Adaptive Construction of Classification Decision Trees

by

Ahmad S. Moghrabi

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Engineering (M. Eng.)
in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
Faculty of Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6, Canada

June 11, 2004

© 2004, Ahmad Moghrabi

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file   Votre référence

ISBN: 0-612-97465-0

Our file   Notre référence

ISBN: 0-612-97465-0

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

Abstract

Better classification decision trees can be constructed by adaptively choosing the most suitable subdivision algorithm for each data-points subset than by using the same subdivision algorithm on all the subsets as is normally done. This thesis presents results for decision tree construction procedures that adaptively choose from among a set of separating-surface placement algorithms as the decision tree is being constructed. The results show that this new approach to decision tree construction improves the overall classification accuracy of the decision tree.

Dedicated to
Michael, Gabriel, and Daniel

Acknowledgments

The author wishes to thank his wife Nadine for support and his thesis supervisor Professor John W. Chinneck for guidance.

Table of Contents

1	Introduction	1
1.1	Contributions	3
1.2	Overview of the thesis.....	4
2	Decision Trees: A Literature Survey	5
2.1	Overview.....	5
2.2	Basics of Decision Tree Construction	8
2.2.1	Separating Surfaces	8
2.3	Using Mathematical Programming for Finding Multivariate Splits	14
2.3.1	Chinneck's Surface-placement Methods.....	15
2.4	Pruning.....	22
2.4.1	Reduced Error Pruning.....	23
2.4.2	Pessimistic Error Pruning.....	24
2.4.3	Minimum Error Pruning	24
2.4.4	Critical Value Pruning	24
2.4.5	Cost-Complexity Pruning.....	25
2.4.6	Error-Based Pruning	25
3	Towards Better Decision Trees	26
3.1	Decision Tree Construction by Classical Classifiers.....	26
3.2	Problem Statement.....	27
4	Adaptive Decision Tree Construction	28
4.1	Definitions	28
4.2	Adaptive Tree Construction Policies.....	29
4.2.1	Policy 1: Highest Total Accuracy	30
4.2.2	Policy 2: Method Dependent Ranking.....	32
4.2.3	Policy 3: Method Dependent Ranking with Minority Protection.....	35
4.2.4	Policy 4: Method Dependent Ranking with Mixed-Zone Detection	39
4.2.5	Policy 5: Lowest Gini Index.....	42
4.2.6	Policy 6: Highest Information Gain Ratio	43
4.2.7	Policy 7: Lowest Pessimistic Estimated Error.....	45
4.2.8	Policy 8: Lowest Zone Gini Index.....	47
4.2.9	Policy 0: Best of Policies 1-8.....	49
5	Experimental Setup	51
5.1	Subdivision Algorithms.....	51
5.2	Stopping Criterion.....	53
5.3	Pruning Algorithm	54
5.4	Accuracy Calculation	54
5.5	Software Failures	55
5.6	Non-Adaptive and Baseline Classifiers	55
5.7	UCI Datasets.....	56
5.7.1	Datasets Excluded	56
5.7.2	Datasets used in the experiments.....	59

6 Results and comparisons.....	61
6.1 Results	61
6.2 Comparisons	64
6.2.1 Performance Profiles	64
6.2.2 Adaptive Versus Non-Adaptive	65
6.2.3 Individual Policies Versus Non-Adaptive	67
6.2.4 Execution time.....	69
6.2.5 Performance profiles with software failures eliminated.....	70
7 Conclusions	77
7.1 Research directions	78

List of figures

Figure 1: A hypothetical data set and a decision tree that can be used for classification.....	7
Figure 2: Gini's measure for income inequality	10
Figure 3: A graphical comparison between multivariate and univariate decision trees.....	14
Figure 4: A decision tree constructed using policy 1	32
Figure 5: A decision tree constructed using policy 2.....	35
Figure 6: Protecting a minority class with policy 3. Policy 2 (top) vs. policy 3 (bottom).	38
Figure 7: Avoiding over-fitting with policy 4. Policy 2 (top) vs. policy 4 (bottom).	41
Figure 8: A decision tree constructed using Policy 5	43
Figure 9: A decision tree constructed using Policy 6	45
Figure 10: A decision tree constructed using Policy 7	47
Figure 11: A decision tree constructed using Policy 8	49
Figure 12: Adaptive Policy 0 versus Non-Adaptive Performance Profiles.....	66
Figure 13: Performance Profiles of Adaptive Policies 1, 2, and 3.....	68
Figure 14: Performance Profiles of Adaptive Policies 4, 5, and 6.....	69
Figure 15: Performance Profiles of Adaptive Policies 7 and 8	70
Figure 16: Adaptive Policy 0 vs. Non-Adaptive Performance Profiles with the effect of robustness eliminated.....	73
Figure 17: Performance Profiles of Adaptive Policies 1, 2, and 3 with the effect of robustness eliminated	74
Figure 18: Performance Profiles of Adaptive Policies 4, 5, and 6 with the effect of robustness eliminated	75
Figure 19: Performance Profiles of Adaptive Policies 7 and 8 with the effect of robustness eliminated.....	76

List of tables

Table 1: Numerical example of calculating the Gini Index	11
Table 2: Numerical example of calculating the information-gain to split-information ratio	13
Table 3: Chinneck's algorithm method summary	52
Table 4: Converting attribute values from categorical to numerical.....	57
Table 5: A list of the datasets used in the experiments.....	60
Table 6: Experimental Results.....	62
Table 7: Experimental results with software failures eliminated	71

Introduction

Classification is a fundamental process in the fields of Artificial Intelligence and Expert Systems. It involves predicting the class or type of data samples based on rules that are learned or discovered by automatic exploration of previous data samples with known classes.

Advances in data gathering, storing, and processing technologies facilitate and at the same time necessitate the use of classification methods. The readily available digitized data in any discipline you can imagine coupled with cheap yet very powerful processors make it quite feasible to employ computers to classify data and assist in decision-making. The sheer volume of the data gathered and stored makes it nearly impossible to rely solely on humans to discover the rules and patterns buried in these vast amounts of data.

