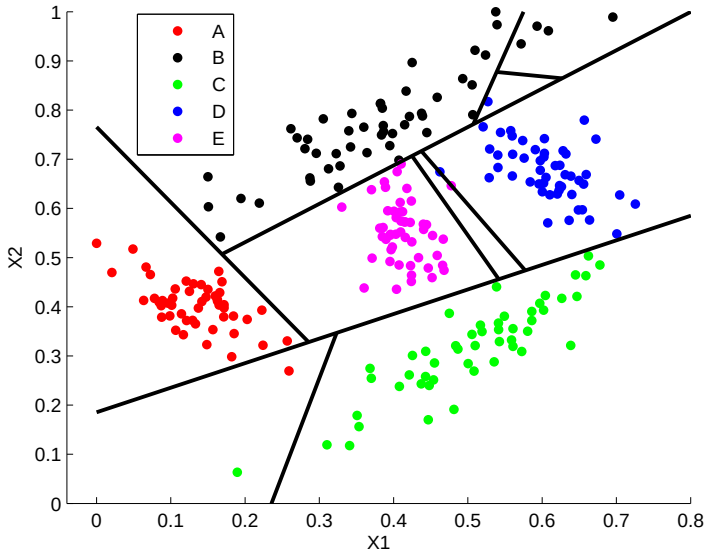


Figure 5.10: Final unpruned partition structure of HHGDT.

rank deficiency of matrices. GDT has one parameter, allowable node misclassification rate (ϵ), to determine whether the node is a leaf node. Manwani and Sastry (2012) use a two-stage ordinary CV procedure to estimate ϵ, in which, for each example set, ϵ is estimated using a 10-fold cross validation (CV) (first-stage) and the estimation of classification accuracy and tree size is done using another 10, 10-fold CVs (second-stage). That is, the estimation of ϵ is done separately from the second 10-fold CV procedure which is used to estimate the accuracy. This may lead to an overly optimistic parameter estimate of ϵ because of the following reason.

Consider a 10-fold CV partition set. At each time, nine folds are used to construct the tree while the other fold is kept for testing. However, the testing fold is not a purely independent, because the entire example set, including the examples in the current test set, has been used when estimating ϵ prior to the tree building process. This may lead to an optimistic result when using the test set. Therefore, it is worthwhile to investigate the performances of the **Nested CV procedure** which is specifically designed to estimate parameters when accuracy is being estimated. The procedure of nested CV is given in Appendix B.

Thus, two sets of experiments are conducted. In the first experiment, the performances are compared using the results obtained from two-stage ordinary CV. That is, as suggested by the authors (Manwani & Sastry, 2012), a separate 10-fold ordinary CV is run prior to the tree building to determine the optimal misclassification rate, ϵ_{TSOCV}. For each example set, the accuracy is examined by varying the misclassification rate from 0.05 to 0.4 with the step size of 0.01. However, the method of choosing the correct value for ϵ_{TSOCV} is not specifically presented by Manwani and Sastry (2012). Therefore, in the first experiment, the misclassification rate which produces the highest accuracy with the lowest tree size is chosen as the ϵ_{TSOCV}. The estimated ϵ_{TSOCV} is then used in the second CV to estimate the average accuracy.

In the second experiment, the nested CV procedure is used to determine the

optimal misclassification rate, ϵ_{NST}. Here the misclassification rate which produces the highest accuracy with the lowest tree size is chosen.

The same procedure is applied to determine the misclassification rate of HHGDT. For both algorithms, the Gini Diversity Index is used to identify the best splits. In the tree building, each method is run on the same data partitions constructed under the cross-validation sampling. Since the SHUT example set comes with a separate training and a test set, the CV procedure is not performed. Instead, GDT and HHGDT are trained on the training set and tested on the test set. Hence, the results for the SHUT example set do not contain the standard error. Moreover, in both algorithms ϵ is set to 0 for the SHUT example set because the author notices that as ϵ increases from 0, the minority classes tend to disappear from the tree. More specifically, the SHUT example set has a highly imbalanced class distribution (the class distributions is given in Table 3.3) in which class 7 has only two examples and GDT fails to produce a terminal node for this class when ϵ is set to 0.01.

5.2.1 Results of two-stage ordinary CV

Results of ten, 10-fold two-stage ordinary CV experiments are reported in Table 5.2 along with respective standard deviations.

Manwani and Sastry (2012) consider the intuitive confidence interval for the accuracy to be one standard deviation on either side of the estimated accuracy. If the confidence intervals of the accuracy of a problem for the two algorithms do not overlap, it can be concluded that the algorithm having the higher classification accuracy is significantly better than the other, say the 1-standard deviation rule (1-SD rule). According to the above results, it can be shown that the average size of the tree of HHGDT is significantly smaller than that of GDT for all the datasets except for BS, BNK, WINE, HRT, SUR and PIND. A substantial reduction can be seen in the

Table 5.2: Results of two-stage ordinary CV of HHGDT and GDT methods.

Dataset	DT	Avg. Acc.	Avg. Size	Dataset	DT	Avg. Acc.	Avg. Size
BS	GDT	91.8 ± 0.8	20.2 ± 4.1	PIND	GDT	76.6 ± 0.5	4.4 ± 1.3
	HHGDT	91.9 ± 1.0	15.4 ± 1.0		HHGDT	75.1 ± 1.1	8.21 ± 0.8
BH	GDT	82.2 ± 0.9	33.2 ± 1.6	WINE	GDT	95.8 ± 1.0	3.6 ± 0.3
	HHGDT	83.3 ± 1.3	9.5 ± 1.1		HHGDT	95.1 ± 1.0	3.8 ± 0.2
BC	GDT	96.3 ± 0.4	11.3 ± 0.6	SUR	GDT	73.8 ± 1.3	6.9 ± 0.7
	HHGDT	97.2 ± 0.3	2.1 ± 0.2		HHGDT	73.2 ± 1.3	5.2 ± 0.7
BUPA	GDT	67.8 ± 1.4	22.19 ± 1.2	HRT	GDT	82.3 ± 0.8	6.9 ± 1.4
	HHGDT	67.9 ± 1.9	10.9 ± 1.3		HHGDT	84.3 ± 0.9	2 ± 0
GLS	GDT	58.4 ± 3.0	45.3 ± 2.5	LET	GDT	84.8 ± 0.2	2480.5 ± 19.0
	HHGDT	67.4 ± 2.3	11.3 ± 1.1		HHGDT	85.3 ± 0.2	1400.6 ± 11.3
BNK	GDT	97.6 ± 0.03	2 ± 0	SHUT	GDT	99.75	187
	HHGDT	98.2 ± 0.1	2 ± 0		HHGDT	99.95	36
SEED	GDT	93.3 ± 1.7	6.6 ± 1.2	CLI	GDT	93.8 ± 0.6	4.5 ± 0.4
	HHGDT	94.0 ± 1.4	4.6 ± 0.5		HHGDT	94.6 ± 0.6	2.1 ± 0.1

SHUT example set. However, for PIND the tree size of HHGDT is significantly larger than that of GDT. HHGDT produces significantly better classification accuracies for BC, BNK, GLS, SHUT and LET. For the other data sets, there are no significant differences observed between accuracies of HHGDT and GDT. Therefore, the empirical results show that HHGDT outperforms GDT either in terms of accuracy or tree size for most of the datasets.

5.2.2 Results of nested CV

Table 5.3 shows the results of ten, 10-fold nested CV experiments along with respective standard deviations. The SHUT example set is not analysed as the only possible value for ϵ is 0.

The accuracies of the nested CV experiment are similar to that of the two-stage ordinary CV for the example sets. The first stage of the two-stage ordinary CV procedure uses the entire example set to estimate the optimal value, ϵ_{TSOCV}. In the second stage, using the same example set that was used in the first stage, another CV is carried out with ϵ_{TSOCV} to estimate the accuracy. Therefore, the accuracy of

Table 5.3: Results of nested CV of HHGDT and GDT methods.

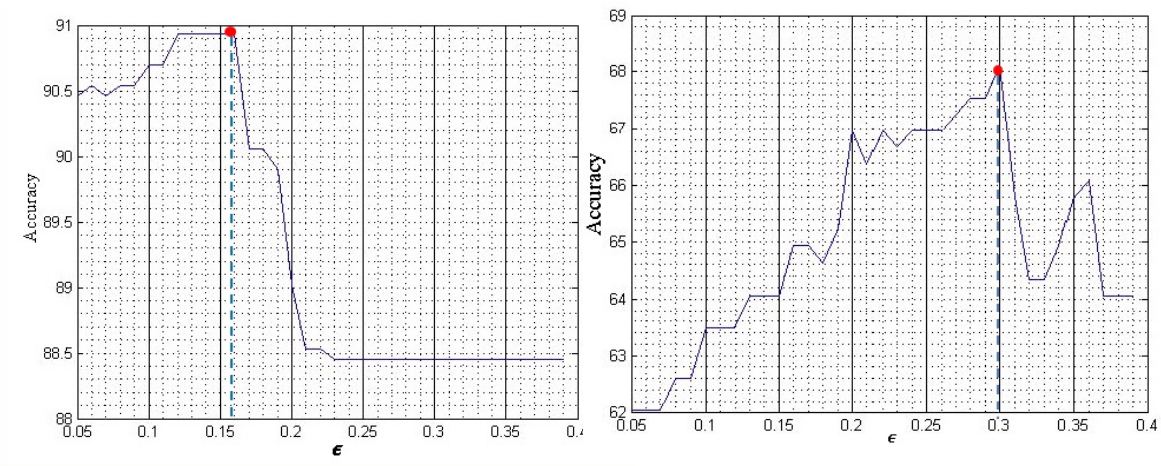
Dataset	**DT**	**Avg. Acc.**	**Avg. Size**	**Dataset**	**DT**	**Avg. Acc.**	**Avg. Size**
BS	GDT	91.8 ± 0.6	24.7 ± 3.7	PIND	GDT	76.5 ± 0.8	6.7 ± 2.9
	HHGDT	92.2 ± 1.1	17.6 ± 0.7		HHGDT	75.0 ± 1.0	10.0 ± 3.2
BH	GDT	82.00 ± 0.9	32.4 ± 3.3	WINE	GDT	95.9 ± 0.9	4.0 ± 0.4
	HHGDT	84.0 ± 1.4	16.2 ± 2.3		HHGDT	94.8 ± 1.0	3.4 ± 0.2
BC	GDT	96.3 ± 0.4	10.9 ± 0.8	SUR	GDT	73.6 ± 1.2	8.1 ± 1.7
	HHGDT	97.2 ± 0.3	2.1 ± 0.3		HHGDT	73.0 ± 1.3	5.1 ± 1.9
BUPA	GDT	67.9 ± 2.0	21.01 ± 4.93	HRT	GDT	82.1 ± 0.9	4.4 ± 1.8
	HHGDT	66.5 ± 2.3	11.2 ± 2.4		HHGDT	84.4 ± 0.8	2.2 ± 0.3
GLS	GDT	57.8 ± 2.4	47.03 ± 3.9	LET	GDT	85.1 ± 0.8	2477.6 ± 21.6
	HHGDT	67.18 ± 2.6	14.2 ± 2.4		HHGDT	85.4 ± 0.7	1581.6 ± 20.4
BNK	GDT	97.6 ± 0.04	2.1 ± 0.2	CLI	GDT	94.2 ± 0.5	5.1 ± 0.6
	HHGDT	98.2 ± 0.1	2.0 ± 0.03		HHGDT	94.6 ± 0.7	2.6 ± 0.5
SEED	GDT	93.9 ± 1.1	8.2 ± 1.3				
	HHGDT	93.8 ± 1.2	3.8 ± 0.4				

two-stage ordinary CV may have an optimistic bias. On the other hand, nested CV keeps the test set independent of the estimation of ϵ_{NST} and hence, the results should be more realistic. But the observed results in Table 5.2 and Table 5.3 indicate that they are not optimistic under two-stage ordinary CV when compared to the results of nested CV, and thus needed further investigation. Note that this observation is made using the results obtained by averaging the accuracies over ten CVs. In order to get a precise view, the averages should be compared fold by fold. Hence, another experiment is carried out. The steps and a detailed discussion of the experiment are given below:

[1] Two example sets are selected (BS and BUPA) such that the classification task of one (BS) is relatively easier than that of the other (BUPA). For the GDT algorithm, the average accuracies of the BS and BUPA example sets are 91.8% and 68.1% under the two-stage ordinary CV respectively. Hence, these two example sets are selected to cover a wide range of degree of classifiability.

[2] Two-stage ordinary CV procedure is run on each example set as follows.

[a] A 10-fold CV is performed to estimate ϵ_{TSOCV} for each of the example sets prior to the second stage of the two-stage ordinary CV. Figure 5.11 shows the change in accuracy with respect to ϵ for each example set. The red dot shows the highest accuracy while the point where the blue dash-line meets ϵ axis gives ϵ_{TSOCV}. It can be seen that the accuracy fluctuates



(a) BS. (b) BUPA.

Figure 5.11: Variation of accuracy with ϵ for the two example sets.

between approximately $66 - 68\%$ when ϵ is in between 0.2 and 0.3 for the BUPA example set and for the BS example set the accuracy varies between $89 - 91\%$ when ϵ is in the range of $0.05 - 0.2$. For the BS example set, the estimated ϵ_{TSOCV} is taken as 0.16, as it produces the highest accuracy while simplifying the tree. For the BUPA example set ϵ_{TSOCV} is taken as 0.3.

[b] Using the chosen ϵ, two repetitions of ordinary 10-fold CVs are run on each example set to estimate the average accuracy. The results are given in Table 5.4.

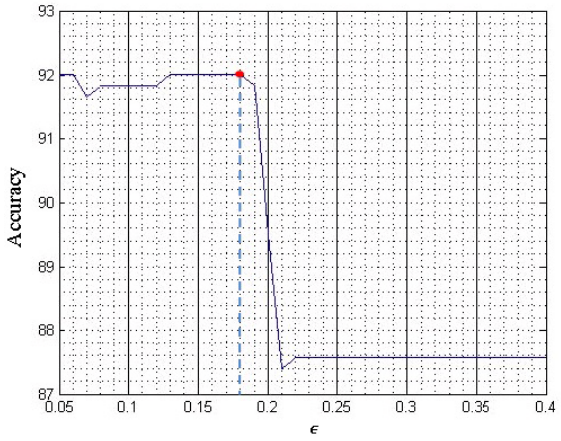
[3] Two repetitions of 10-fold nested CV are performed to estimate the accuracy of

Table 5.4: Accuracies of BS and BUPA example sets.

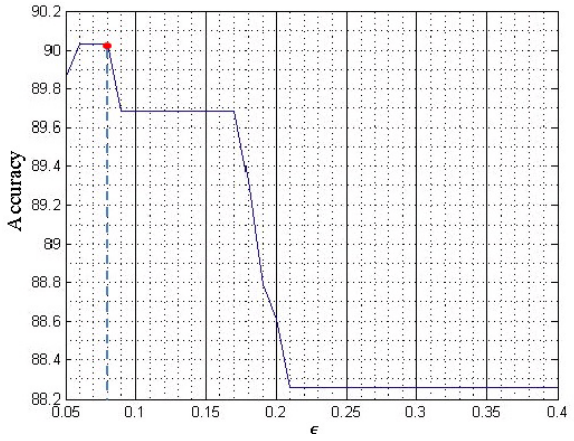
Example set	**CV**	**Avg. Accuracy**	**Avg. Tree size**
BS	1	92.5	19.7
	2	92.7	25.5
BUPA	1	69.3	15.3
	2	67.2	15

the tree. In order to obtain a fair comparison of results between the two-stage ordinary CV and the nested CV procedures, the two 10-fold CVs used in step 2.b are used here. For each training fold in the outer CV, a tree is trained on an ϵ value which is chosen using another 10-fold CV (inner CV) performed on that training fold. That is, the outer CV is used to estimate the accuracy of the tree while inner CV is used to choose ϵ_{NST} for each training set in the outer CV. For the first repetition of the nested CV, the variation of the accuracy as ϵ changes is given in Figure 5.12 and Figure 5.13 for the each training set of BS and BUPA respectively. Here also, the red dot and the blue dash-line have the same meaning as in Figure 5.11.

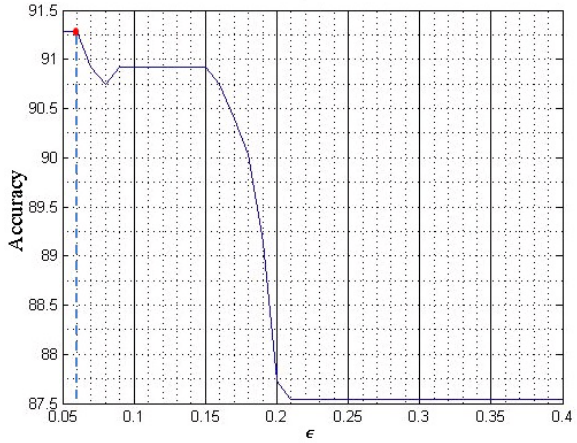
It is shown in Figure 5.12, for BS, ϵ_{NST} values vary between 0.06 (see Figure 5.12c) and 0.18 (see Figure 5.12h). Furthermore, ϵ_{TSOCV} found in the two-stage ordinary CV is 0.16 which is in the range of ϵ under the nested CV. Also, in the nested CV, at each optimal point, the estimated accuracy stays around 90% which is almost the same in two-stage ordinary CV. For the BUPA example set, ϵ_{NST} fluctuates between 0.25 (see Figure 5.13i) and 0.34 (see Figure 5.13g). The optimal value, ϵ_{TSOCV}, for two-stage ordinary CV is 0.3 which falls in this region. Moreover, the accuracies of each optimal point vary between approximately 66% - 72% which includes the accuracy at ϵ_{TSOCV}. Hence, this information confirms that ϵ_{NST} is consistent with ϵ_{TSOCV} and is subjected to



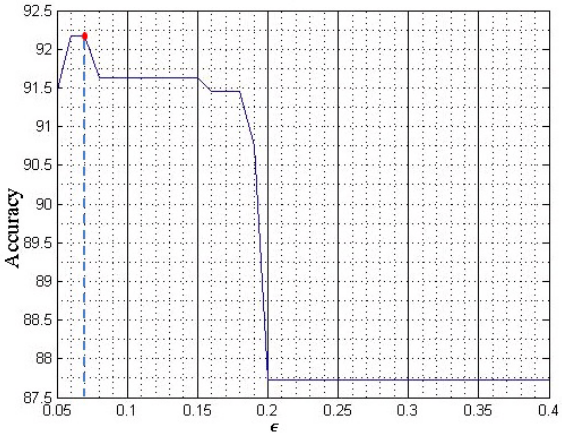
(a) Fold-1 results.



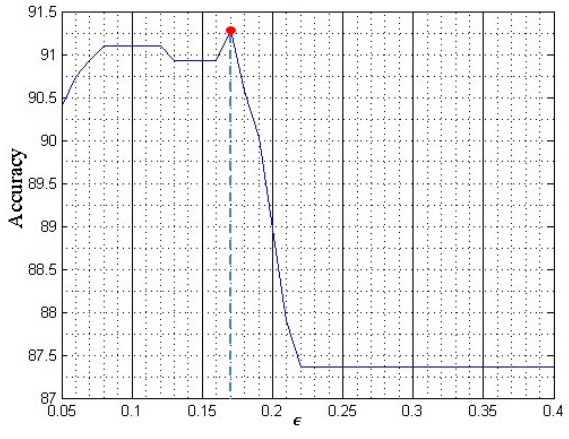
(b) Fold-2 results.



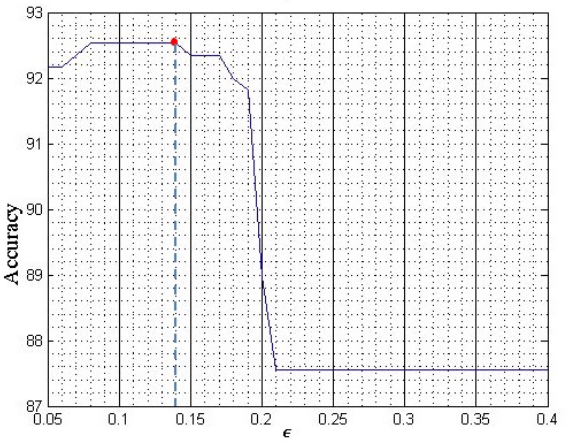
(c) Fold-3 results.



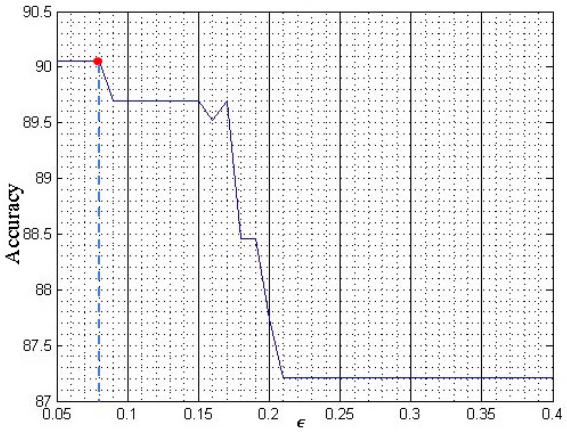
(d) Fold-4 results.



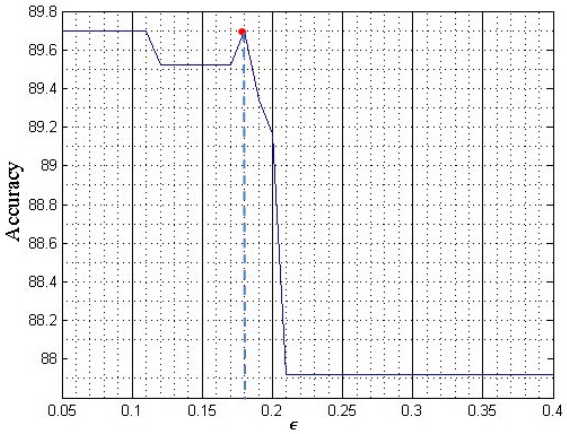
(e) Fold-5 results.



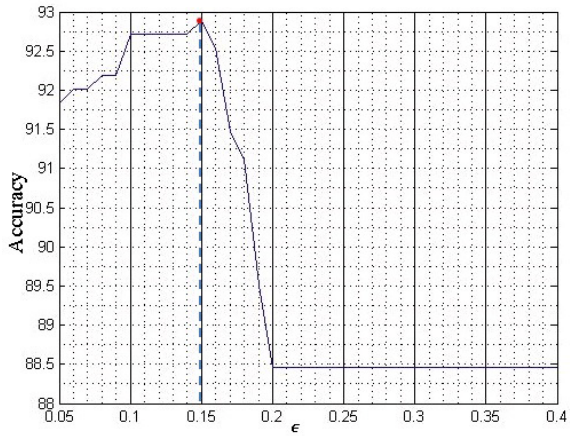
(f) Fold-6 results.



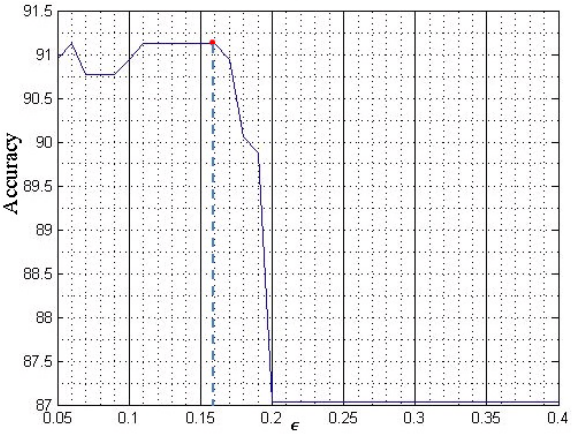
(g) Fold-7 results.



(h) Fold-8 results.

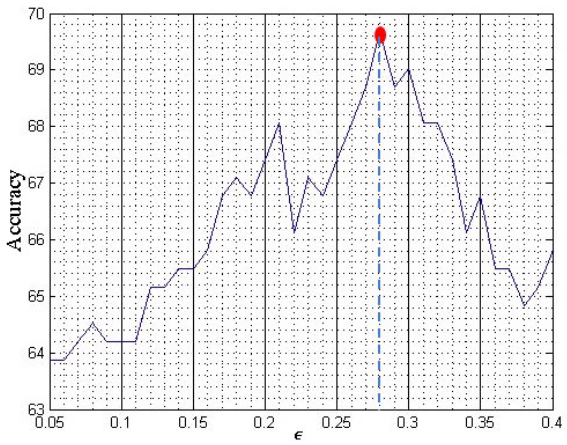


(i) Fold-9 results.

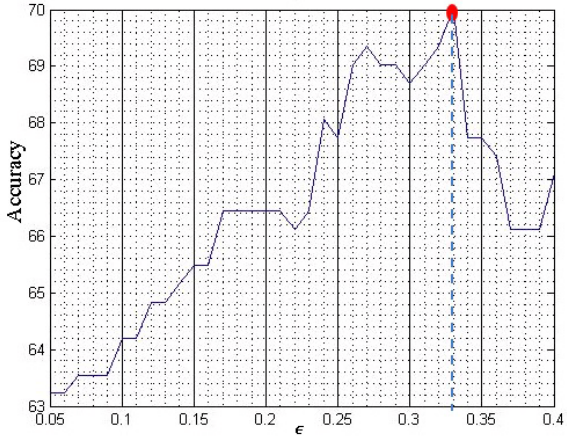


(j) Fold-10 results.

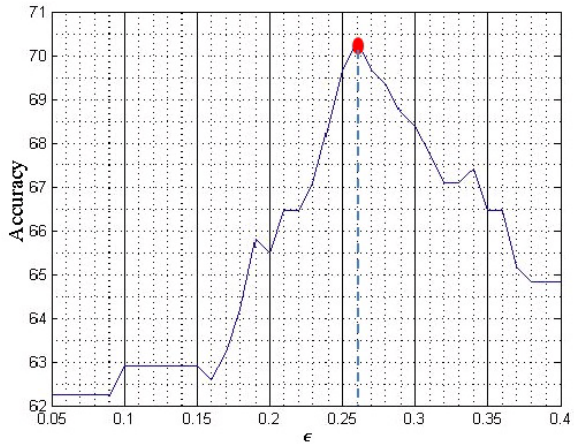
Figure 5.12: Fluctuation of accuracy with ϵ in each CV fold for BS.



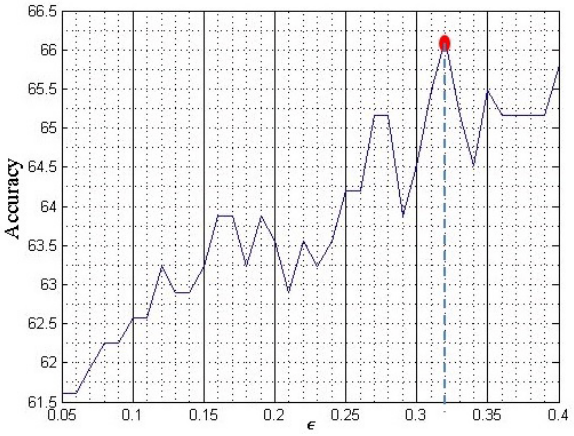
(a) Fold-1 results.



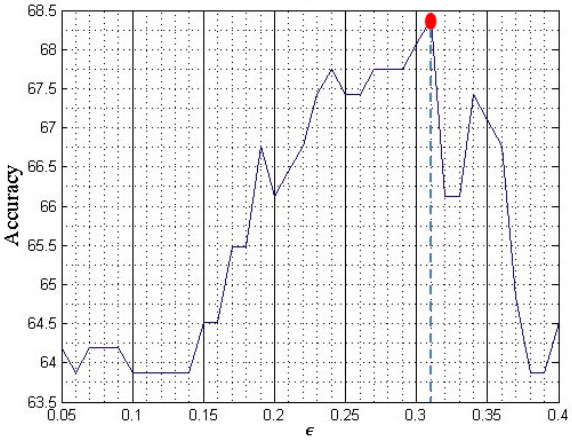
(b) Fold-2 results.



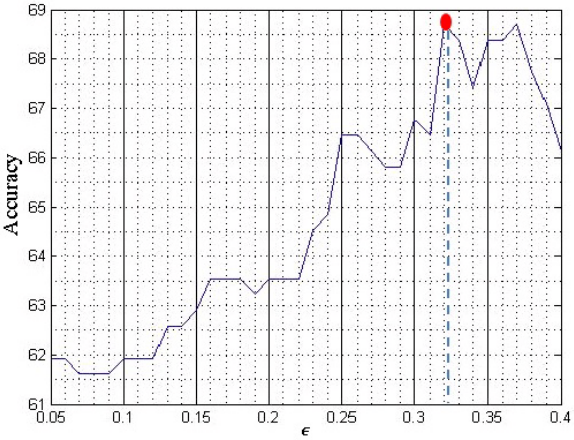
(c) Fold-3 results.



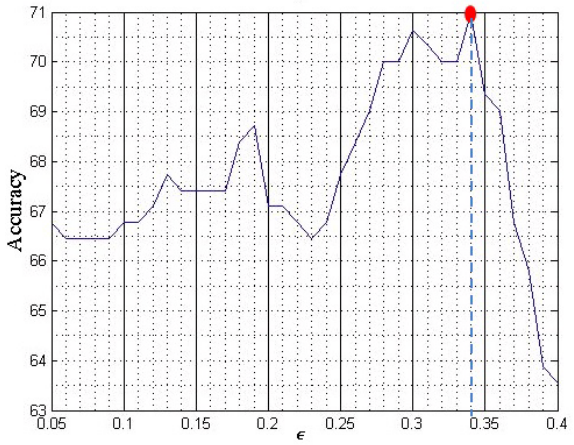
(d) Fold-4 results.



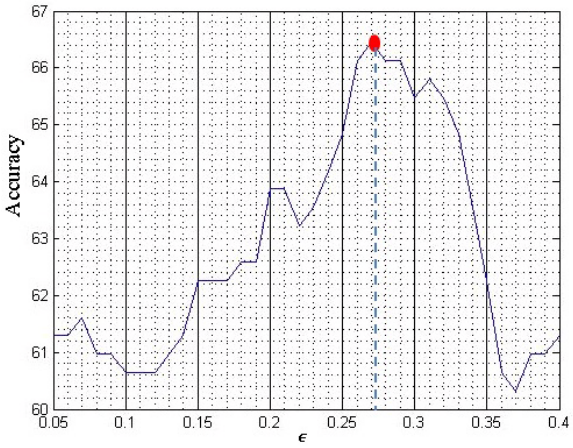
(e) Fold-5 results.



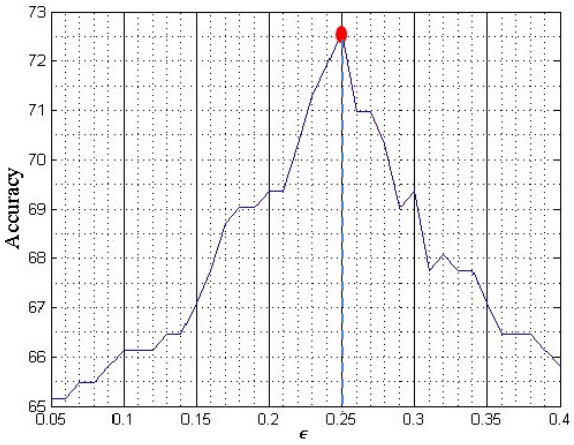
(f) Fold-6 results.



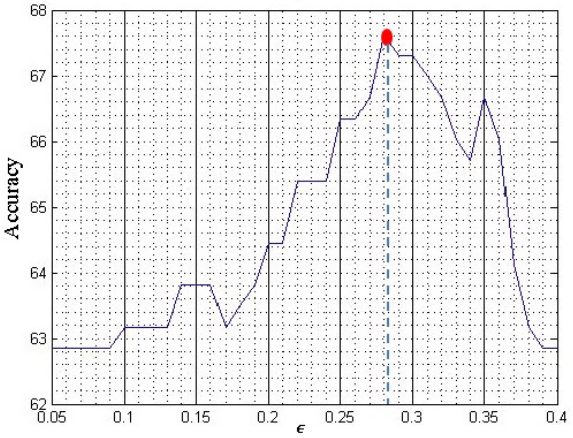
(g) Fold-7 results.



(h) Fold-8 results.



(i) Fold-9 results.



(j) Fold-10 results.

Figure 5.13: Fluctuation of accuracy with ϵ in each CV fold for BUPA.

a greater variability. However, the average accuracy at all optimal points approximately equals to the accuracy at the optimal point of two-stage ordinary CV.

[4] The chosen ϵ, ϵ_{NST}, is used to build a tree using the training set of the outer CV. In two-stage ordinary CV, there is one ϵ_{TSOCV} involved for each training set whereas in nested CV a separate ϵ_{NST} is used for each training set. Since, the same CV partitions are used to estimate the tree accuracy in the two-stage ordinary CV and the nested CV procedure, a particular tree is trained and tested on the same training and test set respectively. A fair comparison of the results is thereby obtained. The accuracy for each test fold is then compared for both CV methods and is given in Table 5.5 - 5.8.

Table 5.5: Results of the first repetition of CV for BUPA.

	Nested CV			**Ordinary CV:** $\epsilon_{TSOCV} = 0.3$	
Test Fold	ϵ_{NST}	**Accuracy**	**Tree Size**	**Accuracy**	**Tree Size**
1	0.28	62.9	23	65.7	12
2	0.33	68.6	4	68.6	12
3	0.26	68.6	28	71.4	19
4	0.32	68.6	9	68.6	20
5	0.31	68.6	12	74.3	17
6	0.37	65.7	8	65.7	27
7	0.34	74.3	13	74.3	13
8	0.27	74.3	22	74.3	15
9	0.25	62.6	35	57.1	2
10	0.28	73.3	20	73.3	16
Mean	0.3	68.8	17.4	69.3	15.3
SD	0.04	4.3	9.8	5.5	6.5

First, consider the BUPA example set. Looking at the ϵ_{NST} values, in Table 5.5 and Table 5.6, it is evident that there is a considerable impact on the tree size due to the variability in ϵ_{NST}. The reason for the variation in ϵ_{NST} is each training

Table 5.6: Results of the second repetition of CV for BUPA.

		Nested CV		**Ordinary CV:** $\epsilon_{TSOCV} = 0.3$	
Test Fold	ϵ_{NST}	**Accuracy**	**Tree Size**	**Accuracy**	**Tree Size**
1	0.26	68.6	20	68.6	18
2	0.31	80.0	16	80.0	16
3	0.3	62.9	17	54.3	7
4	0.28	74.3	22	68.6	13
5	0.31	60.0	9	60.0	9
6	0.22	80.0	46	62.9	13
7	0.31	77.1	19	77.1	19
8	0.25	62.9	34	65.7	22
9	0.37	60.0	9	68.6	13
10	0.24	60.3	42	66.6	20
Mean	0.3	68.6	23.4	67.2	15.0
SD	0.04	8.5	13.0	7.5	4.9

Table 5.7: Results of the first repetition of CV for BS.

		Nested CV		**Ordinary CV:** $\epsilon_{TSOCV} = 0.16$	
Fold	ϵ_{NST}	**Accuracy**	**Tree Size**	**Accuracy**	**Tree Size**
1	0.18	96.8	20	96.8	20
2	0.14	88.9	24	88.9	23
3	0.19	90.5	6	90.5	6
4	0.17	93.7	14	93.7	14
5	0.18	96.8	19	96.6	19
6	0.17	93.7	20	93.7	20
7	0.15	88.9	29	88.9	29
8	0.19	88.9	14	88.9	14
9	0.11	90.5	25	90.5	23
10	0.17	96.6	19	96.5	29
Mean	0.16	92.5	19.0	92.5	19.7
SD	0.02	3.4	6.5	3.3	7.1

Table 5.8: Results of the second repetition of CV for BS.