Examples abound of possible applications of automated classification and fields in which it can be used. An astronomer trying to classify stars and galaxies based on telescope photos or a microbiologist trying to identify a bacterium based on microscope images. A forensic scientist trying to find out the source of a glass fragment based on lab description of its properties or a financial advisor trying to make an educated guess of the future performance of a certain stock based on historical data. All can automate their tasks by feeding the appropriate data to a classifier. Perhaps the use of classifiers for diagnosis and

prediction of prognosis in the field of medicine is the biggest testimony to their importance.

Different technologies and methodologies can be used in building classifiers and each has its own advantages and disadvantages. Most classifiers are built using either decision trees or artificial neural networks. Decision trees can extract classification rules from relatively small training sets (data samples with known classes) and they are transparent when it comes to exposing their discovered rules for human examination and tweaking. However, decision trees require that the output (i.e. class) to be a discrete attribute and they are sensitive to errors and omissions in the datasets. Artificial neural networks, on the other hand, are very robust and can produce both discrete and real-valued output but they need relatively large training datasets before they can produce accurate predictions and their classification rules are not readily accessible. The research work of this thesis focuses on decision-tree classifiers.

Decision tree classifiers use a recursive divide-and-conquer strategy to build decision trees in which the internal nodes represent tests, the branches represent the outcome of these tests and the leafs represent class labels that are assigned to sample data. At the core of this strategy is choosing the actual test that is performed at the internal nodes. Different classifiers employ different algorithms to generate the tests or splits at the internal nodes of a decision tree. These algorithms usually try to maximize a certain measure of merit associated with the internal splits. Examples of such measures of merit include the increase of homogeneity at the node, the information gain, or the classification accuracy measured on the training dataset.

Chinneck’s subdivision algorithm [Chinneck 2003] is one of many algorithms that can be used by a classifier to generate classification splits as it builds a decision tree. One unique property of this algorithm, however, is that it is highly parameterized allowing the calling routine (i.e. the classifier) to fine-tune its internal working including its measure of merit.

This thesis considers the effect on the overall prediction accuracy of the resulting decision tree of adaptively changing the subdivision algorithm while a decision tree is being constructed. A Classifier is implemented using Chinneck’s subdivision algorithm, which was chosen for the flexibility it offers. A comprehensive set of experiments using publicly available datasets is then performed to compare the prediction accuracy of adaptively constructed versus non-adaptively constructed decision trees. The results of these experiments show that by adaptively varying the subdivision algorithm including its measure of merit while the classifier is constructing a decision tree, the overall prediction accuracy of the resulting decision tree is generally improved.

1.1 Contributions

The main contributions of this thesis are the following:

- Demonstrating that classifiers capable of adaptive modification of their subdivision algorithms can produce decision trees with higher prediction accuracy than those produced by similar classifiers but with a fixed subdivision algorithm.
- Development of a set of algorithms (here called policies) that are used to define systematic methods by which a classifier can adaptively change its subdivision algorithm while it is constructing a decision tree.

1.2 Overview of the thesis

This thesis can be viewed as consisting of three parts. Part one presents the existing work in the field of decision trees classifiers with some historical background. Part two states the research problem, proposes a solution and describes in detail the experimental setup to test the validity of the solution. Part three presents the results of the conducted experiments, draws conclusions and suggests possible continuations to this work.

With the exception of chapter two, the thesis is experimental in nature; it presents empirical observations and draws conclusions based on carefully designed and controlled experiments.

Decision Trees: A Literature Survey

2.1 Overview

Decision trees are tools used for classification, prediction and data exploration.

A decision tree is induced or constructed from a training set, which consists of instances. Each instance is described by a set of attributes and a class and is represented by an n -dimensional point. Attributes can have ordered values, e.g. integer and decimal numbers, or they may have unordered values, e.g. colours or gender. We will be looking at different methods of constructing decision trees later on in this chapter.

Decision trees contain two types of nodes, internal nodes and leaf nodes. Internal nodes represent tests and leaf nodes represent classes that the points are thought to belong to. Branches connecting the nodes represent transitions.

A decision tree is traversed top-down starting from a root node. At an internal node a function of the attributes or independent variables of the data point is evaluated and the outcome determines the child node to which a transition is taken. If the target node is an internal node, it will have its own test function and the process is repeated. If, on the other hand, the target node is a leaf node, it will have a class label and the data point or object is classified accordingly.

Figure 1 shows a hypothetical set of points and a decision tree that can be used for classifying these points. The decision tree shown is a binary decision tree i.e. it has exactly two child nodes for each internal node. In this example, “X” points belong to class “A” while “O” points belong to class “B”.

Compared to artificial neural networks, the classification rules of a decision tree can be more readily expressed as simple mathematical equations. In fact, a decision tree is nothing but a representation of the underlying set of classification rules. Depending on the application, this advantage could be of great importance; for example where a human verification of the classification rules is desired. Moreover, decision trees provide better accuracies than artificial neural networks when dealing with small training sets [Markham et al. 1998].

Data Set: **Decision Tree:**

Figure 1: A hypothetical data set and a decision tree that can be used for classification.

2.2 Basics of Decision Tree Construction

Decision trees are constructed for many reasons, the more common of which are discrimination and classification. Discrimination is the process of extracting classification rules from a training set and classification is the process of applying such rules to points in a data set with unknown classes as a way of deducing or guessing their classes [Hand 1981].

The actual construction of a decision tree can be summarized in the following high-level algorithm [Murthy 1997]:

1. If all the training examples at the current node belong to the same class “c”, assign class label “c” to the current node making it a leaf node and stop.
2. Otherwise, choose a rule that will partition the training examples at the current node and create a child node from each partition. The current node is thus an internal node.
3. Apply the algorithm recursively to each of the child nodes.

2.2.1 Separating Surfaces

The most important part of constructing a decision tree is finding the rule at each internal node that is useful in discriminating among the different classes present in the data set.