		Nested CV		Ordinary CV: $\epsilon_{TSOCV} = 0.16$	
Fold	ϵ_{NST}	**Accuracy**	**Tree Size**	**Accuracy**	**Tree Size**
1	0.15	95.2	27	95.2	27
2	0.16	90.5	5	90.5	5
3	0.07	95.2	37	95.2	30
4	0.17	93.7	18	93.7	18
5	0.15	92.1	37	92.1	37
6	0.16	93.7	17	93.7	17
7	0.18	88.9	43	88.9	43
8	0.16	96.8	22	96.8	22
9	0.18	87.3	23	87.3	23
10	0.16	93.1	33	93.1	33
Mean	0.15	92.7	26.2	92.7	25.5
SD	0.03	3.0	11.5	3.0	11.0

partition is different from each other. On the other hand, an inherent feature of tree classifiers is that they are very sensitive to data perturbation (Hand, Mannila, & Smyth, 2001). Therefore, different combinations of partitions of the same example set can produce heterogeneous trees and thus lead to a different value of ϵ_{NST} for each training partition.

Furthermore, Table 5.5 and Table 5.6 show the accuracies of each test set for the two repetitions of CV. In Table 5.5, for test folds 1, 3 and 5, two-stage ordinary CV produces the higher accuracies while for test fold 9, the accuracy is higher for the nested CV. For all other folds, the accuracies remain the same. Table 5.6 shows that the test folds 3, 4 and 6 produce higher accuracies for nested CV while 8, 9 and 10 produce higher accuracies for two-stage ordinary CV. For the other test folds the results remain the same. Therefore, on average, none of the procedures outperform each other. Moreover, Table 5.5 and Table 5.6 show that the mean accuracies and the standard deviations are similar in the two CV procedures.

When considering the results of the BS example set, a relatively easy classification

problem, Table 5.7 and Table 5.8 show that the ϵ_{NST} values are very similar to ϵ_{TSOCV}. Hence, the accuracy and the tree size of nested CV for each fold in both repetitions are almost equal to that of the two-stage ordinary CV. This shows that if the classes are separable, the optimal ϵ is similar in both CV procedures and hence, the final classification accuracies are similar.

As mentioned in Section 5.2.2, it is believed that the two-stage ordinary CV procedure may have higher accuracy than the nested CV procedure. However, fold by fold observation reveals that there is no apparent difference in accuracies in both procedures. That is, trees trained using ϵ_{NST} perform more or less the same as the trees trained using ϵ_{TSOCV}. Therefore, it can be concluded that the effect of ϵ_{NST}, which is independent of the test set, in determining the accuracy on average is the same as that of ϵ_{TSOCV}. Hence, the estimate of ϵ_{TSOCV} obtained from the first stage CV is almost independent of the second stage CV partitions even though the same example set is used in both occasions.

5.2.3 Remarks

The following observations are also made in the experiment.

[1] In Table 5.5, under two-stage ordinary CV, the estimated accuracy of the tree tested on the 9^{th} test fold is 57%. However, that tree has only 2 terminal nodes. The same can be seen under the nested CV where the tree tested on the 2^{nd} test fold gives 68.6% accuracy with the tree size of 4. When compared to the other tree sizes and the accuracies in the table, these two observations reveal that most of the data can be correctly classified in the upper nodes and the rest of the nodes actually classify the hard-to-classify examples. This observation is also made by (Manwani & Sastry, 2012).

[2] Furthermore, the nested CV procedure takes considerably more time than that

of two-stage ordinary CV to estimate the optimal ϵ. In the ordinary CV, initially a separate CV is run varying ϵ from 0.05 to 0.4 with the step size of 0.01, which requires 36 trees to be built to estimate ϵ_{TSOCV}. However, to estimate ϵ_{NST}, each training set requires 36 tree to be built. Hence, when estimating the accuracy, a one repetition of 10-fold CV requires only 36 trees under two-stage ordinary CV, while the nested CV requires a total of 360 trees to be built.

In summary, the nested cross validation procedure is a logically correct procedure to apply in experiments where the parameter estimation is done simultaneously with the estimation of accuracy. However, when considering accuracies given in Table 5.5 - 5.8, it is evident that the accuracies of both procedures are similar. In the two-stage CV, first a separate CV is run to estimate ϵ_{TSOCV}. The example set is then divided into v-folds ($v = 10$), and for each v, $v - 1$ partitions are used to induce a classifier with the optimal node level misclassification of ϵ_{TSOCV}. The accuracy is then estimated on the test set, the v^{th} fold. It is important to note that, the v^{th} fold does not specifically act on choosing ϵ_{TSOCV}. Therefore, it is reasonable to assume that the test set is independent of ϵ_{TSOCV} in the two stage ordinary CV. On the other hand, theoretically there should be at least some optimistic bias. However, it cannot be detected because of the variation in the accuracies. Two-stage CV can be applied to estimate the accuracy of the classifier although nested CV is the ideal method.

5.3 Use of Class Representative Vectors (CRVs)

Here, another possible set of vectors which can be used as alternative vectors for the Householder reflection is introduced. In the principal method of HHCART, the eigenvectors of classes' covariance matrices are used to represent the class orientation. In this section, another vector which represents the orientation of a class is defined,

called *Class Representative Vector* (CRV). CRV for a set of examples belongs to one class (say Class A) is derived as follows.

Let $\mathrm{X}_{n\times p}$ be a data matrix containing points of class A. Assume the centre of class A is the origin and each example has a unit length. That is, $\mathrm{X}_{n\times p}$ contains mean corrected examples with a unit length. The aim is to find a vector $\mathbf{v}$ such that the sum of the squared perpendicular distances from the data points of A to the line $\lambda\mathbf{v}, \lambda \in \mathbb{R}$ is minimised.

Let $\mathbf{x}_i$ be the vector representing the i^{th} example. The perpendicular distance d_i from $\mathbf{x_i}$ to the line $\lambda\mathbf{v}, \lambda \in \mathbb{R}$ can be obtained as follows.

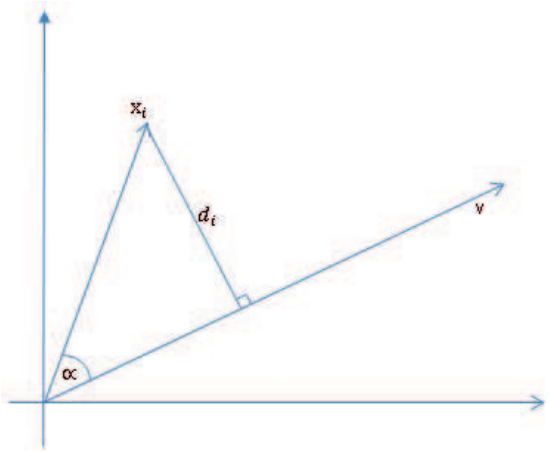


Figure 5.14: Geometrical view of the proof.

$$
\begin{aligned}
d_i^2 &= \|\mathbf{x_i}\|^2 \sin^2\alpha \text{ (see Figure 5.14)} \\
\cos\alpha &= \frac{\mathbf{v}\cdot\mathbf{x_i}}{\|\mathbf{v}\|\,\|\mathbf{x_i}\|} \\
d_i^2 &= \|\mathbf{x_i}\|^2 \left(1 - \frac{(\mathbf{v}\cdot\mathbf{x_i})^2}{\|\mathbf{v}\|^2\,\|\mathbf{x_i}\|^2}\right) \text{ since } 1-\cos^2\alpha = \sin^2\alpha \\
d_i^2 &= \frac{\|\mathbf{v}\|^2\,\|\mathbf{x_i}\|^2 - (\mathbf{v}\cdot\mathbf{x_i})^2}{\|\mathbf{v}\|^2}
\end{aligned}
\tag{5.3.1}
$$

Setting $\|\mathbf{v}\|^2 = 1$, the total squared perpendicular distance D is given by $D = \sum_{i=1}^{n}(1 - (\mathbf{v} \cdot \mathbf{x}_i)^2) = n - \sum_{i=1}^{n}(\mathbf{v} \cdot \mathbf{x}_i)^2 = n - \mathbf{v}^T \mathrm{X}^T \mathrm{X}\mathbf{v}$. Minimising D is equivalent to finding the $\mathbf{v}^*$ that maximises $\mathbf{v}^T \mathrm{X}^T \mathrm{X}\mathbf{v}$ subject to $\|\mathbf{v}^*\| = 1$. Noting that X is rank p, then $\mathrm{X}^T\mathrm{X}$ is positive definite and so $\mathbf{v}^T \mathrm{X}^T \mathrm{X}\mathbf{v}$ is maximised if $\mathbf{v}^*$ is the dominant eigenvector of $\mathrm{X}^T\mathrm{X}$.

Then Householder reflection, defined in equation (3.3.1), is used to make $\mathbf{v}^*$ parallel to one of the feature axes so that the axis-parallel splits can be searched in the reflected space. This algorithm is called HHCRV. The split finding method of HHCRV works as follows. At each non-terminal node t, HHCRV finds $\mathbf{v}^*$ for each class of examples. Then for each $\mathbf{v}^*$, a Householder matrix is constructed and the set of examples available at node t is reflected using each Householder matrix. axis-parallel splits are then searched in each reflected space and the best split found is chosen to split the node. The algorithm is recursively applied on all child nodes until each child node satisfies either:

[1] The misclassification rate at the child node is not greater than a user specified threshold (MisRate); or

[2] The number of examples in the node is less than or equal to a user specified threshold (MinParent).

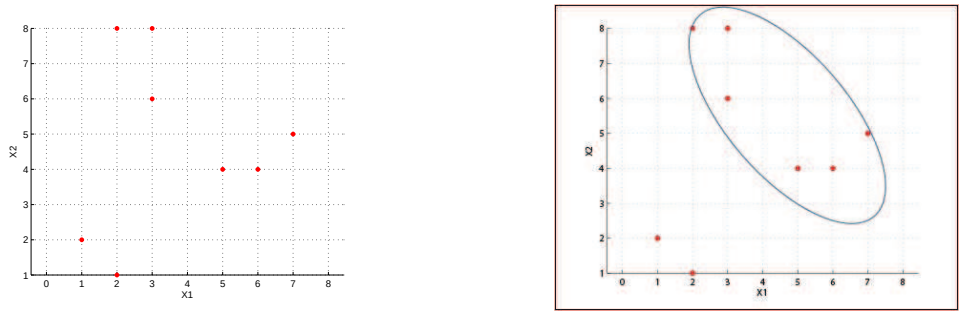
An overview of HHCRV algorithm at a non-terminal node t is given in Algorithm 9.

The CRV may be beneficial in some situations where the dominant eigenvector fails to capture the orientation of its class. It is tempting to assume that the orientation of a class is defined by the orientation of the majority of the points in the class. If the class contains some extreme values or outliers, then the orientation defined by the majority points may be distorted and hence, the dominant eigenvector may fail to capture the class orientation. This is illustrated in Figure 5.15.

The superimposed ellipse shows the orientation of the majority of examples.

Data: Input: Examples at node t, $\mathfrak{D}_t$.
initialization;
Define C_t = number of classes at node t;
$\Delta(I_{max}) = 0$;
$h_t = empty$;
for *i=1:*C_t **do**
  Extract the examples that belong to the i^{th} class in $\mathfrak{D}_t$, called D_i;
  Transform D_i such that its centre becomes the origin and divide the each row of D_i by its own norm: call the new dataset as D_i^{cs};
  $\mathbf{v}^*$ = eigenvector corresponding to the maximum eigenvalue of $(\mathrm{D}_i^{cs})^T \mathrm{D}_i^{cs}$;
  Call Algorithm 8 with $\mathbf{d} = \mathbf{v}^*$, $MinParent = 2$, $MisRate = 0$ and $\tau = 0$;
  Let h_{t_i} be the hyperplane returned by Algorithm 8 ;
  if *impurity reduction of* $h_{t_i} > \Delta(I_{max})$ **then**
    Replace h_t with h_{t_i}, the best hyperplane found so far;
    Replace $\Delta(I_{max})$ with the impurity reduction of h_{t_i}
  end
end

Algorithm 9: Overview of HHCRV algorithm at a single node.



(a) Scatter plot of the set of examples. (b) Scatter plot super imposed with an ellipse.

Figure 5.15: Effect of extreme values on the orientation of data.

Therefore, what would be anticipated is that the dominant eigenvector would be in the most stretched direction or otherwise known as the major axis of the ellipse. However, because of the two extreme values, the most stretched direction of the majority points is deviated towards to the least stretched direction of the majority points and is shown in Figure 5.16. The blue line shows the dominant eigenvector of the covariance matrix which is not aligned with the orientation of the majority points. The CRV found for the above example is shown by the blue line in Figure 5.17 which

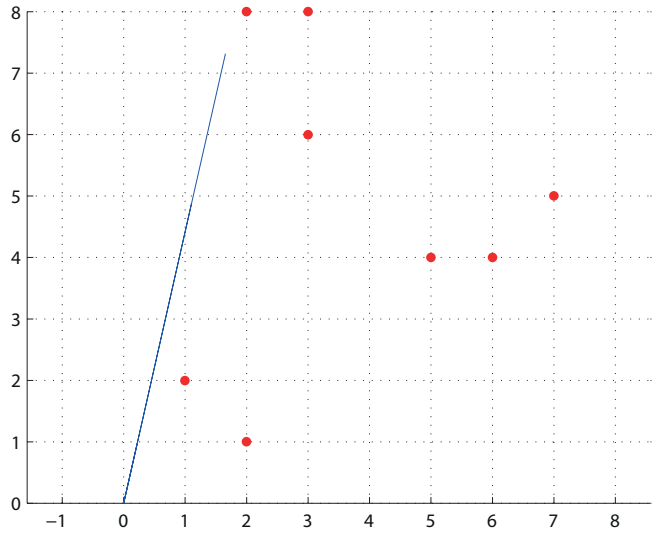


Figure 5.16: Scatter plot and the dominant eigenvector.

captures the orientation of the class properly.

5.4 Experiments on real life data sets

In this section, the empirical results are presented to show the performance of HHCRV and they are compared to the results of OC1 and OC1-LC. For OC1 and OC1-LC, the

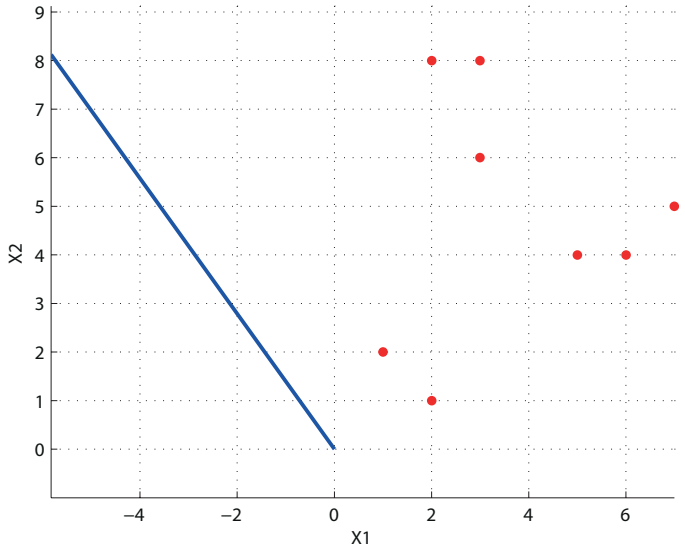


Figure 5.17: Scatter plot and the CRV.

minimum number of observations to split a node was set to 2. For OC1, the number of restarts and number of jumps were set to 20 and 5 (default values), respectively. For the HHCRV algorithm, MinParent, MisRate and τ was set to 2, 0 and 0 respectively. All algorithms used the Twoing rule as the measure of impurity (Breiman et al., 1984) and cost complexity pruning (Breiman et al., 1984) with 0-SE[2]. For all algorithms, ten, 5-fold ordinary CV procedures were used. For each fold, 10% of the training set was used exclusively for pruning. Results are reported in Table 5.9 along with respective standard deviations.

Results of Table 5.9 show that HHCRV's accuracies and tree sizes are comparable with OC1 and OC1-LC except for the BS and LET datasets. Eight example sets have more than 8 features and can be considered as higher dimensional classification problems. Of those eight sets, the performance of HHCRV is comparable with the

[2]see Section 1.7.1

Table 5.9: Results of HHCRV, OC1 and OC1-LC methods.

Dataset	DT	Avg. Acc.	Avg. Size	Dataset	DT	Avg. Acc.	Avg. Size
BS	HHCRV	87.0 ± 1.8	12.8 ± 4.1	PIND	HHCRV	72.8 ± 1.4	12.2 ± 7.5
	OC1	92.2 ± 0.6	9.34 ± 2.5		OC1	72.8 ± 1.4	9.5 ± 4.0
	OC1-LC	85.5 ± 1.7	11.0 ± 6.3		OC1-LC	73.2 ± 1.27	10.24 ± 5.4
BH	HHCRV	82.1 ± 0.9	9.3 ± 3.1	WINE	HHCRV	89.8 ± 3.0	4.2 ± 0.7
	OC1	82.4 ± 1.0	9.1 ± 3.4		OC1	89.9 ± 2.0	4.3 ± 0.2
	OC1-LC	82.1 ± 2.1	9.7 ± 3.3		OC1-LC	90.2 ± 2.2	4.1 ± 0.3
BC	HHCRV	96.4 ± 0.3	2.8 ± 0.8	SUR	HHCRV	72.2 ± 1.4	5.7 ± 4.7
	OC1	95.5 ± 0.7	3.4 ± 1.4		HHGDT	71.4 ± 2.0	7 ± 1.8
	OC1-LC	95.5 ± 0.7	4.3 ± 1.1		OC1-LC	70.7 ± 1.5	7.2 ± 4.1
BUPA	HHCRV	63.2 ± 3.0	9.9 ± 4.6	HRT	HHCRV	75.1 ± 2.7	6.6 ± 2.8
	OC1	66.5 ± 1.9	7.8 ± 4.4		OC1	76.7 ± 2.4	4.37 ± 0.8
	OC1-LC	64.9 ± 2.2	8.4 ± 3.7		OC1-LC	75.5 ± 2.4	5.6 ± 1.3
GLS	HHCRV	65.4 ± 1.6	11.1 ± 2.9	LET	HHCRV	83.3 ± 0.3	1250.3 ± 115.5
	OC1	64.7 ± 2.2	13.2 ± 2.5		OC1	84.3 ± 0.3	1318.6 ± 93.9
	OC1-LC	69.0 ± 3.0	12.2 ± 4.80		OC1-LC	85.39 ± 0.4	1497 ± 71
BNK	HHCRV	98.9 ± 0.1	4.3 ± 0.1	CLI	HHCRV	92.0 ± 0.8	3.1 ± 0.7
	OC1	98.9 ± 0.3	7.1 ± 1.3		OC1	91.5 ± 0.9	3.1 ± 0.9
	OC1-LC	98.5 ± 1.0	7.3 ± 1.1		OC1-LC	92.9 ± 0.7	4.1 ± 1.5
SEED	HHCRV	90.5 ± 2.3	3.7 ± 0.4				
	OC1	92.8 ± 1.8	3.6 ± 0.6				
	OC1-LC	88.4 ± 1.1	3.8 ± 0.7				

other benchmark methods. The maximum time complexity of HHCRV at a non-terminal node is $O(Cp(p + n \log n))$ while for OC1[3] it is $O(n^2 p \log n)$. Therefore, for larger n, HHCRV is a good alternative to the OC1 and OC1-LC algorithms. Moreover, the number of impurity function evaluations of OC1 at each non-terminal node is $20\beta\gamma np$ (see Section 2.2) whereas HHCRV evaluates the impurity function only Cnp times.