In finding the decision rule at an internal node, one can look for the most discriminating attribute of the data set at the node in question and then base the decision rule on that single attribute. The split in this case would be univariate.

Another possibility would be to base the decision rule on a combination of data set attributes at the internal node and thus have a multivariate decision rule.

Univariate decision trees

Most of the early work on decision trees used univariate, also known as axis-parallel, splits. Of these algorithms, perhaps Leo Breiman's CART [Breiman et al. 1984] and Ross Quinlan's ID3 [Quinlan 1986] and later C4.5 [Quinlan 1993] and C5.0 are the most popular.

CART uses a diversity measure to decide on which of the independent variables is the best candidate to use in the splitter function. A high diversity index indicates an even distribution of classes in the data set while a low diversity index indicates that one class is predominant. The objective is therefore to find a splitter that decreases the diversity by the greatest amount. In other words, the best splitter is the one that maximizes:

$$Diversity(\text{internal node}) - (Diversity(\text{left child}) + Diversity(\text{right child}))$$

Breiman adopted the diversity measure from economics. The Italian economist Gini devised a measure for income inequality. On a plot where the X-axis represents the percentage of population ordered from the poorest to the richest and the Y-axis represents the cumulative percentage of the total income, if the income were equally distributed then the plot would be a straight line at 45° angle. However, in real life this is never the case. The actual plot is more likely to be a curve in which the poorest 10% of the population receive less than 10% of the total income and the richest 10% receive more than 10% of the total income.

Gini's index [Xu 2003] is calculated as the area between the 45° line and the actual curve.

The higher the score the less even is the distribution. In figure 2, the Gini measure is calculated as the shaded area between the ideal and actual plots on a normalized graph.

Figure 2: Gini's measure for income inequality

For CART, an adaptation of the Gini index is used as a measure of impurity and is calculated as follows:

$$GINI(D) = 1 - \sum_j [p(D, j)]^2$$

Where $p(D, j)$ is the relative frequency of class j at node D .

If n_c is the total number of classes, then $GINI(D)$ has a maximum of $\left(1 - \frac{1}{n_c}\right)$ when

records are equally distributed among all classes, implying least interesting information.

The minimum is 0 when all records belong to one class, implying most interesting information.

Table 1 gives an example of how the Gini index varies with the distribution of records among classes.

Table 1: Numerical example of calculating the Gini Index

Class A records	Class B records	Gini index
6	0	0.000
5	1	0.278
4	2	0.444
3	3	0.500

The best univariate splitter is found by first finding the value that minimizes the Gini index from the range of allowable values of each attribute, then choosing the lowest among all attributes.

Quinlan has been refining his classification algorithm for years. He started with ID3 then evolved it to C4.5, and most recently to C5.0. C4.5 is very similar to CART with only a few differences. The most noticeable difference is while CART produces a binary tree, C4.5 produces trees with varying number of branches per node. This difference is the result of the way C4.5 treats categorical variables. C4.5 creates a branch for each possible value of the categorical variable. For continuous variables, both CART and C4.5 use binary splits.

Another important difference between CART and C4.5 is the default splitting criterion. While Gini is the default splitting criterion for CART, C4.5 uses the gain ratio as its default splitting criterion.

The *gain ratio* is used to assess the desirability of a test and is defined as the ratio of information gain of a test to its split information.

The information gain is defined as follows:

$$Gain(D, T) = Info(D) - \sum_{i=1}^k \frac{|D_i|}{|D|} \times Info(D_i)$$

Where T represents a test that divides the set of cases D into k subsets $D_1, D_2, \dots, D_k$.

If C is the number of classes and $p(D, j)$ is the proportion of cases in D that belong to class j , then the residual uncertainty of the class to which a case in D belongs is defined as:

$$Info(D) = - \sum_{j=1}^C p(D, j) \times \log_2(p(D, j))$$

The split information is defined as:

$$Split(D, T) = - \sum_{i=1}^k \frac{|D_i|}{|D|} \times \log_2 \left(\frac{|D_i|}{|D|} \right)$$

From all possible tests, the one with the maximum gain ratio is selected provided that it is greater than a preset threshold.

Table 2 presents, as an example, a hypothetical dataset consisting of 9 type A points and 5 type B points. Two splits of the dataset are considered. The calculation of the Information

Gain to the Split Information ratio for each split shows that Split 2 provides a better subdivision than Split 1.

Table 2: Numerical example of calculating the information-gain to split-information ratio

Split	Classified A		Classified B		Information Gain	Split Information	Ratio
	Actual A	Actual B	Actual A	Actual B			
Split 1	6	2	3	3	0.048127	0.985228	0.048849
Split 2	6	1	3	4	0.151836	1	0.151836

Multivariate decision trees

Most of the work on multivariate decision trees uses linear combinations of attributes to define splits at the internal nodes; such decision trees are sometimes referred to as oblique decision trees.

Finding optimal multivariate separating surfaces is much more difficult than finding univariate splits. Thus, heuristics are developed for finding good splits but optimality is never guaranteed.

In general, multivariate decision trees are much smaller than their axis parallel counterparts and in most cases they provide a better representation of the underlying domain subdivisions. Figure 3 shows how a two dimensional domain can be divided by just two multivariate (left) lines that are a true representation of the class boundaries while a

stepwise univariate (right) approximation of those boundaries cannot be as clear and simple.

Figure 3: A graphical comparison between multivariate and univariate decision trees

Breiman et al. [1984] were the first to propose a multivariate decision tree classifier. It was a variation on CART called CART with linear combination or CART-LC for short. CART-LC starts with a univariate split and then systematically applies linear perturbations to the separating surface and chooses the surface that maximizes a goodness measure; that surface may very well be the original axis parallel surface. Murthy, Kasif, and Salzberg [1994] later introduced the Oblique Classifier 1 or OC1 as an extension to CART-LC. OC1 addressed some of the limitations in CART-LC - most notably, OC1's non-deterministic (randomized) search that enabled it to escape local minima on which CART-LC may get stuck.