5.5 Conclusions and discussion

This chapter presents a set of alternative vectors that can be used to define the Householder reflection to improve classification results. In the first case, we show that the

[3]This is valid only if Max minority or Sum minority impurity function is used. For other functions, obtaining an upper bound is an open question (Murthy & Salzberg, 1995b).

Householder reflection based on the normal vector of the angular bisector of clustering hyperplanes, defined in Manwani and Sastry (2012), improves or shows no difference from the classification results of GDT for most of the datasets. Furthermore, the tree sizes of HHGDT are significantly smaller than that of GDT for most of the problems and hence, based on the empirical results, it can be concluded that HHGDT performs better than GDT because simpler classification rules are obtained without affecting accuracy. These improvements are basically due to two reasons and are as follows:

[1] In the GDT algorithm, the splitting hyperplane defined by the angular bisector of two clustering hyperplanes is a specific hyperplane defined by the normal vector of the angular bisector. In the HHGDT algorithm, an axis-parallel search is made along the angular bisector (the first coordinate axis of the reflected space) to find the best split. Therefore, HHGDT evaluates a series of hyperplanes which are parallel to the separating hyperplane found by GDT and hence, HHGDT has higher chance of finding a better hyperplane than GDT along the said normal vector. This is especially beneficial in multi-class classification problems because GDT is specifically designed only for two-class problems. GDT converts a multi-class classification problem into two-class classification problem by forming two super-classes and then it finds a clustering hyperplane for each super-class. Clustering hyperplanes are intended to capture the dominant linear tendency of the class (Manwani & Sastry, 2012). However, the clustering hyperplane for the super-class which represents all the classes other than the one having the most number of examples may not capture an effective linear tendency. In fact it is difficult to define the dominant linear tendency for a super-class. Therefore, the angular bisector may not be effective in this situation. On the other hand, HHGDT searches splits along the reflected axes and more importantly, when it evaluates those splits it uses all the classes instead of using two super-classes.

[2] The HHGDT algorithm explores more space than the GDT algorithm. More specifically, HHGDT evaluates np splits at a node compared to just one solitary split of the GDT algorithm. Therefore, HHGDT has higher chance of finding better splits than GDT.

A new methodology is presented to find the clustering hyperplanes when the matrices are singular. It is shown in the Subsection 5.1.2 that the existing method fails to find the clustering hyperplanes when the null space of the denominator matrix is a subspace of the null space of the numerator matrix. The proposed method overcomes this problem by performing matrix operations in the range space of the denominator matrix. The significance of the method is that it can be used for any application which solves a generalised eigenvalue problem under ill conditioned matrices.

Effects of the nested CV and the ordinary CV on the final classification accuracy were thoroughly studied. The GDT algorithm, and hence HHGDT, needs the node level misclassification, ϵ, to be estimated prior to the tree building. In this situation, a theoretically suitable method of estimating ϵ is to use nested CV. However, it is observed that there is no significant difference between the two stage ordinary and nested CV procedures in terms of accuracy. In two-stage ordinary CV, the two CV procedures run on different partitions. Hence, test set examples which are used to estimate the accuracy (in the second stage CV) have no direct effect in deciding the optimum ϵ and thus, it is reasonable to assume the test set is independent of the optimal ϵ. Furthermore, because of the variability associated with accuracies, the study failed to identify the subtle optimistic bias which may arise when using two-fold ordinary CV. Therefore, using two-stage ordinary CV over nested CV is recommended as:

[1] There is no significant difference between the accuracies.

[2] The time taken to run the nested CV procedure is much longer than that of

two-stage ordinary CV.

In the second case, a new set of vectors, CRVs, is introduced to define the Householder reflection. The algorithm which uses CRVs for data classification is called HHCRV. The empirical results show that the classification accuracies of HHCRV are comparable with the other competitive methods. The time complexity of the HHCRV algorithm is also lower than those methods and hence, HHCRV can be regarded as a good alternative for those competitive methods on large example sets.

Chapter 6

HHBUT: HouseHolder Bottom-Up Tree

6.1 Introduction

As an alternative to the top-down tree building approach, the bottom-up approach has recently been explored to induce binary decision trees. This chapter presents the motivation of the bottom-up tree induction approach followed by the existing bottom-up strategies of tree building. The approach starts with identifying clusters in the data and finding splitting hyperplanes to separate these clusters. Some of the more recently proposed bottom-up approach use the Expectation-Maximisation (EM) algorithm for data clustering and support vector machines (SVM) for finding the separating hyperplanes between clusters. A brief introduction on model based clustering via the EM algorithm and SVM is presented. The new algorithm, HHBUT (HouseHolder Bottom-Up Tree), proposed in this chapter explores the possibility of replacing SVM with the split finding principle of the HHCART algorithm to find separating hyperplanes between clusters. Finally, the results of bottom-up trees based on SVM are compared to HHBUT. The effect of the proposed heuristic of the thesis on top-down and bottom-up approaches is investigated via the HHCART(A) and

HHBUT algorithms.

6.2 Bottom-Up tree induction approach

6.2.1 Motivation

First, a brief explanation of the top-down tree building approach is given prior to the introduction of the bottom-up method. The top-down approach starts with the root node, where the full set of examples reside, and then recursively partitions the feature space into disjoint sub-regions until each sub-region becomes homogeneous or near-homogeneous with respect to a particular class. A tree induced for a two-class two-dimensional classification problem using the top-down approach is given in Figure 6.1 and the corresponding partition sequence is given in Figure 6.2.

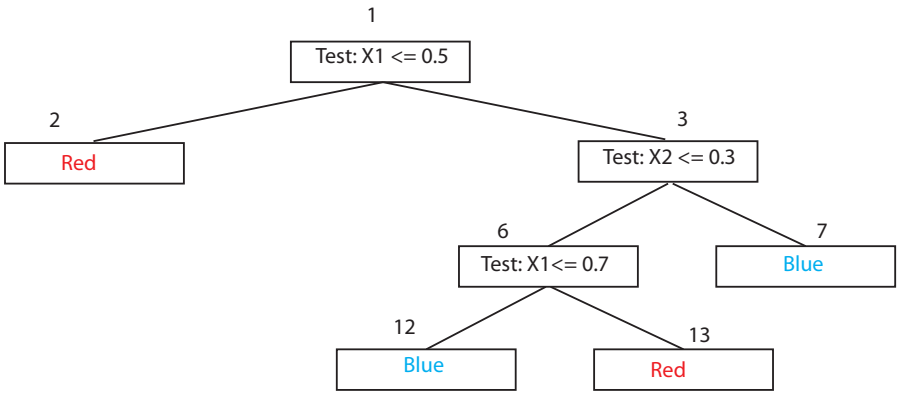


Figure 6.1: Basic structure of a classification tree.

Figure 6.2 shows the sub-division of the feature space at each non-terminal node in the tree given in Figure 6.1. Each sub-region is numbered by the corresponding node number. This sub-division of the feature space can be perceived as an attempt to identify possible clusters within each class of the example set. It is evident that

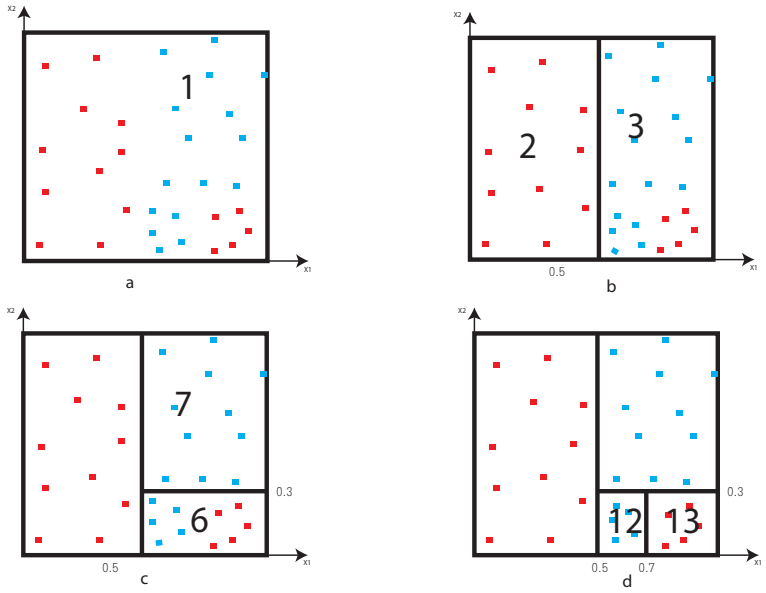


Figure 6.2: Feature space partition sequence.

according to the final partition structure (see Figure 6.2 (d)) each individual partition can be thought of as a cluster of that particular class. In this example, the spatial distribution of the classes can form more than one cluster per class. These individual partitions (or clusters) determine the terminal nodes of the tree. This motivated the bottom-up approach, which is explained in the next section.

6.2.2 Bottom-Up tree induction strategy

The first binary tree using the bottom-up approach is given in Landeweerd, Timmers, Gelsema, Bins, and Halie (1983). In this approach, the authors believe that each class forms one cluster in the feature space. Hence, initially a leaf or a terminal node is created for each class. Then for each pair of terminal nodes, the Mahalanobis distance is calculated. The pair having the smallest Mahalanobis distance is then merged to form a new class. This process is repeated until the root node is formed.

At each merging point, Fisher's linear discriminant analysis is used to determine the separating hyperplane of the two classes. The major drawback of this method is that it forces only one terminal node per class to be in the tree at the outset of the tree building. For example, a three class classification problem can only has three terminal nodes in the tree. This is quite restrictive as in practice, the distribution of a set of examples belonging to one class can form several clusters in the feature space (see Figure 6.2 for example).

A complete framework of the bottom-up tree building approach, Bottom-Up Oblique Decision-Tree Induction Framework (BUTIF), was introduced by Barros, Jaskowiak, Cerri, and de Carvalho (2014). This approach addresses the problem of having one cluster per class and uses a cluster analysis to identify the number of clusters within a given class. The steps of the BUTIF framework are given below:

[1] Divide the training data into pure subsets based on the class labels.

[2] Apply a clustering algorithm over each pure subset and identify clusters within each class. These clusters are considered as terminal nodes in the tree. Any clustering algorithm is possible, however, priority is given to the methods which are capable of automatically estimating the number of clusters from the data.

[3] For each cluster (terminal node), compute the cluster mean vector to determine the cluster centroid.

[4] Merge two nearest nodes, belonging to different classes, into a new class (meta-class). The authors suggest using Euclidean distance to measure the nearness between centroids. Once the new class is formed, the centroid is computed. Once the nodes are merged they are not considered for merging again.

[5] Use a feature selection algorithm for rule simplification. This is an optional task.

[6] Generate the separating hyperplane using any binary classifier to find the boundary between two classes being merged. BUITA uses the SVM algorithm with a linear-kernel for this purpose.

[7] Repeat the steps [4] to [6] until there are no classes to be merged. That is, this repetition occurs until the root node is attained.

One of the advantages of bottom-up induction method is that it always guarantees at least one terminal node per class. This characteristic is really appealing when handling classification problems, especially with unequal class sizes. In the top-down approach this characteristic is not guaranteed because smaller classes can get pruned out from the tree. This is especially so with unequal class sizes. Unless pre-pruning is used top-down trees tend to over-fit the training examples. Therefore, tree pruning is an instrumental procedure in top-down tree building as it helps to reduce over-fitting and thereby increases the prediction accuracy. However, in the bottom-up approach, pruning is not required since it does not over-fit the data. Consider the illustration given in Figure 6.3. The two clusters found, one for each class (Red and Blue), from the clustering method, are to be separated and the best split found is shown by the line. It is clear that some examples are misclassified in bottom-up approach. However, no further splitting is considered for examples. Therefore, over-fitting does not happen in the bottom-up approach and hence, pruning is not required.

6.3 Model based clustering

One of the main processes in the bottom-up approach is to identify clusters within each class. Various techniques have been proposed to identify clusters within data and basically these techniques can be categorised into: (a) hierarchical clustering, (b) optimisation clustering, and (c) model based clustering (Everitt, Landau, Leese, &

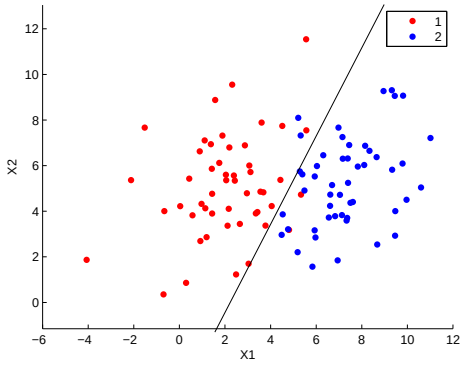


Figure 6.3: Separation hyperplane found by bottom-up approach.

Stahl, 2011). However, in this research the model based clustering approach is used to identify clusters within each class following the approach presented in Barros et al. (2014). The model based clustering approach attempts to fit a finite mixture of probability densities to the data. In this approach, often the family of the distributions are assumed but the parameters which specify the distributions are unknown. The number of probability densities found to be in the mixture model is then taken as the number of clusters found in the data. In practice, the common approach is to take the component densities to be univariate or multivariate Gaussian (Normal) (McLachlan & Peel, 2004). Moreover, Marron and Wand (1992) show that Gaussian mixtures are very flexible in approximating many arbitrarily shaped distributions. This is very important in the tree building context as the user does not know about the cluster structure of a particular class a priori. The following section introduces model based clustering and the parameter estimation procedure of Gaussian densities.

6.3.1 Finite Gaussian Mixture Model (FGMM)

Let $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ be a random sample of size n, where each $\mathbf{X_i}$ is a p-dimensional random vector with Gaussian probability density function of $\Phi_p(\mathbf{X}|\mu, \Sigma)$ on $\mathbb{R}^p$, where $\mu_{p\times 1}$ and $\Sigma_{p\times p}$ are the mean vector and the covariance matrix respectively. Let $\mathbb{D} = (\mathbf{X}_1^T, \mathbf{X}_2^T, \ldots, \mathbf{X}_n^T)^T$ represent the entire random sample and $\mathbb{d} = (\mathbf{x}_1^T, \mathbf{x}_2^T, \ldots, \mathbf{x}_n^T)^T$ is a realization of $\mathbb{D}$.

The K-component FGMM, f, is given by McLachlan and Peel (2004):

$$f(\mathbf{x_i}|\Psi) = \sum_{k=1}^{K} \pi_k \Phi(\mathbf{x_i}|\mu_k, \Sigma_k). \tag{6.3.1}$$

where π_i's are the mixing proportions and $\Psi = \{\pi_1, \ldots \pi_{K-1}, \mu_1 \ldots \mu_K, \}$ and the distinct elements of component covariance matrices $\Sigma_k, k = 1 \ldots K$. Equation (6.3.1) means that each observation $\mathbf{x_i}$ of $\mathbb{d}$ has been drawn from one of the Gaussian densities (known as components) designated by μ_k and Σ_k and selecting the k^{th} Gaussian component has the probability of π_k.

6.3.2 Maximum likelihood estimates of a Gaussian mixture model

Parameter estimates of a Gaussian mixture model are obtained via maximum likelihood estimation procedure. The likelihood function of FGMM given in equation (6.3.1) is given by:

$$l(\Psi|\mathbb{d}) = \prod_{i=1}^{n} \left\{ \sum_{k=1}^{K} \pi_k \Psi(\mathbf{x}_i|\mu_k, \Sigma_k) \right\}. \tag{6.3.2}$$

The log likelihood function is therefore, is given by:

$$\log l(\Psi|\mathbb{d}) = \sum_{i=1}^{n} \log \left\{ \sum_{k=1}^{K} \pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k) \right\}. \tag{6.3.3}$$

If the component label of each $\mathbf{x}_i$ had been observed, then estimation would be much easier. Specifically, if that is the situation, it is possible to extract all $\mathbf{x}_i$'s belonging to each component separately (subsets) and ML estimates can be obtained by applying the ML estimation procedure to each subset of $\mathbb{d}$. However, since the component labels have not been observed, the common approach of estimating the parameter vector, which maximises equation (6.3.3), is to use the Expectation-Maximisation (EM) algorithm (Dempster, Laird, & Rubin, 1977). In this particular situation, the EM algorithm assumes that each $\mathbf{X}_i \in \mathbb{D}$ is observed with its component label (known as latent variable) $\mathbf{Z}_i$ a K-dimensional binary random vector. The k^{th} element of $\mathbf{Z}_i$ is defined as:

$$Z_{ik} = \begin{cases} 1 & \text{if } \mathbf{X}_i \text{ belongs to the } k^{th} \text{ component} \\ 0 & \text{otherwise.} \end{cases}$$

Since, the random vector $\mathbf{X}_i$ belonging to the k^{th} component of the mixture model is given by $Z_{ik} = 1$, the marginal distribution of $\mathbf{Z}_i$ is given by:

$$p(Z_{ik} = 1) = \pi_k \text{ where } \sum_{k=1}^{K} \pi_k = 1. \tag{6.3.4}$$

Thus, $\mathbf{Z}_i$ is distributed according to a multinomial distribution consisting of one draw on K components with probabilities $\pi_1, \ldots, \pi_K$, that is:

$$p(\mathbf{z}_i) = \prod_{k=1}^{K} \pi_k^{z_{ik}}, \tag{6.3.5}$$

where $\mathbf{z}_i$ is a realisation of $\mathbf{Z}_i$.

Moreover, in the Gaussian mixture model, the conditional density of $\mathbf{X}_i$ given $\mathbf{Z}_i$ is Gaussian and is given by:

$$f_{\mathbf{X}_i|\mathbf{Z}_i}(\mathbf{x}_i|\mathbf{z}_i) = \Phi(\mathbf{x}_i|\mu_k, \Sigma_k). \tag{6.3.6}$$

Equation (6.3.6) can be written as:

$$f_{\mathbf{X}_i|\mathbf{Z}_i}(\mathbf{x}_i|\mathbf{z}_i) = \prod_{k=1}^{K} (\Phi(\mathbf{x}_i|\mu_k, \Sigma_k)^{z_{ik}} \tag{6.3.7}$$

Therefore, the joint density of $\mathbf{X}_i$ and $\mathbf{Z}_i$ is given by:

$$f_{\mathbf{X}_i\mathbf{Z}_i}(\mathbf{x}_i, \mathbf{z}_i|\Psi) = \prod_{k=1}^{K} (\pi_k(\Phi(\mathbf{x}_i|\mu_k, \Sigma_k))^{z_{ik}} . \tag{6.3.8}$$

Using the Bayes rule, for any given Ψ:

$$\begin{aligned} E(z_{ik}|\mathbf{x}_i, \Psi) &= p(z_{ik} = 1|\mathbf{x}_i, \Psi) \\ &= \frac{\pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k)} \\ &= \gamma_{ik}(\Psi). \end{aligned} \tag{6.3.9}$$

The quantity $\gamma_{ik}(\Psi)$ is the posterior probability of $z_{ik} = 1$ after observing $\mathbf{x}_i$ and is called *responsibility*.

Denote the set of all latent variables by matrix $\mathrm{Z}_{n\times K}$, where $\mathbf{z}_i^T$ corresponds to the i^{th} row of Z. For each observation vector $\mathbf{x}_i \in \mathbb{d}$, there is a particular row in Z and hence both $\mathbb{d}$ and Z together are called the *complete data* and the observed sample $\mathbb{d}$ alone is called the *incomplete data*. For example, the complete data set has the following structure:

$$\mathbb{d}^{\mathbb{C}} = \left[\begin{array}{c|c} x_{11}, x_{12} \dots x_{1p} & z_{11}, z_{12} \dots z_{1K} \\ \vdots & \vdots \\ x_{i1}, x_{i2} \dots x_{ip} & z_{i1}, z_{i2} \dots z_{iK} \\ \vdots & \vdots \\ x_{n1}, x_{n2} \dots x_{np} & z_{n1}, z_{n2} \dots z_{nK} \end{array}\right] \tag{6.3.10}$$

where each $x_{ij} \in \mathbb{R}$ and only one element of i^{th} row $(z_{i1}, z_{i2} \dots z_{iK})$ of Z equals one for $i = 1, \dots, n$.

According to equation (6.3.8), the complete data likelihood is therefore given by:

$$l(\Psi|\mathbb{d}^{\mathbb{C}}) = \prod_{i=1}^{n} \left\{ \prod_{k=1}^{K} (\pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k))^{z_{ik}} \right\}, \tag{6.3.11}$$

and the complete data log likelihood is given by:

$$\log l(\Psi|\mathbb{d}^{\mathbb{C}}) = \sum_{i=1}^{n} \left\{ \sum_{k=1}^{K} z_{ik} \log \left[\pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k) \right] \right\}. \tag{6.3.12}$$

The EM algorithm is then used to obtain the maximum likelihood estimates of the parameters in FGMM using the log likelihood function given in equation (6.3.12). It has two steps, namely: (a) Expectation step (E), and (b) Maximisation step (M).

The two steps of the EM algorithm are as follows:

1. E-Step

Compute $Q(\Psi, \Psi^{(j)})$ which the is expectation of the complete data log likelihood w.r.t. the conditional density of latent variables conditioned on incomplete data where the current value of Ψ is given by $\Psi^{(j)}$

$$\begin{aligned} Q(\Psi, \Psi^{(j)}) &= E(\log l(\Psi|\mathbb{d}^{\mathbb{C}})) \\ &= \sum_{i=1}^{n} \left\{ \sum_{k=1}^{K} E\left[z_{ik}|\mathbf{x_i}, \Psi^{(j)} \right] \log \left[\pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k) \right] \right\} \\ &= \sum_{i=1}^{n} \left\{ \sum_{k=1}^{K} \gamma_{ik}(\Psi^{(j)}) \log \left[\pi_k \Phi(\mathbf{x}_i|\mu_k, \Sigma_k) \right] \right\}. \end{aligned} \tag{6.3.13}$$

2. M-Step

Find $\Psi^{(j+1)}$ which maximises $Q(\Psi, \Psi^{(j)})$ over Ψ. That is:

$$\Psi^{(j+1)} \leftarrow \text{argmax}_{\Psi} \left[Q(\Psi, \Psi^{(j)}) \right]. \tag{6.3.14}$$

The $(j+1)^{th}$ value of the each element in the set Ψ is obtained by setting:

$$\frac{\partial Q(\Psi, \Psi^{(j)})}{\partial \psi_{\in \Psi}} = 0. \tag{6.3.15}$$

The EM algorithm iterates through the E-step and M-step until there is no significant change in estimates of Ψ from $\Psi^{(j)^{th}}$ step to $\Psi^{(j+1)^{th}}$ step.

Once the k-component Gaussian mixture model is fitted to the data, each example is assigned to one of the K Gaussian components based on the estimated posterior

probabilities given by $\hat{\gamma}(\hat{\Psi})$ where $\hat{\Psi}$ is the estimated parameter vector. $\hat{\gamma}(\hat{\Psi})$ is a $n \times K$ matrix where the element in the i^{th} row and j^{th} column gives the probability of the i^{th} example belongs to the j^{th} component of the mixture model. More specifically, the i^{th} row of $\hat{\gamma}(\hat{\Psi})$ is given by:

$$\hat{\gamma_i}(\hat{\Psi}) = \left\{\hat{\gamma_{i1}}(\hat{\Psi}), \hat{\gamma_{i2}}(\hat{\Psi}), \ldots, \hat{\gamma_{iK}}(\hat{\Psi})\right\}.$$

The corresponding Gaussian component $i^{(k^*)}$ for the i^{th} example is then found according to the criterion:

$$i^{(k^*)} = \text{argmax}_k \left\{\hat{\gamma_{i1}}(\hat{\Psi}), \hat{\gamma_{i2}}(\hat{\Psi}), \ldots, \hat{\gamma_{iK}}(\hat{\Psi})\right\}. \tag{6.3.16}$$

Based on the assigned component label, each observation is then grouped into clusters where each cluster comprises of observations having the same component label. These clusters are considered as the terminal nodes in the bottom-up approach.

6.3.3 Determining the number of clusters

In the finite mixture modelling context the number of mixture components, K, corresponds to the number of clusters. The number of clusters found in each class determines the number of terminal nodes belonging to that class in the tree. Usually, the number of clusters present in the example set is unknown. Various methods have been suggested to estimate K for a mixture model (Everitt et al., 2011), for example:

[1] Log likelihood ratio test statistics.

[2] Markov Chain Monte Carlo methods using reversible jump MCMC or birth and death process methodology.

[3] Information theoretic approaches.

[4] v-fold CV.

[5] Monte Carlo cross-validation[1].

All these methods test the hypothesis of $H_0 K = k_0$ *Vs* $H_1 K = k_1$ for some $k_1 > k_0$ and usually $k_1 = K_0 + 1$. The use of log likelihood ratio test fails, as the regularity conditions do not hold. Therefore, the distribution of the corresponding test statistic $(-2 \ln \lambda)$ does not follow the chi-square distribution (Everitt et al., 2011; McLachlan & Peel, 2004). Monte Carlo methods require extensive use of simulations to test the hypothesis (McLachlan & Peel, 2004). Therefore, they are not ideal choices to include into the bottom-up approach as it increases the induction time of the tree. Information theoretic based approaches to determine K are also popular in finite mixture modelling. In particularly, the BIC produces a consistent estimate of K when a normal mixture model is used (Roeder & Wasserman, 1997). (McLachlan & Peel, 2004) used a simulation study to show that Akaike Information Criterion (AIC) tends to overestimate the number of clusters .

The v-fold cross-validation method partitions the example set into v-folds and each time the likelihood of the test partition is evaluated on the model fitted on the remaining partitions. The steps of v-fold CV are given below:

[1] Set the number of components (clusters) to 1.

[2] The training set is divided randomly into v-folds.

[3] EM is performed over $v - 1$ sets to fit the mixture model and the remaining set is used to evaluate the log likelihood of the fitted model.

[4] The log likelihood is averaged over all v results.

[5] If the log likelihood is increased the number of clusters is increased by 1 and the procedure continues at step 2.

[1] this is introduced in Section 1.9

Smyth (1996, 2000) shows that v-fold CV is inferior to BIC under Gaussian mixtures and recommend the Monte Carlo cross-validation method. However, both v-fold CV and Monte Carlo CV are time consuming procedures. Thus, incorporating these procedures into the tree building algorithm causes the total induction time to increase. Hence, although there are many criteria/procedures are proposed to determine K, selecting the most suitable method in the context of tree building depends on various factors, which are listed below.

[1] It is important to select a method which takes less time to determine the number of clusters.

In the bottom-up approach, total tree induction time mainly depends on two factors: (a) the time taken by the clustering algorithm, and (b) the time taken to build the tree. The time spent by the clustering algorithm can be reduced by choosing a time efficient clustering method to determine the number of clusters.

[2] User intervention should be minimal during the tree induction process.

Determining the number of clusters is a part of the model selection which is usually done by statisticians and sometimes with the help from problem domain experts. However, non-statisticians or non-experts may not be familiar with these concepts and therefore, data mining tools which need expert user intervention in the middle of the process may inhibit their usefulness to others (Campos, Stengard, & Milenova, 2005). Hence, fully automated data mining tools, in this instance oblique trees, are desirable to cater for non-experts' requirements. Thus, clustering methods which rely on user intervention to determine the number of clusters are not ideal choices to use in tree building algorithms.

[3] It is reasonable enough to identify the number of clusters such that the induced tree will be accurate and compact. In the data clustering perspective, the main

aim is precise identification of clusters and the number of clusters. However, in the context of bottom-up decision tree induction, precise cluster identification is not that important as long as it does not have an adverse effect on classification accuracy and the tree size. Consider the scatter plot of hypothetical data given in Figure 6.4. The scatter plot shows that there are two clusters in the blue

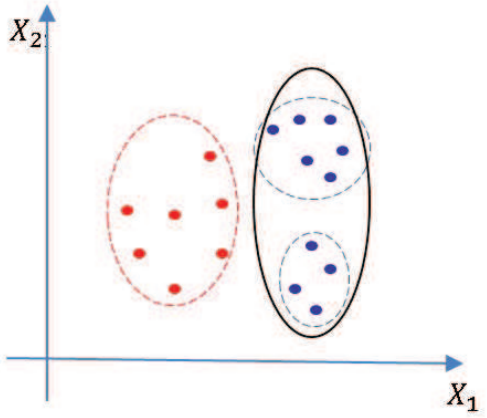


Figure 6.4: Clusters found and separation.

class and one cluster in the red class. A good clustering algorithm will easily explore this structure and a tree can be built upon the clusters found. However, failing to identify two blue clusters does not have any adverse effect of the final classification as the red class and the blue class can be easily separated. In this particular example, the former case produces a larger tree (three terminal nodes) while the latter case produces a compact tree (only two terminal nodes) without trading off the accuracy. Hence, the intuition is that a criterion which under-estimates the number of clusters would be preferred as long as the classification accuracy is not penalised. Abas (2013) refer to Yang and Zwolinski (2001) and state that BIC tends to under-estimate the number of clusters when the example set is small.

To the best of the author's knowledge, estimating the number of clusters in the context of decision tree induction has not been explored much. Barros et al. (2014) use two methods, namely: (a) 10-fold CV, and (b) The ordered multiple runs procedure to estimate the number of clusters. Moreover, the 10-fold CV is used as the criterion for the EM algorithm whereas the ordered multiple runs procedure was used as the criterion for the k-means algorithm. Barros et al. (2014) show that EM based trees produce a better classification accuracy than k-means based trees when support vector classifiers (given in Section 6.4) are used for split finding. However, Barros et al. (2014) empirically show that the results can be improved by applying a feature selection method at each non-terminal node before the splits are searched. In this research, two methods are used in the experiments to determine K, namely: (a) the BIC criterion, and (b) the CV procedure.

6.4 Support Vector Machine

A Support Vector Machine (SVM) (Vapnik (2000), Shawe-Taylor and Cristianini (2000), Hastie et al. (2009) and Boser, Guyon, and Vapnik (1992)) is a classifier which uses a margin maximisation technique to find the separating hyperplane for a two-class classification problem. The margin M is defined as the width between the decision boundary $f(x)$ and the closest examples from either classes. Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ be the training sample where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$. The classification rule given by SVM assigns $\mathbf{x}$ to class $+1$ if $f(\mathbf{x}) \geq 0$ and to class -1 otherwise.

Since, this thesis only considers linear classifiers, the discussion is limited to how SVM finds a linear hyperplane between classes. If the classes are linearly separable, then the SVM problem is known as a hard margin classification problem whereas when the classes are not linearly separable it is know as a soft margin classification problem. First, the SVM strategy of finding a separating hyperplane for a hard margin

classification problem is explained and then the soft margin strategy, the general case, is explained.

6.4.1 Hard margin classification problem

The hard margin problem is illustrated in Figure 6.5. SVM tries to find a function $f(\mathbf{x}) = \beta_0 + \mathbf{x}^T\beta$ by solving the optimisation problem given in equation (6.4.1):

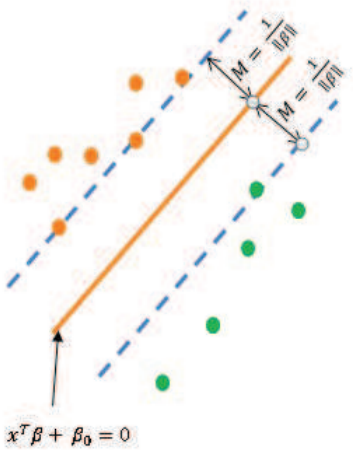


Figure 6.5: SVM in separable case.

$$\begin{aligned} &\max_{\beta_0,\beta} M \\ &\text{subject to } \frac{y_i f(\mathbf{x}_i)}{\|\beta\|} \geq M, i = 1, \ldots, n. \end{aligned} \tag{6.4.1}$$

Taking $M = 1/\|\beta\|$, equation (6.4.1) can be written as:

$$\begin{aligned} &\min_{\beta_0,\beta} \frac{1}{2}\|\beta\|^2 \\ &\text{subject to } y_i f(\mathbf{x}_i) \geq 1, i = 1, \ldots, n. \end{aligned} \tag{6.4.2}$$

The convex optimisation problem given in equation (6.4.2) is solved by using the Lagrange Multiplier technique (Boser et al., 1992; Hastie et al., 2009), where the

Lagrange primal function is given by:

$$L_p = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{n} \alpha_i[y_i f(\mathbf{x}_i) - 1] \text{ where } \alpha_i\text{'s are Lagrange multipliers.} \quad (6.4.3)$$

It can be shown that the examples whose $\alpha_i > 0$ define the separating hyperplane (Hastie et al., 2009) and are called support vectors.

6.4.2 Soft margin classification problem

The soft margin problem is illustrated in Figure 6.6. If the classes are linearly non-

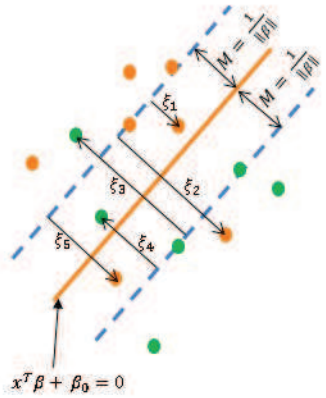


Figure 6.6: SVM in non-separable case.

separable, the optimisation problem given in equation (6.4.1) does not converge to a solution as $\forall \mathbf{x_i}$, $y_i f(\mathbf{x}_i) \geq M, i = 1, \ldots, n$ will not be satisfied. In order to relax the constraints of the optimisation problem, a slack variable is introduced for each example and the optimisation problem (6.4.1) now can be written as:

$$\begin{aligned} &\min_{\beta_0,\beta} \frac{1}{2}\|\beta\|^2 + \Gamma \sum_{i=1}^{n} \xi_i \\ &\text{subject to } y_i f(x_i) \geq 1 - \xi_i, \ \forall i \\ &\xi_i \geq 0 \end{aligned} \quad (6.4.4)$$

The Lagrange function corresponding to problem (6.4.4) is given by:

$$L_p = \frac{1}{2}\|\beta\|^2 + \Gamma\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i f(x_i) - (1-\xi_i)] - \sum_{i=1}^{n}\mu_i\xi_i, \tag{6.4.5}$$

where α_i's and μ_i are Lagrange multipliers.

Here too, the only examples that have $\alpha_i > 0$ contribute to the separating hyperplane $f(\mathbf{x})$. The constant Γ, cost parameter or box constraint, is a user defined parameter. Larger Γ values penalise the margin violations ($\sum_{i=1}^{n}\xi_i$) more and therefore, the margin is narrower. On the other hand, smaller Γ values allow some margin violations and hence, the margin is larger. Thus, the choice of Γ determines the width of the margin which in turn affects the accuracy of the classifier. Therefore, the value of Γ which maximises the accuracy has to be estimated before the classifier is constructed. Hastie et al. (2009) propose v-fold CV while A and Gopal (2010) use separate validation set to estimate the optimal value of Γ. Barros et al. (2014) use SVM at each non-terminal node to obtain the class separation hyperplane. However, Barros et al. (2014) have not mentioned how the parameter Γ is estimated.