2.3 Using Mathematical Programming for Finding Multivariate Splits

Mathematical programming can be utilised in finding multivariate separating surfaces. In the early 1970s Duda and Hart [1973] proposed using linear programming for finding

splits by minimizing the distance of misclassified points from the separating hyper-plane. Later, Mangasarian, Bennett, and Bredensteiner [Bennett and Mangasarian 1994] [Bennett and Bredensteiner 1997] used linear and bilinear programming to find the separating surface that minimizes the number of misclassified points. They also demonstrated that minimizing the number of misclassified points rather than the distance of such points to the separating surface produces better accuracies. More recently, Chinneck [1998] adapted his heuristic for infeasibility analysis of linear programs to find separating surfaces by also minimizing the number of misclassified points. Chinneck's results show an improvement in accuracy at the price of execution time. Chinneck, later on, addressed the performance issue and was able to significantly improve the execution speed with only a marginal drop in the accuracy of the original heuristic [Chinneck 2001].

2.3.1 Chinneck's Surface-placement Methods

Chinneck's algorithm involves converting a training dataset of I instances ($i = 1 \dots I$) and J attributes ($j = 1 \dots J$) to a set of linear constraints as follows:

- If point i is of type 0 then it is represented by: $\sum_j d_{ij} w_j \leq w_0 - \epsilon$
- If point i is of type 1 then it is represented by: $\sum_j d_{ij} w_j \geq w_0 + \epsilon$

Where d_{ij} is attribute j of point i and ϵ is a small positive constant, normally 1. The variables are w_j , the unknown parameters of the separating hyperplane.

If the constraints can be simultaneously satisfied, then solving the system using linear programming yields a set of values for w_j , which in turn represents a hyperplane $\sum_j w_j x_j = w_0$ that completely separates the type 0 from type 1 points. In the more general case where the points cannot be completely separated by a single hyperplane, the set of constraints resulting from the conversion cannot be simultaneously satisfied and the corresponding linear program is said to be infeasible. The problem, in this case, becomes that of finding the Maximum Feasible Subsystem of constraints. Alternatively, the problem is to find the minimum cardinality set of constraints that has to be removed from the system to render it feasible. In the corresponding classification problem, the removed constraints represent data points that will usually end up misclassified. Finding a Maximum Feasible System is equivalent to finding a separating hyperplane that maximizes the overall classification accuracy.

Chinneck's [2001] heuristic algorithm for finding the maximum feasible subsystem removes constraints one by one as it finds a large (in many cases maximum) cardinality feasible subsystem of constraints. In the classification problem, this means that data points are removed one by one from the original training set until the remaining points are completely classifiable by a single separating hyperplane. Chinneck uses this as an opportunity to control the removal of data points so that other objectives in addition to the original maximization of the total classification accuracy can be pursued. Examples of additional objectives that the heuristic can accommodate are:

- Maintaining the ratio of cardinalities of type 0 to type 1 in the retained data points the same as that in the original dataset within a preset margin of error.
- Balancing the population accuracies for type 0 and type 1.
- Balancing the zone accuracies for type 0 and type 1.
- Maintaining zone accuracy for type 0 above a certain threshold.
- Maintaining zone accuracy for type 1 above a certain threshold.
- Preventing the occurrence of degenerate solutions (i.e. where all the data points are on the same side of the separating hyperplane).

In the above, a *population* is a set of points belonging to the same type (i.e. *actual* type 0 or 1). A *zone* is a set of point on one side of the separating hyperplane (i.e. *nominal* type 0 or 1).

The various accuracies are defined as follows:

The total classification accuracy: $A_t = \frac{N_{c0} + N_{c1}}{N_{t0} + N_{t1}}$

The population accuracy for type 0: $A_{p0} = \frac{N_{c0}}{N_{t0}}$

The population accuracy for type 1: $A_{p1} = \frac{N_{c1}}{N_{t1}}$

The zone accuracy for type 0: $A_{z0} = \frac{N_{c0}}{(N_{t1} - N_{c1}) + N_{c0}}$

The zone accuracy for type 1: $A_{z1} = \frac{N_{c1}}{(N_{t0} - N_{c0}) + N_{c1}}$

Where N_{t0} is the cardinality of points of type 0, N_{t1} is the cardinality of points of type 1, N_{c0} is the number of points of type 0 that are correctly classified by the separating hyperplane, and N_{c1} is the number of points of type 1 that are correctly classified by the hyperplane.

The current implementation of Chinneck's algorithm provides a set of methods for the placement of the separating hyperplane. In essence, each method is an implementation of an objective or condition to be fulfilled while trying to meet the original objective of maximizing the total classification accuracy. The methods are as follows:

- **Balance the population fractions:** This method tries to keep the fraction of the smaller population in the retained data points equal to its fraction in the original dataset within a predefined margin of error. For example, consider a dataset composed of 60% type 0 points and 40% type 1 points. With a $\pm 25\%$ margin of error, the fraction of the smaller population (in this case type 1) will be maintained in the range $0.4 \pm 0.25(0.4)$ i.e. between 30% and 50% of the retained set of points. Consequently, the fraction of the type 0 population will be maintained between 50% and 70% of the retained set of points.

- **Balance the population accuracies:** This method tries to keep the two population accuracies as close in value to each other as possible. The current implementation of Chinneck's algorithm provides three different versions of this method. In version 1 only the removed points are considered for estimating the classification accuracy, all these points are assumed to be misclassified. Version 2 is similar but it considers the remaining points and assumes that all of them will be correctly classified by the final separating surface. Version 3 calculates the classification accuracies by examining all the points in the dataset.
- **Balance the zone accuracies:** This method tries to keep the two zone accuracies as close in value to each other as possible. There three implementation versions of this method as well. The differences between the versions are the same as in the previous method.
- **Favour the type 0 zone accuracy:** This method tries to keep the zone accuracy for type 0 above a predefined threshold. High thresholds usually result in low zone accuracy for type 1. The current implementation of the algorithm has two versions of this method. In version 1, the classification accuracy is estimated by assuming that all the removed points will end-up misclassified by the final separating surface while version 2 calculates the classification accuracy using all the points in the dataset.
- **Favour the type 1 zone accuracy:** This method tries to keep the zone accuracy for type 1 above a predefined threshold. High thresholds usually result in low zone

accuracy for type 0. As in the previous method, there are two implementation versions of this method.