Ideally, at each non-terminal node the parameter Γ should be tuned to obtain a locally optimal split. However, this consumes more time, especially for trees with a large number of non-terminal nodes. Hence, in the experiments, the 10-fold CV is used, prior to the tree building, to estimate the optimal value of Γ for the entire tree. This Γ value can be regarded as a common optimal value rather than a node specific optimal value.

6.5 The principle of HHCART in place of SVM

In the bottom-up induction approach, as given in Section 6.2.2, terminal nodes (clusters) are merged upwards until the root node is reached. Once the two terminal nodes are merged, a new class (meta-class) is formed and that meta-class is represented by

a non-terminal node in the tree. At each non-terminal node, a hyperplane is found to separate the examples coming to that node. Barros et al. (2014) use SVM to find the separating hyperplane. However, in the description of the bottom-up algorithm, Barros et al. (2014) state that any binary classifier can be used to find a separating hyperplane in non-terminal nodes. Therefore, in this research, the split finding methodology of the HHCART algorithm is used to find the separating hyperplane at each non-terminal node. Hence, at each non-terminal node, the proposed algorithm, HHBUT, first computes the covariance matrix of each class (or meta-class), and then finds all the eigenvector of these matrices. Recall that there are only two classes at each non-terminal node. The Householder matrices are then constructed using eigenvectors and the example set at the node is reflected using each matrix. axis-parallel splits are then searched in the reflected spaces and the best split found is used as the separating hyperplane at the non-terminal node. The overview of the proposed algorithm, HHBUT, is given in Algorithm 10.

Therefore, in this research, SVM is replaced by the split finding principle of HHCART to find the separating hyperplane between classes. Both BIC criteria and CV procedure are used to determine the number of clusters in a class. The results of the trees based on BIC and CV are compared and a suggestion is made as to which method is suitable to determine the number of clusters in the context of tree building.

6.6 Experiments on real life example sets

The main objectives of the experiments are:

[1] To find the most suitable method to estimate the number of clusters in the bottom-up tree building context.

Data: Input: Training example set $\mathfrak{D}$.
initialization;
C -Number of Classes;
Rm_Nodes - Number of remaining nodes;
Distance matrix, $\mathcal{M}$;
$h_t = empty$;
for *i=1:C* **do**
    Extract the examples that belong to the i^{th} class in $\mathfrak{D}$, called D_i;
    Apply a clustering algorithm on D_i to find clusters within D_i. Let Θ_i be the set of clusters found in the i^{th} class;
    Create a terminal node for each element in Θ_i and assign the corresponding class label to each terminal node;
    Compute the centre of each element in Θ_i;
end
Calculate the distance between each pair of terminal nodes (distance between two centres) which are not belonging to the same class and store into $\mathcal{M}$;
$Rm\_Nodes = \sum_{i=1}^{C} |\Theta_i|$;
if $Rm\_Nodes \neq 1$ **then**
    while $Rm\_Nodes > 1$ **do**
        Find the two nodes (t_1, t_2) which has the minimum distance (belonging to two separate classes) ▷ Distances are stored in $\mathcal{M}$;
        Merge t_1 and t_2 and make a new node t (non-terminal);
        Compute the centre of the new node and assign a new label;
        Update $\mathcal{M}$ ▷ compute distances between existing nodes and t;
        Compute the covariance matrices of examples belonging to t_1 and t_2 ;
        Perform eigenanalysis of each covariance matrix;
        For each eigenvector construct a Householder matrix;
        Transform examples belonging to t ;
        Perform axis-parallel splits in the reflected spaces;
        h_t = The best split found for the non-terminal node t;
        $Rm\_Nodes = Rm\_Nodes - 1$;
    end
end

Algorithm 10: Overview of the HHBUT algorithm.

[2] To compare the performances of SVM and HHBUT with respect to the classification accuracy.

An experiment is conducted using 5-fold CV. For each training fold, a finite mixture model is fitted to identify the number of clusters within each class. The number of clusters is estimated using a particular criterion, BIC or CV. Once the clusters are identified, a split finding algorithm, SVM or HHBUT, is applied to build a decision tree. Then the test fold is applied to estimate the accuracy. Ten repetitions of 5-fold CV are used to estimate the final classification accuracy of a given example set. Since there are two criteria to determine the number of clusters and two split finding algorithms, four trees are induced for each example set, namely: (a) SVM with BIC (b) SVM with CV (c) HHBUT with BIC, and (d) HHBUT with CV. Each tree is induced and tested on the same training partition and testing partition respectively and hence, a fair comparison is reached.

Two assumptions are made when fitting a finite mixture model to an example set and are given below:

[1] It is assumed that the each cluster has a Gaussian distribution.

[2] Heteroscedastic covariance matrices in the mixture model.

As discussed in Section 6.3.1, the author assumes that each component of the mixture model has a Gaussian distribution. Finite mixture models are fitted with various covariance structures. For example, diagonal covariance matrices are chosen assuming that there is no correlation between features. In some cases, the finite mixture model is fitted with a common covariance matrix. However, in the experiment, no prior assumptions on the structure of the covariance matrices are made and hence, the results obtained generalise to any Gaussian mixture models.

Linear soft margin SVMs have a user input parameter, the cost parameter, which is usually estimated using the available examples. Here again, either two-stage ordinary

CV or more ideally nested CV can be used for the estimation. However, as it is identified in Section 5.2.1 in Chapter 5, estimation is done using two-stage ordinary CV instead of nested CV. That is, prior to the tree building, a separate CV is run to determine the optimal value of the cost parameter (Γ).

The example sets used in the experiments are given in the Appendix A. The results obtained are given in Table 6.1 and Table 6.2.

Table 6.1: Classification results of SVM and HHBUT when BIC is used to determine the number of clusters.

Dataset	SVM	HHBUT	Tree Size	Avg. no. of Clusters
BS	89.8 +/- 1.1	89.6 +/- 1.2	3.9 +/- 0.3	(1.0, 1.4, 1.5)
BC	94.6 +/- 0.6	96.5 +/- 0.4	4.3 +/- 0.2	(2.9, 1.4)
BH	85.0 +/- 1.5	81.7 +/- 1.2	5.7 +/- 0.6	(3.1,2.6)
BUPA	66.2 +/-5.6	66.2 +/- 2.2	4.2 +/-0.2	(2.0, 2.2)
GLASS	64.0 +/- 2.4	59.5 +/ -2.6	6.3 +/- 0.2	(1.0, 1.3, 1.0, 1.0, 1.0, 1.0)
BNK	95.8 +/- 3.4	96.8 +/- 0.7	12.1 +/- 1.2	(6.1, 6.0)
PIND	70.6 +/- 1.9	65.6 +/- 2.1	5.6 +/- 0.3	(3.1, 2.5)
WINE	96.5 +/- 1.1	91.3 +/- 2.3	3.0 +/- 0	(1.0, 1.0, 1.0)
SUR	68.4 +/- 3.0	73.4 + /- 1.6	4.9+ /- 0.2	(3.0, 1.9)
HRT	82.7 +/- 0.9	73.4 +/ - 2.1	2.0 +/- 0	(1.0,1.0)
CLI	90.00 +/- 1.0	90.7 +/- 1.6	2.0 +/- 0	(1.0,1.0)
SEED	94.2 +/-0.9	90.4 +/- 1.3	3.0 +/- 0	(1.0,1.0,1.0)

Unlike the top-down approach, tree size is not dependent on the split finding algorithm in the bottom-up approach. In the latter approach, the cluster algorithm finds the clusters in each class and all the clusters found then become the terminal nodes of the tree. Hence, the number of terminal nodes is fixed prior to the tree building process. Only two nodes are merged at a time. Therefore, the size of the tree does not depend on the split finding algorithm.

The last column of each table shows the average number of clusters per class over the 10 repetitions of 5-fold CV. The far left figure is for the class indexed by 1 whereas the far right figure is for the class having the largest index. For all example sets, each

Table 6.2: Classification results of SVM and HHBUT when CV is used to determine the number of clusters.

Dataset	SVM	HHBUT	Tree Size	Avg. no. of Clusters
BS	84.8 +/- 1.3	84.2 +/- 1.7	10.6 +/- 0.9	(1.3, 4.7,4.6)
BC	92.7 +/- 2.2	95.9 +/- 0.8	6.8 +/- 0.5	(4.7, 2.1)
BH	84.8 +/- 1.4	80.4 +/- 1.7	9.0 +/- 0.3	(4.7,4.3)
BUPA	63.6 +/-2.8	64.8 +/- 2.1	4.5 +/-0.3	(2.2, 2.3)
GLS	65.9 +/- 2.1	62.4 +/- 2.7	13.4 +/- 0.9	(3.8, 3.0,2.7, 1.6, 1.0, 1.3)
BNK	97.8 +/- 0.9	97.4 +/- 0.9	26.3 +/- 2.2	(14.2, 12.1)
PIND	70.0 +/- 1.8	65.2 +/- 1.0	8.1 +/- 0.4	(4.5, 3.6)
WINE	96.5 +/- 1.1	90.9 +/- 2.6	3.1 +/- 0.0	(1.0, 1.1, 1.0)
SUR	69.0 +/- 2.2	73.0 + /- 1.7	5.0+ /- 0.3	(3.2, 1.8)
HRT	80.4 +/- 2.2	72.5 +/ - 2.0	2.6 +/- 0.2	(1.6,1.0)
CLI	90.2 +/- 0.9	88.9 +/- 1.1	4.5 +/- 0.4	(1.1,3.4)
SEED	92.6 +/-1.2	89.2 +/- 1.1	4.7 +/- 0.3	(1.7,1.5,1.5)

class has at least one terminal node irrespective of the size of the class. This is a significant property in the bottom-up approach as in the top-down approach, classes with very few examples can have a chance of being eliminated from the tree at the pruning stage.

Table 6.1 presents the accuracies of the SVM and HHBUT algorithms for each example set when the number of clusters is determined by the BIC criterion. The results show that the performance of HHBUT is significantly (1-SD rule) better than SVM on the BC and SUR example sets whereas SVM perform significantly better classification on the BH, PIND, WINE, HRT and SEED example sets. For all other example sets, results do not indicate significant difference between two methods.

Table 6.2 shows the accuracies of SVM and HHBUT when CV is used to determine the number of clusters. According to the empirical results, it is evident that the relative performances of both the algorithms are similar to the relative performances of them under the BIC criterion and thus, it can be concluded that the clustering method has no interaction with the splitting method with respect to the accuracy.

In terms of accuracy, SVM performs significantly better than HHBUT on five example sets whereas for the other seven example sets, HHBUT shows at least the same or better performances than SVM (based on 1-SD rule). However, the better performance of SVM comes with the extra cost of time because, prior to the tree building, the cost parameter has to be estimated for each example set.

The results of Table 6.1 and Table 6.2 reveal that more compact trees can be obtained when BIC is used to determine the number of clusters instead of CV. For all example sets, except for BUPA and SUR the tree size of BIC-based clustering is significantly smaller than that of CV-based clustering. Furthermore, BIC tree sizes for the BS, BH, GLS, BNK and CLI example sets, are almost half of the corresponding CV-based tree sizes.

However, it is important to note that, though the BIC method produces small trees, there is no trade off between accuracy and tree size. In fact the accuracy of the BIC tree on the BS example set is significantly higher than that of the CV based tree. For all other example sets, there is no significant difference observed between the BIC and CV methods on classification accuracies of both algorithms. Therefore, using BIC helps in reducing the tree size without trading off the accuracy.

When considering the time complexity of SVM and HHBUT, the time complexity of SVM, at a non-terminal node in the worst case scenario is $O(n^3)$ (Bordes, Ertekin, Weston, & Bottou, 2005). Moreover, SVM needs a cost parameter to be estimated prior to the tree building, which also affects the time complexity. Recall that HHCART has time complexity of $O(Cp^2(p+n\log n))$ in the worst case scenario. However, HHBUT always considers only two classes, and hence the time complexity of HHBUT at a non-terminal node will be $O(p^2(p+n\log n))$ and for $n >> p$ this will be equivalent to $O(p^2 n\log n)$. Therefore, HHBUT is a good alternative for SVM when $n >> p$.

A comparison is performed on the results of the bottom-up approach and the

top-down approach. In the bottom-up approach, the BIC method performs better than the CV method. Therefore, the results of the BIC-method is compared with the results of the top-down approach. For both approaches, the comparison is made against the tree building method which produces the best classification accuracy on each example set. The results for the top-down approach are extracted from Table 3.1. The results are obtained from two separate 5-fold CV, and thus, both approaches are run on different partitions. The results are given in Table 6.3.

Table 6.3: Comparison of the top-down and bottom-up approaches.
Tree Abbreviations: S=SVM, HB=HHBUT, HH(A)=HHCART(A), OC1=OC1, LC=(OC1-LC), AP=(OC1-AP).

	Bottom-Up induction			Top-Down induction		
Example Set	Tree	Accuracy	Size	Tree	Accuracy	Size
BS	S	89.8 ± 1.1	3.9 ± 0.3	HH(A)	93.7 ± 1.3	7.9 ± 1.7
BC	HB	96.5 ± 0.4	4.3 ± 0.2	HH(A)	97.0 ± 0.3	2.4 ± 0.6
BH	S	85.0 ± 1.5	5.7 ± 0.6	HH(A)	83.3 ± 0.9	6.5 ± 2.1
BUPA	HB	66.2 ± 2.2	4.2 ± 0.2	OC1	66.9 ± 2.2	8.9 ± 6.1
GLASS	S	64.0 ± 2.4	6.3 ± 0.2	LC	67.4 ± 2.0	12.0 ± 3.6
BNK	HB	96.8 ± 0.7	12.1 ± 1.2	HH(A)	99.4 ± -0.2	3.0 ± 0.3
PIND	S	70.6 ± 1.9	5.6 ± 0.3	AP	73.6 ± 1.4	15.9 ± 8.7
WINE	S	96.5 ± 1.1	3.0 ± 0.0	HH(A)	91.3 ± 1.6	3.4 ± 0.3
SUR	HB	73.4 ± 1.6	4.9 ± 0.2	HH(A)	73.5 ± 1.5	5.3 ± 2.7
HRT	S	82.7 ± 0.9	2.0 ± 0.0	OC1	77.1 ± 2.5	3.6 ± 1.0
CLI	HB	90.7 ± 1.6	2.0 ± 0.0	HH(A)	91.7 ± 1.0	2.4 ± 0.9
SEED	S	94.2 ± 0.9	3.0 ± 0.0	OC1	92.9 ± 1.8	3.6 ± 0.6

According to the results, it is difficult to find which approach provides better classification accuracy in general. The performances of the top-down approach on BS and BNK are significantly (1-SD rule) better than the bottom-up approach while for the WINE and HRT example sets the reverse is true. There is no significant difference in performance on all other example sets. Therefore, it is difficult to recommend one approach over the other. However, it is worthwhile to investigate the two approaches on a given classifier. Thus, another analysis is carried out to examine which approach

is more suitable for the proposed split finding heuristic in this thesis. That is, the comparison between HHCART(A) and HHBUT. The results for the HHCART(A) and HHBUT algorithms are extracted from Table 3.1 and Table 6.1 respectively and are given in Table 6.4.

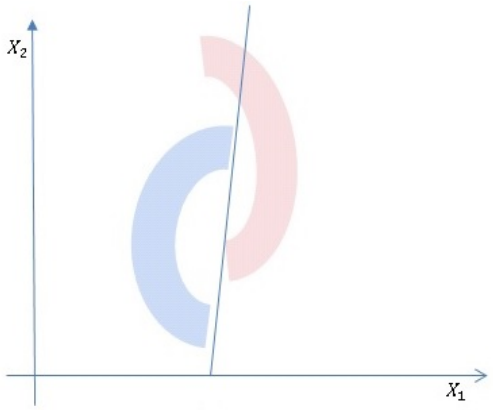
Table 6.4: Comparison between HHCART(A) and HHBUT.

	HHBUT		HHCART(A)	
Example Set	Accuracy	Size	Accuracy	Size
BS	89.6 ± 1.2	3.9 ± 0.3	93.7 ± 1.3	7.9 ± 1.7
BC	96.5 ± 0.4	4.3 ± 0.2	97.0 ± 0.3	2.4 ± 0.6
BH	81.7 ± 1.2	5.7 ± 0.6	83.3 ± 0.9	6.5 ± 2.1
BUPA	66.2 ± 2.2	4.2 ± 0.2	64.1 ± 2.6	6.5 ± 1.5
GLASS	59.5 ± 2.6	6.3 ± 0.2	60.3 ± 3.0	8.5 ± 3.0
BNK	96.8 ± 0.7	12.1 ± 1.2	99.4 ± 0.2	3.0 ± 0.3
PIND	65.6 ± 2.1	5.6 ± 0.3	72.2 ± 2.0	9.1 ± 5.1
WINE	91.3 ± 2.3	3.0 ± 0.0	91.3 ± 1.6	3.4 ± 0.3
SUR	73.4 ± 1.6	4.9 ± 0.2	73.5 ± 1.5	5.3 ± 2.7
HRT	73.4 ± 2.1	2.0 ± 0.0	74.1 ± 2.9	4.5 ± 1.7
CLI	90.7 ± 1.6	2.0 ± 0.0	91.7 ± 1.0	2.4 ± 0.9
SEED	90.4 ± 1.3	3.0 ± 0.0	90.4 ± 1.4	3.9 ± 0.8

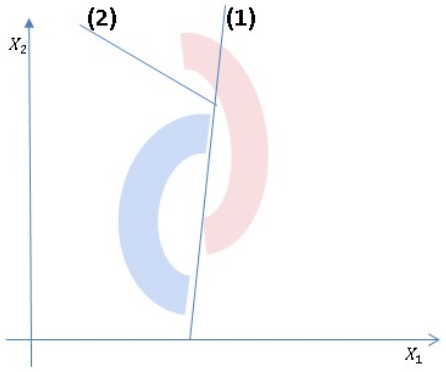
According to the results, HHCART(A) performs significantly (1-SD rule) better than HHBUT for the BS, BNK and PIND example sets. For all other example sets, HHCART(A) and HHBUT perform similarly. However, the accuracies of HHBUT are lower than that of HHCART for most of the problems. Therefore, HHCART(A) can be recommended if higher accuracy is the major concern. It can be seen that the average tree size of HHCART(A) is generally higher than that of HHBUT.