- **Prevent degenerate hyperplane placement:** This method prevents the placement of a degenerate separation hyperplane. A degenerate separation hyperplane has the effect of putting all the data points on one side of the hyperplane i.e. classifying them as belonging to one type. This usually happens when one type constitutes the majority of the dataset. Classifying all the points as belonging to this type produces a high overall accuracy. However, this will also cause all the points from the minority type to be misclassified. Chinneck found that the most effective remedy to this situation is to replace the degenerate separation hyperplane by one produced with a method that favours the zone accuracy of the majority type with the threshold accuracy set higher than the overall accuracy obtained by the degenerate hyperplane.

For a detailed discussion of Chinneck's separating hyperplane placement methods, please refer to [Chinneck 2003].

In addition to providing control over the choice of the line placement method to be used, the implementation of Chinneck's algorithm allow the calling routine to select one of three maximum-feasible-system heuristics to be used and to choose from one of four *secondary objectives*. A secondary objective provides a chance to make a final adjustment to the position of a separating surface after the heuristic has finished execution and without causing any of the remaining points to be misclassified.

The current implementation of Chinneck's algorithm includes the following heuristics:

- **Original heuristic:** Chinneck [1998] presented this heuristic as a tool for placing linear separation surfaces with improved total accuracies. Although very accurate, this heuristic is slow because it solves a potentially large number of linear programs for each point removed.
- **Maximum-product heuristic:** This is a fast version [Chinneck 2001] of the original heuristic where only 1 linear program is solved for each point removed. While the original heuristic solves multiple linear programs to determine the point whose removal will produce the largest reduction in the sum of infeasibilities, this heuristic uses the product $(\text{constraint violation}) \times |(\text{constraint sensitivity})|$ as a predictor of the best candidate for removal. The point with the highest product is removed. The maximum-product heuristic provides a drastic improvement in execution speed over the original heuristic with only minor reduction in the classification accuracy.
- **Close-to-maximum-product heuristic:** In this heuristic a small number of linear programs are solved for each removed point. The set of points that are candidate for removal is reduced to those whose $(\text{constraint violation}) \times |(\text{constraint sensitivity})|$ product is within 99% of the maximum product. The original heuristic is used to determine the best candidate for removal from this reduced set of points. Because more than one LP may be solved for each removed point, this

heuristic is slower than the maximum-product heuristic. The accuracies of the two heuristics are about the same.

After the selected heuristic places a separation surface in a dataset, one last step can be applied to make a final adjustment of the position of the hyperplane. The implementation of Chinneck's algorithm allows the calling routine to request one of four possible adjustments:

- **No adjustment:** Simply leave the separating surface as was placed by the heuristic.
- **Maximize the total slack of the remaining points:** This tends to push the separating surface towards the lower cardinality type.
- **Minimize the total slack of the removed points:** This adjustment minimizes the total or average distance of misclassified points from the separating surface.
- **Average the positions:** In this option, the total slack of the remaining type 1 points is minimized and the coefficients of the hyperplane are noted then the total slack of the remaining type 0 is minimized and the new coefficients of the hyperplane are noted. The final separating surface is determined by taking the average of each coefficient in the two hyperplanes.

2.4 Pruning

When a decision tree is constructed, it is normally biased by the training data set, which is the set of points that is used for finding the splits at the internal nodes of the tree.

However, if the intended use of the decision tree is that of a classifier, the accuracy obtained on the training set is of no value. It is the classification accuracy achieved on points not included in the training data set – referred to as the test data set – that make all the difference. To cancel the bias towards the training set and thus produce decision trees that generalize well on test data, the decision tree is usually pruned. Pruning is the process of limiting the size of a decision tree to improve its classification accuracy on the test data set. This can be achieved either by establishing stopping criteria that would limit the growth of a decision tree and prevent over-fitting to the training set (pre-pruning) or by collapsing some of the branches of a fully-grown decision tree into leaves (post-pruning). While pre-pruning has the added advantage of simplicity, it is inherently a shortsighted approach. Post-pruning is the preferred and more common form of pruning because it avoids a major problem with pre-pruning known as the “horizon effect” [Frank 2000] where the stopping criteria can prune a branch, which if left un-pruned would lead to a sub-tree that would not be pruned according to the same criteria. Post-pruning is usually referred to as simply pruning because it is more common and because it represents a better fit to the horticulture analogy.

There are many pruning methods in the field of decision tree construction. In what follows, we provide a quick overview of some of the better known methods. Esposito, Malerba, and Semeraro [1997] provide a good comparative analysis of these methods.

2.4.1 Reduced Error Pruning

Proposed by Quinlan [1987], this method is one of the simplest pruning algorithms. The concept is to hold out part of the training set for pruning – this set is usually referred to as

the pruning set. Branches are then recursively pruned until further pruning increases the misclassification rate on the pruning set.

2.4.2 Pessimistic Error Pruning

Again, this method was proposed by Quinlan [1987]. Unlike Reduced Error pruning, this method does not require a separate pruning set. The error rate on the training set is adjusted to obtain a more realistic estimate of the actual error rate – the error rate on instances not used in growing the decision tree. A sub-tree is pruned into a leaf if the adjusted error rate estimate of the sub-tree is higher than that of the leaf.

2.4.3 Minimum Error Pruning

This method was proposed by Niblett and Bratko [1986]. Like Pessimistic Error Pruning, it relies on an error estimate that is derived from the training set. However, it differs in the way it adjusts the error rate to make it more realistic.

2.4.4 Critical Value Pruning

This is a simple bottom-up pruning method proposed by Mingers [1987]. Pruning decisions are based on information about the goodness of a split – for example the Gini index or the information gain – that is collected during the phase of growing a decision tree. At each node, the goodness of the split is compared to a threshold value called the critical value. An internal node is pruned if the goodness measure of its split is below the critical value and all its child nodes are leafs. It is the second constraint that differentiates this method from a pre-pruning method where the stopping criterion is the threshold constraint.

2.4.5 Cost-Complexity Pruning

Breiman et al. [1984] introduced this pruning method in CART. Cost-Complexity defines α as the average increase in error per leaf of a subtree when the subtree is pruned based on the training data set. A family of progressively smaller subtrees $(T_0, T_1, T_2, \dots, T_L)$ is constructed where T_{i+1} is obtained by pruning all nodes in T_i with the lowest value of α . T_0 is obtained by pruning branches with $\alpha=0$. The best subtree in the parametric family is chosen based on its predictive accuracy, which in turn is estimated based on an independent pruning set.

2.4.6 Error-Based Pruning

Quinlan [1993] implemented this method in C4.5. This method is similar to Pessimistic Error Pruning in that it derives error estimates from the training set. A confidence interval is estimated based on a binomial error distribution. The error rate corresponding to the upper limit of the confidence interval – set to 25% by default – is used as the node's estimated error rate.

Towards Better Decision Trees

3.1 Decision Tree Construction by Classical Classifiers

In constructing a decision tree, a classifier starts with a root node that represents the entire training set. The dataset at the root node is subdivided into multiple datasets each represented by an internal node. The process of subdivision is then recursively repeated on all internal nodes until a stopping criterion is met – at which point the subsets are represented by leaves or leaf-nodes. A final step of pruning back the resulting tree is commonly done.

There are three major decisions that the designer of any classifier has to make:

- The choice of a method or algorithm to subdivide a dataset at internal nodes.
- The choice of a criterion to stop the recursive subdivision.
- The choice of a method or algorithm to prune the resulting tree back to a smaller tree that is less biased by the training set.

With classical classifiers, these decisions are made at design time. The algorithms themselves are refined over the years to provide a good common method for various types of datasets but at runtime these algorithms are fixed and they do not adapt to the dataset being processed.

3.2 Problem Statement

This thesis proposes that an improvement in classification accuracy can be achieved if the subdivision algorithm is dynamically chosen while the decision tree is being constructed. This is in contrast to the common practice of choosing a fixed subdivision algorithm for a classifier and using it on all the nodes of all the decision trees that are constructed by the classifier. In particular, the thesis will study the effect of changing the subdivision algorithm adaptively during the phase of growing a decision tree as dictated by the dataset at hand and in accordance to a predefined policy, which specifies what changes are to be made and under what conditions. The quantifiable measure for comparing the two approaches is the overall accuracy of the resulting decision tree on a test dataset.

A survey of existing classifiers showed that the idea of changing the subdivision algorithm as the decision tree is being constructed has not been tried before.

Adaptive Decision Tree Construction

As mentioned in the previous chapter, this thesis studies the effect of adaptively changing the subdivision algorithm while a decision tree is being constructed.

For a classifier to be able to change its subdivision algorithm at runtime it needs a set of rules and instructions that define the possible algorithms to consider and how to choose what is believed to be the best subdivision algorithm at a particular node. We call such set of rules and instructions a *policy*.

This chapter describes the policies that were implemented to adaptively change the subdivision method while the tree is being constructed.

4.1 Definitions

The following definitions are used throughout this chapter (see section 2.3.1 for detailed definitions):

- The *total* accuracy is defined as the total number of correctly classified points divided by the total number of points in a dataset.
- The *population* accuracy of type n is defined as the number of points of type n that are correctly classified divided by the total number of type n points in the dataset.

- The *zone* accuracy of type n is defined as the number of points that are correctly classified to be of type n divided by the total number of points that are classified to be of type n .
- A dataset is said to contain a *minority* class c if the number of points belonging to class c is less than a small fraction of the total number of points in the dataset. We use 10% as the defining fraction in this research.
- A *basic* subdivision method is one where no correction or bias is applied while placing the separating surface. A classically constructed decision tree uses the same basic subdivision method at all of its internal nodes.

4.2 Adaptive Tree Construction Policies

A total of nine different policies for adaptive decision tree construction were implemented. This section describes these policies using pseudo code. Graphical illustrations are also provided based on hypothetical two-dimensional datasets and using Chinneck's surface placement algorithm [Chinneck 2003] to make the subdivisions. The illustrations show the unpruned decision trees constructed using various policies.

The main goal of showing these figures is to demonstrate that the policies have a drastic impact on the shape of the resulting decision tree. It may not be apparent in the figures which policy is best for a given dataset. However, the fact that changing the policy results in a noticeably different decision trees and the fact that these policies represent a systematic way of changing the subdivision method at each node of the decision tree being

constructed suggest that well crafted policies can produce better decision trees that are adapted to the dataset under consideration and thus have higher classification accuracy.

4.2.1 Policy 1: Highest Total Accuracy

In this policy, all subdivision methods are tried at each node. The method that produces the highest total accuracy is used. Algorithm 4.1 provides a pseudo code description of this policy.

```

bestAccuracy = 0

for each subdivision method  $m$  in all methods

    use method  $m$  to temporarily split the data points at
    the current node into two classes

    if there were no software failures

        currentAccuracy = total accuracy corresponding to
        the split by method  $m$ 

        if currentAccuracy > bestAccuracy

            bestAccuracy = currentAccuracy

            bestMethod =  $m$ 

    use bestMethod to make the permanent split

```

Algorithm 4.1: Pseudo code of the steps taken at each node when policy 1 is used to construct the decision tree.

The default behaviour or basic method of a separating-surface placement algorithm could be to maximize the total accuracy at each split (by using Chinneck's algorithm for example). However, the basic method may fail to provide a separation surface at some nodes as the decision tree is being constructed. In a non-adaptively constructed decision

tree such node would not be subdivided any further while in an adaptively constructed decision tree using policy 1, if the basic method fails to generate a split, other methods are likely to succeed.