6.7 Shortcomings of the bottom-up approach

The top-down approach has a higher chance of finding better splits since the tree building methodology keeps partitioning the feature space until each sub-region becomes homogeneous (or near homogeneous) with respect to a class. However, this may increase the tree size. On the other hand, the bottom-up approach finds hyperplanes merely to separate the terminal nodes which have already been determined. Therefore, if the terminal nodes cannot be separated by a single hyperplane this approach constructs heterogeneous sub-regions and hence, produces lower accuracy. This can happen in two situations namely: (a) terminal nodes (clusters) are not linearly separable, or (b) terminal nodes (clusters) overlap each other. Both situations are illustrated in Figure 6.7 and Figure 6.8 respectively.



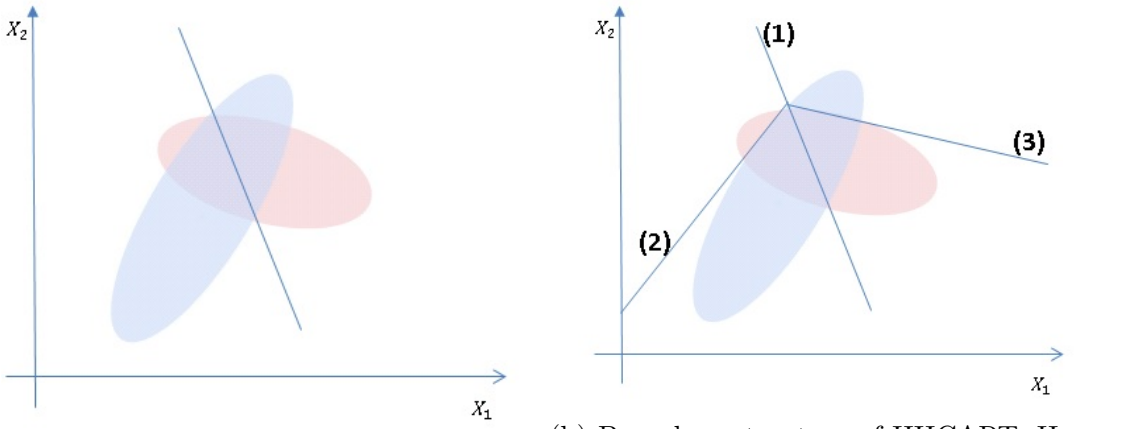
(a) Boundary structure of HHBUT.



(b) Boundary structure of HHCART. Hyperplane number is in the parenthesis.

Figure 6.7: Linearly non-separable terminal nodes.

Figure 6.7a and Figure 6.8a show that the hyperplane generated by the bottom-up approach to separate the red and blue terminal nodes (clusters). In Figure 6.7a, the hyperplane separates the blue class from the red class however, some portion of the red class has been misclassified to the region belonging to the blue class. When the terminal nodes overlap, HHBUT produces (see Figure 6.8a) heterogeneous terminal



(a) Boundary structure of HHBUT.

(b) Boundary structure of HHCART. Hyperplane number is in the parenthesis.

Figure 6.8: Overlapped terminal nodes.

nodes. The bottom-up approach does not have a mechanism for further partitioning these nodes and therefore, the final boundary structure contains heterogeneous sub-region. However, Figure 6.7b and Figure 6.8b show that the top-down approach keeps partitioning until each sub-region becomes homogeneous and thus, it has the ability to find pure sub-regions at the expense of larger tree size.

Hence, the linear separability and non-overlapping terminal nodes are critical for HHBUT's success and they are heavily dependent on the clustering algorithm and the spatial distribution of examples. Therefore, considering all these factors, the top-down approach can be recommended for the heuristic presented in this thesis.

6.8 Conclusions and discussion

This chapter explores the possibilities of improvement in the bottom-up tree induction approach. In this approach, a clustering method is used prior to the tree building to identify the terminal nodes in the tree. A pair of nodes at a time are merged until the root node is reached. The bottom-up method presented in Barros et al. (2014) used

a finite mixture modelling approach and the k-means algorithm to find the clusters. SVMs are used for splitting. Moreover, Barros et al. (2014) use CV procedure to determine the number of clusters. In this research, the bottom-up method is explored in two directions: (a) search for an effective criterion to determine the number of clusters, and (b) examine the effectiveness of the HHCART split finding principle over SVM.

Based on the empirical evidence, it can be concluded that the performance of SVMs is superior to HHBUT. However, for some problem domains HHBUT performs better than SVMs. SVMs need to tune a cost parameter before starting tree building and this can take considerable time. Except for the HRT and PIND example sets, the 2-SD confidence interval for the average accuracy of both type of trees has sufficient overlap. Hence, HHBUT is also a competitive alternative to the SVM bottom-up trees especially when $n >> p$.

The other important finding of this chapter is that the use of BIC to determine the number of clusters helps to induce better trees both in terms of accuracy and tree size. The CV procedure is shown to be ineffective in the decision tree context. The number of clusters identified by the CV method is generally larger and hence the tree size becomes large. Moreover, for some problems accuracy of the CV-based method is lower than that of the BIC-based method. These facts reveal that the cluster structure determined by CV is not favourable in the decision tree context. Therefore, based on these empirical results, BIC can be recommended for use in the bottom-up tree building procedures to determine the number of clusters.

Top-down and bottom-up induction approaches are compared. In general, there is not enough evidence to recommend one approach over the other. When comparing HHCART(A) and HHBUT the top-down approach can be recommended as it produces better classification accuracies.

However, there is room for exploring the bottom-up approach further. In this

approach, clusters (terminal nodes or meta-class nodes) are selected to merge based on the Euclidean distance between them. The cluster overlap is not taken into consideration. Therefore, it is worthwhile to investigate cluster overlapping and find a better criterion to select clusters for merging. Furthermore, ensemble methods are popular in the decision tree context to reduce the final classification error. Hence, a tree ensemble for bottom-up trees is worth exploring.

Chapter 7

Summary of conclusions and future work

The main objectives of this research work are presented in Chapter 1 and a comprehensive description of methodologies used to achieve those objectives are given in Chapter 3 - 6. This chapter summarises the entire work of this thesis briefly and highlights the main contributions of the work towards the field of study. Furthermore, future research directions, revealed through the study, are presented as possible research areas.

7.1 Summary of conclusions

The main focus of this thesis was to propose a time efficient oblique DT induction methodology using the Householder reflection. A DT is a non-parametric statistical model which uses a set of rules to predict a class or a value of a response variable given a feature vector. If the prediction is a class, then the tree is known as a classification tree, otherwise it is a regression tree. The work presented in this thesis focused on inducing classification trees. Classification trees are widely applied in many fields, such as, medicine, engineering and marketing.

Chapter 2 presented the early attempts of inducing oblique trees. Three categories of oblique decision tree induction methods were identified based on the split finding mechanism of the tree. The use of optimisation techniques can be regarded as a benchmark method as it does not make any assumptions about the distribution of examples or boundary structure between classes when finding splits. However, these methods are computationally expensive. Hence, standard statistical techniques such as linear discriminant analysis are used to find splits in a shorter time. Although they are fast, they often make assumptions about the structure of the feature covariance matrices. Meanwhile, DTs based on heuristic arguments are explored as alternative methods. In heuristic methods, a logic is assumed about the structure of the class separating boundary. These methods have been shown to be efficient and competitive compared with optimisation based DT methods. Robertson et al. (2013) proposed the CARTopt algorithm, which uses a heuristic oblique decision trees to find a minimiser of a non-smooth function. The CARTopt oblique DT first makes the orientation of a class parallel to $\mathbf{e_1}$ axis using a transformation. The orientation of a class is captured by the dominant eigenvector, $\mathbf{d}$, of the class covariance matrix. Examples are then transformed using a Householder matrix which is defined using $\mathbf{d}$, and axis-parallel splits are searched in the transformed space. The best split found in the transformed space will be oblique in the original space. However, the CARTopt algorithm is specifically designed to solve optimisation problems and hence, the CARTopt oblique decision tree is only capable of classifying two-class problems. Moreover, the transformation is done only once at the root node. This thesis explores the concept used in the CARTopt oblique decision tree to introduce a range of decision trees based on the Householder reflection for statistical data classification.

The significant contribution of this thesis is introduced in Chapter 3. The CARTopt oblique decision tree concept is comprehensively extended in a number of ways to induce a time efficient oblique decision tree for statistical data classification as an

alternative to the decision trees which use optimisation techniques. HHCART is a multi-class classifier. Different classes can take different orientations in the feature space. Moreover, the orientation of a class at different non-terminal nodes can also be varied. Therefore, HHCART performs Householder reflections at each non-terminal node to transform the example set at that node. This strategy immensely helps in effective classification, especially for multi-class data classification.

Classification problems originate from a wide range of disciplines and hence, the feature space often contains both qualitative and quantitative features. However, most oblique decision trees have been designed to work only with quantitative features which limits the applicability of such trees. On the other hand, HHCART is capable of handling both qualitative and quantitative features in the same oblique split. Therefore, HHCART is practical and useful decision tree algorithm which can be used in a wide range of classification problems.

Two versions of HHCART were presented, namely: (a) HHCART(A), and (b) HHCART(D). At each non-terminal node, HHCART(A) uses all the eigenvectors from each classes' covariance matrix to define Householder matrices. HHCART(D) instead uses only the dominant eigenvector of each class to define Householder matrices. The empirical results show that the performances of HHCART(A) are better than that of benchmark decision trees for most of problem domains. HHCART(D) performances are compatible with those benchmark methods and also, the time complexity is less than that of HHCART(A). Therefore, HHCART(D) is a good alternative for higher dimensional classification problems.

The other contribution from Chapter 3 was the investigation of effect of sampling scheme used to construct CV partitions on the tree accuracy. It was found that using stratified random sampling based CV increases the classification accuracy of minority classes.

Chapter 4 presented the proposed modification of HHCART in order to handle

massive example sets. In the first instance, it was shown that HHCART can be implemented as a disk resident algorithm. Then a HHCART implementation under the parallel computing architecture was introduced. The parallel version is helpful not only to handle massive example sets but also it helps to speed up the induction time by distributing the workload to different slave processors. The unique feature of HHCART compared to most of the other oblique decision trees is that it can be parallelised under all parallelism strategies such as task, data and hybrid parallelism. Most decision trees implemented under parallel computing architecture use axis-parallel splits. Therefore, the use of axis-parallel splits (in the transformed space) in HHCART enables the algorithm to be easily parallelised. Thus, the proposed HHCART algorithm can be considered as a flexible oblique tree algorithm where it can be implemented under any computing architecture and is a good alternative to handle classification problems with ever increasing size of data.

The two versions of HHCART, HHCART(A) and HHCART(D), use eigenvectors of classes' covariance matrices to construct Householder matrices. However, the user can replace the eigenvector, by which the Householder matrix is constructed (equation (3.3.1)), with other vectors. In Chapter 5 two such vectors are examined, namely: (a) the normal vector of the angular bisectors of the two clustering hyperplanes defined in Manwani and Sastry (2012), and (b) class representative vectors. Angular bisectors are used as splitting hyperplanes in the GDT algorithm. However, the HHGDT algorithm presented in Chapter 5 uses the normal vector of the angular bisectors to construct the Householder reflection. It is shown that HHGDT significantly improves the performance of GDT, either in terms of accuracy or tree size.

The GDT algorithm has a tuning parameter, the allowable node misclassification rate ϵ, which should be estimated prior to the tree building. Manwani and Sastry (2012) use the two stage ordinary CV to estimate ϵ although the ideal way is to use nested CV. Therefore, the method used in Manwani and Sastry (2012) may lead

to an optimistic bias in the estimator. Within the scope of the literature review of this thesis, the author was unable to find a reference to justify the use of two stage ordinary CV over nested CV. Therefore, another experiment was conducted and it was empirically shown that two stage ordinary CV is a good alternative to nested CV for estimating ϵ.

Furthermore, a new methodology is proposed to find clustering hyperplanes under the rank deficiency of matrices. The method is useful not only in GDT or HHGDT but also in any other situations where the solution of a generalised eigenvalue problem is required when rank deficiency of matrices exists.

The class representative vectors (CRV) are derived and proposed as alternative vectors to the eigenvectors in equation (3.3.1). Empirical results show that the new algorithm, HHCRV, which uses CRVs to construct Householder matrices, is a competitive method when compared with benchmark decision trees.

The final objective of this thesis was to use the Householder reflection in the bottom-up approach to induce trees in bottom-up fashion. The bottom-up strategy first uses a clustering algorithm to find clusters within each class. These clusters are considered as terminal nodes and then a pair of terminal nodes (which do not belong to the same class) are merged until the root node is reached. The clustering algorithm plays an important role in bottom-up approach. Barros et al. (2014) use finite Gaussian mixture model with the CV procedure to determine the number of clusters. In this work, it was found that the CV method is inferior to the BIC method in determining the number of clusters in terms of decision tree induction context. Empirical results show that the BIC based clustering is more accurate and produces more compact trees compared with the CV based clustering.

Barros et al. (2014) use SVM to find the separating hyperplane at each non-terminal node. In this work, the SVM method is replaced by the concept of HHCART to propose a new algorithm, HHBUT. The results show that SVM produces better

trees (based on 1-SD rule) for most of the classification problems considered. However, the better results come with extra computational time, especially as SVM has a tuning parameter which has to be estimated prior to the tree building. The classification accuracy of HHBUT is similar to that of SVM tree under 2-SD rule and therefore, HHBUT is also a good alternative to SVM.

7.2 Future work

HHCART is an easily parallelisable algorithm, a significant property when compared with benchmark oblique decision tree algorithms. Therefore, it will be an important contribution to the field of statistical learning/machine learning to introduce the parallel implementation of the HHCART algorithm. Moreover, Chapter 4 presented the methodological development of the parallel version of HHCART. Hence, the author wishes to implement the parallel version of HHCART which makes the proposed tree induction algorithm in this thesis a comprehensive tool for statistical data classification.

Tree structured classifiers are often regarded as sensitive to data perturbation. Hence, random forest models have been explored and have been shown to be more robust than a single tree. Random forest is a collection of axis-parallel unpruned trees where each unpruned tree finds the best split at each non-terminal node in the following way: (a) a subset of features is selected randomly, and (b) the best axis-parallel split is chosen from the subset to split the node. The HHCART algorithm can easily be modified to construct a random forest based oblique tree classifiers, where a subset of features can be selected in transformed spaces. However, the best split found in a transformed space is an oblique in the original space. Therefore, implementation of the random forest version of HHCART will be an important future work.

In the bottom-up tree building approach, cluster analysis is performed within each

class. Therefore, clusters found in one class are independent of the clusters found in another class. Hence, clusters belonging to different classes can overlap in the feature space. This may have an adverse effect on classification if the degree of overlap is severe. A measure of the degree of cluster overlap would be important to determine whether those clusters represent a specific class (if the degree of overlap is less than a threshold) or if the class of clusters are undecided (if the degree of overlap is grater than a threshold). Thus, inducing a bottom-up oblique DT taking into account the degree of cluster overlap will be an important future study.

As mentioned in Section 6.7, overlapping clusters belonging to different classes or linearly inseparable clusters prevent the bottom-up tree building approach producing better results when compared with the top-down approach. On the other hand, the appealing feature in this approach is that it produces at least one terminal node for each class whereas in the top-down approach, terminal nodes belonging to rare classes can be eliminated at the pruning stage. Hence, to preserve the important qualities and remove the shortcomings of both algorithms, it is worthwhile to investigate the possibility of inducing a hybrid tree building approach. In this approach, tree building starts with cluster analysis and identifies terminal nodes. Terminal nodes are then merged upwards until the root node is reached. Then the tree building procedure shifts to the top-down approach from impure terminal nodes (spatially overlapping clusters). These nodes can be split further until each node becomes homogeneous (or near homogeneous) with respect to a class.

Appendix A

Downloaded datasets used in the analysis

The following datasets were downloaded from UCI repository (Lichman, 2013). The example sets in Table A.1 contain only quantitative features while the example sets in Table A.2 contain both type of features. The descriptions of the example sets are given in the Section A.1.

A.1 Descriptions of the example sets

[1] **Balance Scale**

This example set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced.

[2] **Boston Housing**

This example set contains the housing values in suburbs of Boston along with 13 predictor variables. The intention is to predict the housing value which is a

quantitative feature. However, Murthy converts the problem into a classification problem as follows: if the house value < 21000, then the category =1 otherwise 2.

[3] **Breast Cancer**

This example set was compiled on the problem of diagnosing the two types of the breast cancer namely: benign and malignant using nine predictor variables.

[4] **BUPA**

The results of 5 blood tests, which are thought to be sensitive to the liver disorder of male individuals have been stored.

[5] **Glass**

The classification task of this example set is to predict the type of the glass using its oxide content.

[6] **Heart**

This example set contains 13 attributes to classify whether a patient suffering from a heart disease or not.

[7] **BankNote**

Examples were extracted from images that were taken from genuine and forged banknote-like specimens. The classification task is to predict the given note is genuine or forged.

[8] **Pima Indian**

The example set contains the observations made on 21 year or older Pima Indians. The aim is to diagnostic whether a given patient shows signs of diabetes.

[9] **Shuttle**

The shuttle example set contains 9 attributes all of which are numerical. Approximately 80% of the data belongs to class 1. Hence, the aim is to obtain an accuracy of 99 - 99.9%.

[10] **WINE**

The example set is complied using the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars.

[11] **Letter**

Here the objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet.

[12] **Survival**

The example set contains cases from study conducted on the survival of patients who had undergone surgery for breast cancer. The aim is to predict whether the patient is survived (died before 5 years of the surgery).

[13] **Climate**

The objective of the problem is to predict climate model simulation outcomes given scaled values of climate model input parameters.

[14] **Seed**

Measurements of geometrical properties of kernels belonging to three different varieties of wheat are stored. The objective is to predict the variety given the geometrical properties of kernels.

[15] **Income**

The prediction task of this example set is to determine whether a person makes over 50K a year.

[16] **Bank**

The example set is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

[17] **StatLog**

The example set contains the information extracted from credit card applications without disclosing the identification.

Table A.1: Real Data sets with quantitative features

Data set	No. of feature	No. of classes	No. of examples
Balance Scale (BS)	4	3	625
Boston Housing (BH)	13	2	506
Breast Cancer (BC)	9	2	638
BUPA	6	2	345
Glass (GLS)	9	6	214
Heart (HRT)	13	2	270
BankNote (BNK)	4	2	1372
Pima Indian (PIND)	8	2	768
Shuttle (SHUT)	9	7	58000
Wine(WINE)	13	3	178
Letter (LET)	16	26	20000
Survival (SUR)	3	2	306
Climate (CLI)	18	2	540
Seed (SEED)	6	3	210

Table A.2: Real Data sets with qualitative and quantitative features

Data Set	No. of features (No. of Qualitative)	No. of Classes	No. of Examples
Income	14(8)	2	45222
Bank	16(9)	2	45211
StatLog	14(8)	2	690

Appendix B

Nested CV procedure

Some classifiers require input parameters before the classifier is trained. For example, Geometric Decision Tree (GDT) (Manwani & Sastry, 2012) requires allowable node level misclassification rate ϵ, while the soft margin support vector machine requires a cost parameter or box constraint before it is trained. In the small sample cases, practitioners may not be able to use a separate portion of the example set to estimate these parameters. Therefore, they are estimated using the same examples on which the classifier is trained and tested. Usually v-fold CV is used to test the classifier. However, in this situation the use of ordinary v-fold CV may not provide a proper parameter estimation. Therefore, the nested CV procedure is used to overcome the drawback of the ordinary CV procedure and it is given below.

Let v-fold CV be used to estimate the accuracy of a classifier. First, the entire sample is partitioned to v disjoint subsets. At each time $v-1$ subsets are used to train the classifier and the other set is used to test the classifier. In the nested CV procedure, those $v-1$ subsets, the training set, are used to estimate the input parameters. The strategy is to perform another m-fold ordinary CV on $v-1$ subsets. A Schematic of the nested CV procedure is given below. Assume that the objective

of the nested CV is to estimate the node level misclassification rate ϵ of GDT. Let v and m be 5.

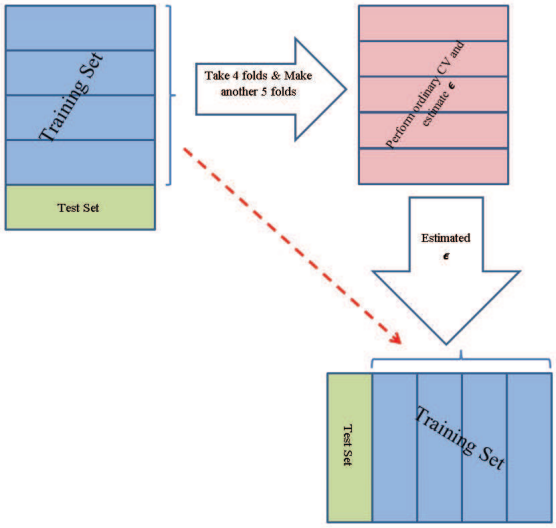


Figure B.1: Schematic of the nested CV procedure.

The steps of the nested CV procedure for a general case are given below.

1. Example set is partitioned into v-folds.

2. A training set $\mathfrak{D}_{train}$ is extracted: comprises of $v-1$ folds. The remaining fold is the test set, $\mathfrak{D}_{test}$.

3. $\mathfrak{D}_{train}$ is again partitioned into m-folds.

4. Ordinary CV procedure is applied to m-folds to estimate the optimal value for the parameter, ϵ. That is a sequence of ϵ values are generated from 0.05 to 0.4 with step size of 0.01. For each ϵ, a m-fold CV procedure is run and accuracy is estimated. The ϵ value which produces the highest accuracy was chosen as the optimal ϵ.

5. Use $\mathfrak{D}_{train}$ with the optimal value of the parameter to train the classifier.

6. $\mathfrak{D}_{test}$ is classified using the classifier and the accuracy is estimated.

7. Repeat steps 2-6 v times assigning a new fold to $\mathfrak{D}_{test}$ at the step 2.

References

A, K. M., & Gopal, M. (2010). A hybrid svm based decision tree. *Pattern Recognition*, *43*(12), 3977–3987.

Abas, A. R. (2013). On determining efficient finite mixture models with compact and essential components for clustering data. *Egyptian Informatics Journal*, *14*(1), 79–88.

Amado, N., Gama, J., & Silva, F. (2001). Parallel implementation of decision tree learning algorithms. In *Progress in artificial intelligence* (pp. 6–13). Springer.

Amasyah, M., & Ersoy, O. (2008). Cline: A new decision-tree family. *Neural Networks, IEEE Transactions on*, *19*(2), 356–363.

Ancona, N., Maestri, R., Marinazzo, D., Nitti, L., Pellicoro, M., Pinna, G., & Stramaglia, S. (2005). Leave-one-out prediction error of systolic arterial pressure time series under paced breathing. *Physiological measurement*, *26*(4), 363.

Barros, R. C., Jaskowiak, P. A., Cerri, R., & de Carvalho, A. C. (2014). A framework for bottom-up induction of oblique decision trees. *Neurocomputing*, *135*, 3–12.

Bell, J. F. (1996). Application of classification trees to the habitat preference of upland birds. *Journal of Applied Statistics*, *23*(2-3), 349–360.

Bellman, R. (1961). *Adaptive control processes: a guided tour* (Vol. 4). Princeton university press Princeton.

Ben-Haim, Y., & Tom-Tov, E. (2010). A streaming parallel decision tree algorithm. *The Journal of Machine Learning Research*, *11*, 849–872.

Bennett, K. P. (1992). *Decision tree construction via linear programming.* Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *The Journal of Machine Learning Research*, *6*, 1579–1619.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–152).

Braha, D., & Shmilovici, A. (2003). On the use of decision tree induction for discovery of interactions in a photolithographic process. *Semiconductor Manufacturing, IEEE Transactions on*, *16*(4), 644–652.

Breiman, L., Olshen, R., Friedman, J., & Stone, C. (1984). *Classification and regression trees.* CRC press.

Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine learning*, *19*(1), 45–77.

Campos, M. M., Stengard, P. J., & Milenova, B. L. (2005). Data-centric automated data mining. In *Machine learning and applications, 2005. proceedings. fourth international conference on* (pp. 8–pp).

Cantu-Paz, E. (2000). *Efficient and accurate parallel genetic algorithms* (Vol. 1). Springer Science & Business Media.

Cantu-Paz, E., & Kamath, C. (2003). Inducing oblique decision trees with evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, *7*(1), 54–68.

Chen, L.-F., Liao, H.-Y. M., Ko, M.-T., Lin, J.-C., & Yu, G.-J. (2000). A new lda-based face recognition system which can solve the small sample size problem. *Pattern recognition*, *33*(10), 1713–1726.

Dahan, H., Cohen, S., Rokach, L., & Maimon, O. (2014). Proactive data mining using decision trees. In *Proactive data mining with decision trees.* Springer.

Decaestecker, C., Remmelink, M., Salmon, I., Camby, I., Goldschmidt, D., Petein, M., ... Kiss, R. (1996). Methodological aspects of using decision trees to characterise leiomyomatous tumors. *Cytometry*, *24*(1), 83–92.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38.

Devroye, L., Györfi, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition.* Springer (New York).

DeWitt, D. J., Naughton, J. F., & Schneider, D. A. (1991). Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Parallel and distributed information systems, 1991., proceedings of the first international conference on* (pp. 280–291).

Duda, R. O., Hart, P. E., & Stork, D. G. (1999). *Pattern classification.* John Wiley & Sons,.

Esposito, F., Malerba, D., Semeraro, G., & Kay, J. (1997). A comparative analysis of methods for pruning decision trees. *Pattern Analysis and Machine Intelligence,*

IEEE Transactions on, *19*(5), 476–491.

Everitt, s. B., Landau, S., Leese, M., & Stahl, D. (2011). *Cluster analysis*. John Wiley & Sons.

Filzmoser, P., Liebmann, B., & Varmuza, K. (2009). *Repeated double cross validation*. na.

Friedl, M. A., & Brodley, C. E. (1997). Decision tree classification of land cover from remotely sensed data. *Remote sensing of environment*, *61*(3), 399–409.

Gama, J., & Brazdil, P. (1999). Linear tree. *Intelligent Data Analysis*, *3*(1), 1–22.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549.

Hand, D. J., Mannila, H., & Smyth, P. (2001). *Principles of data mining*. MIT press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (Vol. 2) (No. 1). Springer.

Heath, D., Kasif, S., & Salzberg, S. (1993). Induction of oblique decision trees. In *Ijcai* (pp. 1002–1007).

Henrichon, G., Ernest, & Fu, K.-S. (1969). A nonparametric partitioning procedure for pattern classification. *Computers, IEEE Transactions on*, *100*(7), 614–624.

Hu, J., Deng, J., & Sui, M. (2009). A new approach for decision tree based on principal component analysis. In *Computational intelligence and software engineering, 2009. cise 2009. international conference on* (pp. 1–4).

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, *5*(1), 15–17.

Ittner, A., & Schlosser, M. (1996). Non-linear decision trees-ndt. In *Icml* (pp. 252–257).

Iyengar, V. S. (1999). Hot: Heuristics for oblique trees. In *Tools with artificial intelligence, 1999. proceedings. 11th ieee international conference on* (pp. 91–98).

Johnson, R. A., & Wichern, D. W. (2002). *Applied multivariate statistical analysis* (Vol. 5). Prentice hall Englewood Cliffs, NJ.

Joshi, M. V., Karypis, G., & Kumar, V. (1998). Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Parallel processing symposium, 1998. ipps/spdp 1998. proceedings of the first merged international... and symposium on parallel and distributed processing 1998* (pp. 573–579).

Kim, H., & Loh, W.-Y. (2001). Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, *96*(454).

Kim, H., & Loh, W.-Y. (2003). Classification trees with bivariate linear discriminant node models. *Journal of Computational and Graphical Statistics*, *12*(3), 512–530.

Kolakowska, A., & Malina, W. (2005). Fisher sequential classifiers. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *35*(5), 988–998.

Kretowski, M. (2004). An evolutionary algorithm for oblique decision tree induction. In *Artificial intelligence and soft computing-icaisc 2004* (pp. 432–437). Springer.

Krstajic, D., Buturovic, L. J., Leahy, D. E., & Thomas, S. (2014). Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of cheminformatics*, *6*(1), 1–15.

Kufrin, R. (1997). Decision trees on parallel processors. *Machine Intelligence and*

Pattern Recognition, *20*, 279–306.

Landeweerd, G., Timmers, T., Gelsema, E. S., Bins, M., & Halie, M. (1983). Binary tree versus single level tree classification of white blood cells. *Pattern Recognition*, *16*(6), 571–577.

Li, N., Zhao, L., Chen, A.-X., Meng, Q.-W., & Zhang, G.-F. (2009). A new heuristic of the decision tree induction. In *Machine learning and cybernetics, 2009 international conference on* (Vol. 3, pp. 1659–1664).

Li, X. B., Sweigart, J. R., Teng, J. T., Donohue, J. M., Thombs, L. A., & Wang, S. M. (2003). Multivariate decision trees using linear discriminants and tabu search. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *33*(2), 194–205.

Li, Y., Dong, M., & Kothari, R. (2005). Classifiability-based omnivariate decision trees. *Neural Networks, IEEE Transactions on*, *16*(6), 1547–1560.

Lichman, M. (2013). *UCI machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Lim, T.-S., Loh, W.-Y., & Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, *40*(3), 203–228.

Liu, K., Cheng, Y.-Q., Yang, J.-Y., & Liu, X. (1992). An efficient algorithm for foley–sammon optimal set of discriminant vectors by algebraic method. *International Journal of Pattern Recognition and Artificial Intelligence*, *6*(05), 817–829.

Loh, W.-Y., & Shih, Y.-S. (1997). Split selection methods for classification trees. *Statistica sinica*, *7*(4), 815–840.

Loh, W.-Y., & Vanichsetakul, N. (1988). Tree-structured classification via generalized

discriminant analysis. *Journal of the American Statistical Association*, *83*(403), 715–725.

López-Chau, A., Cervantes, J., López-García, L., & Lamont, F. G. (2013). Fisher's decision tree. *Expert Systems with Applications*, *40*(16), 6283–6291.

Malerba, D., Esposito, F., & Semeraro, G. (1996). A further comparison of simplification methods for decision-tree induction. In *Learning from data* (pp. 365–374). Springer.

Manwani, N., & Sastry, P. (2012). Geometric decision tree. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *42*(1), 181–192.

Marron, J. S., & Wand, M. P. (1992). Exact mean integrated squared error. *The Annals of Statistics*, 712–736.

McLachlan, G., & Peel, D. (2004). *Finite mixture models*. John Wiley & Sons.

Mehta, M., Agrawal, R., & Rissanen, J. (1996). Sliq: A fast scalable classifier for data mining. In *Advances in database technologyedbt'96* (pp. 18–32). Springer.

Mingers, J. (1987). Expert systems-rule induction with statistical data. *Journal of the operational research society*, 39–47.

Mingers, J. (1989a). An empirical comparison of pruning methods for decision tree induction. *Machine learning*, *4*(2), 227–243.

Mingers, J. (1989b). An empirical comparison of selection measures for decision-tree induction. *Machine learning*, *3*(4), 319–342.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis* (Vol. 821). John Wiley & Sons.

Moro, S., Laureano, R., & Cortez, P. (2011). Using data mining for bank direct marketing: An application of the crisp-dm methodology. In *Proceedings of*

european simulation and modelling conference-esm'2011 (pp. 117–121).

Murthy, S., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of artificial intelligence research*.

Murthy, S., & Salzberg, S. (n.d.). *The OC1 decision tree system., year = 1995, url = //http://salzberg-lab.org/software/,*.

Murthy, S., & Salzberg, S. (1995a). Lookahead and pathology in decision tree induction. In *Ijcai* (pp. 1025–1033).

Murthy, S., & Salzberg, S. L. (1995b). *On growing better decision trees from data* (Unpublished doctoral dissertation). Citeseer.

Nelder, J. A., & McCullagh, R. (1989). *Generalized linear models*. Springer.

Olaru, C., & Wehenkel, L. (2003). A complete fuzzy decision tree technique. *Fuzzy sets and systems*, *138*(2), 221–254.

Patil, D. D., Wadhai, V., & Gokhale, J. (2010). Evaluation of decision tree pruning algorithms for complexity and classification accuracy. *International Journal of Computer Applications*, *11*(2).

Podgorelec, V., Kokol, P., Stiglic, B., & Rozman, I. (2002). Decision trees: an overview and their use in medicine. *Journal of medical systems*, *26*(5), 445–463.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81–106.

Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies*, *27*(3), 221–234.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Robertson, B. L., Price, C. J., & Reale, M. (2013). Cartopt: a random search method

for nonsmooth unconstrained optimization. *Computational Optimization and Applications*, *56*(2), 291–315.

Robertson, B. L., Price, C. J., & Reale, M. (2014). A cartopt method for bound constrained global optimization. *ANZIAM Journal*, *55*, 109–128.

Roeder, K., & Wasserman, L. (1997). Practical bayesian density estimation using mixtures of normals. *Journal of the American Statistical Association*, *92*(439), 894–902.

Rokach, L. (2008). *Data mining with decision trees: theory and applications* (Vol. 69). World scientific.

Sarkar, U., Chakrabarti, P., Ghose, S., & DeSarkar, S. (1994). Improving greedy algorithms by lookahead-search. *Journal of Algorithms*, *16*(1), 1–23.

Scull, P., Franklin, J., & Chadwick, O. (2005). The application of classification tree analysis to soil type prediction in a desert landscape. *Ecological Modelling*, *181*(1), 1–15.

Shafer, J., Agrawal, R., & Methta, M. (1996). Sprint: A scalable parallel classifier for data mining. *In Proc. 22nd Int. Conf. Very Large Databases*, 544–555.

Shawe-Taylor, J., & Cristianini, N. (2000). Support vector machines. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, 93–112.

Smyth, P. (1996). Clustering using monte carlo cross-validation. In *Kdd* (pp. 126–133).

Smyth, P. (2000). Model selection for probabilistic clustering using cross-validated likelihood. *Statistics and Computing*, *10*(1), 63–72.

Sreenivas, M. K., Alsabti, K., & Ranka, S. (1999). Parallel out-of-core divide-and-conquer techniques with application to classification trees. In *Parallel processing, 1999. 13th international and 10th symposium on parallel and distributed processing, 1999. 1999 ipps/spdp. proceedings* (pp. 555–562).

Srivastava A, A., Han, E.-H., Kumar, V., & Singh, V. (2002). *Parallel formulations of decision-tree classification algorithms*. Springer.

Tirenni, G., Kaiser, C., & Herrmann, A. (2007). Applying decision trees for value-based customer relations management: Predicting airline customers' future values. *Journal of Database Marketing & Customer Strategy Management*, *14*(2), 130–142.

Utgoff, P. E., & Brodley, C. E. (1991). Linear machine decision trees. *COINS Technical Report 91-10*.

V, K., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing: design and analysis of algorithms*. Benjamin/Cummings Publishing Company Redwood City, CA.

Vapnik, V. (2000). *The nature of statistical learning theory*. Springer Science & Business Media.

Varma, S., & Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, *7*(1), 91.

Yang, Z. R., & Zwolinski, M. (2001). Mutual information theory for adaptive mixture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *23*(4), 396–403.

Yildiz, O. T., & Alpaydin, E. (2000). Linear discriminant trees. In *Icml* (pp. 1175–1182).

Yıldız, O. T., & Dikmen, O. (2007). Parallel univariate decision trees. *Pattern Recognition Letters*, *28*(7), 825–832.