In other words, a non-adaptively constructed decision tree is more likely to have its growth stopped prematurely than a decision tree that is adaptively constructed using policy 1. As mentioned in chapter 2, it is generally accepted that a fully-grown and then pruned-back decision tree provides better classification accuracy than a decision tree that was never fully-grown.

The above-mentioned advantage is of particular importance as it applies to all policies and not just policy 1. In general, by virtue of trying a set of subdivision methods at each node, a tree constructed using any adaptive policy has a better chance of reaching its full size than a tree constructed using just one subdivision method does.

Figure 4 shows a decision tree constructed on a hypothetical dataset using policy 1. From the figure it is clear that the subdivisions favour maximizing the total classification accuracy over capturing the boundaries of the underlying dataset. This greedy approach can prove advantageous in small decision trees as would be the case for over-pruned trees.

Figure 4: A decision tree constructed using policy 1

4.2.2 Policy 2: Method Dependent Ranking

Various subdivision methods that allow the user to bias the resulting split accuracy can be divided into two categories.

- Category 1: Methods that provide control over the population accuracy (e.g. the method that balances the population accuracies using Chinneck's algorithm)
- Category 2: Methods that provide control over the zone accuracy (e.g. the method that balances the zone accuracies using Chinneck's algorithm).

In policy 2, all subdivision methods are tried at each node. For a subdivision method of category 1, the population accuracy is used as the measure of merit. For a subdivision method of category 2, the zone accuracy is used as the measure of merit. The method with the highest measure of merit is selected. Algorithm 4.2 provides a pseudo code description of this policy.

```

bestAccuracy = 0

for each subdivision method  $m$  in all methods

    use method  $m$  to temporarily split the data points at
    the current node into two classes

    if there were no software failures

        if method  $m$  provides control over zone accuracy

            currentAccuracy = highest of the two zone
            accuracies corresponding to the split by method
             $m$ 

        else

            currentAccuracy = highest of the two population
            accuracies corresponding to the split by method
             $m$ 

    if currentAccuracy > bestAccuracy

        bestAccuracy = currentAccuracy

        bestMethod =  $m$ 

use bestMethod to make the permanent split

```

Algorithm 4.2: Pseudo code of the steps taken at each node when policy 2 is used to construct the decision tree

The idea behind this policy is that if the measure of merit of a split depends directly on the type of subdivision method that produced the split, then the resulting decision tree is, more likely than not, a better representation of the underlying dataset.

Figure 5 shows a decision tree constructed using policy 2 on the same dataset as in Figure 4. Note that in policy 1, the measure of merit of a split is not based on the type of subdivision method used. As can be seen in the figure, using this policy allowed the decision tree to reach 100% purity on all of its leaf nodes. Provided that the classifier is equipped with a good pruning algorithm, it is generally accepted that fully growing a decision tree before pruning it back is the preferred practice that would increase the classification accuracy.

Figure 5: A decision tree constructed using policy 2

4.2.3 Policy 3: Method Dependent Ranking with Minority Protection

This is the same as policy 2 except that when the dataset contains a minority class then the method that maximizes the population accuracy of that class is used regardless of the method category or of other population or zone accuracies. Algorithm 4.3 provides a pseudo code description of this policy.

```
bestAccuracy = 0
```

```
for each subdivision method  $m$  in all methods
```

```

use method  $m$  to temporarily split the data points at
the current node into two classes

if there were no software failures

    if class "A" is a minority class at the current
    node

        currentAccuracy = population accuracy of class
        "A" corresponding to the split by method  $m$ 

    else if class "B" is a minority class at the
    current node

        currentAccuracy = population accuracy of class
        "B" corresponding to the split by method  $m$ 

    else if method  $m$  provides control over zone
    accuracy

        currentAccuracy = highest of the two zone
        accuracies corresponding to the split by method
         $m$ 

    else

        currentAccuracy = highest of the two population
        accuracies corresponding to the split by method
         $m$ 

    if currentAccuracy > bestAccuracy

        bestAccuracy = currentAccuracy

        bestMethod =  $m$ 

use bestMethod to make the permanent split

```

Algorithm 4.3: Pseudo code of the steps taken at each node when policy 3 is used to construct the decision tree

It is not uncommon for a decision tree to poorly reflect the underlying dataset when one of the classes has a much lower cardinality than the other. Even if only a few points of a

minority class are misclassified, the borders of that class can quickly become blurred. This is because those few misclassified points may in fact be a considerable portion of that class.

This policy detects the above situation and attempts to remedy it by giving a higher priority to reducing classification errors on a minority class.

Figure 6 compares a decision tree constructed using policy 3 (bottom) to one constructed using policy 2 (top) on a hypothetical dataset where one of the classes is a minority class. The figure demonstrates how policy 3 generates a decision tree that captures the boundaries of a minority class much better than a decision tree generated by another policy.

Figure 6: Protecting a minority class with policy 3. Policy 2 (top) vs. policy 3 (bottom).

4.2.4 Policy 4: Method Dependent Ranking with Mixed-Zone Detection

This policy is also a variation on policy 2. It introduces the following stopping criterion: If the base method (no control over population or zone accuracies) fails to adequately subdivide the dataset at a given node then convert that node to a leaf thus stopping it from being subdivided. The leaf is given the label of the class with the higher cardinality. Ties are arbitrary broken in favour of type 0. In this research we set the threshold for adequate subdivision at 50% classification accuracy of a binary decision tree on the training set. Algorithm 4.4 provides a pseudo code description of this policy.

```

bestAccuracy = 0

use the basic method mb to temporarily split the data
points at the current node

if there was an error or the total accuracy < 0.5 then
stop (this converts the current node to a leaf whose
label is that of the class with higher cardinality)

for each subdivision method m in all methods

    use method m to temporarily split the data points at
    the current node into two classes

    if there were no software failures

        if m provides control over zone accuracy

            currentAccuracy = highest of the two zone
            accuracies corresponding to the split by method
            m

        else

```

```

    currentAccuracy = highest of the two population
    accuracies corresponding to the split by method
    m

    if currentAccuracy > bestAccuracy

        bestAccuracy = currentAccuracy

        bestMethod = m

    use bestMethod to make the permanent split

```

Algorithm 4.4: Pseudo code of the steps taken at each node when policy 4 is used to construct the decision tree.

An over-fitted decision tree is one that fits the training set so closely that it fails to represent the underlying data set in a useful and generic way. Over-fitting occurs due to anomalies in the training dataset or due to regions of highly mixed and thus inseparable instances of the two classes.

The job of eliminating the effect of over-fitting is usually left for the pruning algorithm. Occasionally, however, a situation may arise where the pruning algorithm cannot fully correct an over-fitted decision tree. This usually happens when the over-fitting is due to a large and highly mixed region in the dataset. In such a situation, it is more beneficial to avoid over-fitting before it occurs than to rely completely on the pruning algorithm for correction.

Figure 7 compares a decision tree constructed using policy 4 (bottom) versus policy 2 on a hypothetical dataset with a highly mixed region. As seen in the figure, policy 4 stops the subdivision of a highly mixed node.

Figure 7: Avoiding over-fitting with policy 4. Policy 2 (top) vs. policy 4 (bottom).

4.2.5 Policy 5: Lowest Gini Index

In this policy, all subdivision methods are tried at each node. The method that produces the lowest Gini index at that node is used. Algorithm 4.5 provides a pseudo code description of this policy.

```

bestGini = 1

for each subdivision method  $m$  in all methods

    use method  $m$  to temporarily split the data points at
    the current node into two classes

    if there were no software failures

         $g$  = the Gini index corresponding to the split by
        method  $m$ 

        if  $g < \text{bestGini}$ 

            bestGini =  $g$ 

            bestMethod =  $m$ 

    use bestMethod to make the permanent split

```

Algorithm 4.5: Pseudo code of the steps taken at each node when policy 5 is used to construct the decision tree

This policy introduces the Gini index from CART [Breiman et al. 1984] as the measure of merit for choosing the best subdivision method at a given node.

Figure 8 shows a decision tree constructed on the dataset of Figure 4 using policy 5. This policy captured the boundaries of the dataset quite nicely which suggests that this policy is a good fit for this particular dataset.

Figure 8: A decision tree constructed using Policy 5

4.2.6 Policy 6: Highest Information Gain Ratio

In this policy, all subdivision methods are tried at each node. The method that produces the highest Information Gain Ratio at that node is used. Algorithm 4.6 provides a pseudo code description of this policy.

```
bestInfoGainRatio = 0
```

```
for each subdivision method  $m$  in all methods
```

```
    use method  $m$  to temporarily split the data points at  
    the current node into two classes
```

```

if there were no software failures

    info = the information gain ratio corresponding to
    the split by method  $m$ 

    if info > bestInfoGainRatio

        bestInfoGainRatio = info

        bestMethod =  $m$ 

use bestMethod to make the permanent split

```

Algorithm 4.6: Pseudo code of the steps taken at each node when policy 6 is used to construct the decision tree

This policy introduces the Information Gain Ratio from C4.5 [Quinlan 1993] as the measure of merit for choosing the best subdivision method at a given node.

Figure 9 shows a decision tree constructed on the dataset of Figure 4 using policy 6. Operating on the same dataset, policies 5 and 6 produced very different decision trees. Notice that the only difference between the two policies is the measure of merit they use to evaluate the various splits at a given node.

Figure 9: A decision tree constructed using Policy 6

4.2.7 Policy 7: Lowest Pessimistic Estimated Error

In this policy, all subdivision methods are tried at each node. For each method, an estimate of the classification error rate is calculated using the formula of the Pessimistic Error Pruning algorithm. The method with the lowest pessimistic estimated error rate is used. Algorithm 4.7 provides a pseudo code description of this policy.

```
bestAccuracy = 0
```

```
for each subdivision method  $m$  in all methods
```

```

use method  $m$  to temporarily split the data points at
the current node into two classes

if there were no software failures

     $e$  = pessimistic error rate estimate corresponding
    to the split by method  $m$ 

    currentAccuracy =  $1 - e$ 

    if currentAccuracy > bestAccuracy

        bestAccuracy = currentAccuracy

        bestMethod =  $m$ 

use bestMethod to make the permanent split

```

Algorithm 4.7: Pseudo code of the steps taken at each node when policy 7 is used to construct the decision tree

This policy borrows the estimated error rate from the Pessimistic Error Pruning algorithm as a measure of merit in choosing the permanent split at a node. As it prunes a decision tree, the Pessimistic Error Pruning algorithm collapses an internal node into a leaf if the estimated error rate of the leaf is lower than that of the node. While constructing a decision tree, policy 7 chooses the split that produces the lowest estimated error rate at each node.

Figure 10 shows a decision tree constructed using policy 7 on a hypothetical dataset.

Figure 10: A decision tree constructed using Policy 7

4.2.8 Policy 8: Lowest Zone Gini Index

In this policy, all subdivision methods are tried at each node. The Gini index is then calculated for each zone. The method that produces the lowest Gini index at any zone is used. Algorithm 4.8 provides a pseudo code description of this policy.

```
bestGini = 1
```

```
for each subdivision method  $m$  in all methods
```

```
    use method  $m$  to temporarily split the data points at  
    the current node into two classes
```

```

if there were no software failures

    gA = the Gini index of zone "A" corresponding to
    the split by method  $m$ 

    gB = the Gini index of zone "B" corresponding to
    the split by method  $m$ 

    g = Minimum(gA, gB)

    if g < bestGini

        bestGini = g

        bestMethod =  $m$ 

use bestMethod to make the permanent split

```

Algorithm 4.8: Pseudo code of the steps taken at each node when policy 8 is used to construct the decision tree

Because the Gini index is lowest for a zone that is composed entirely of one class, this method is good at cutting off pure zones as the decision tree is constructed.

Figure 11 shows a decision tree constructed using policy 8 on a hypothetical dataset. Except for one misclassified point, the decision tree in this example can be considered to be fully grown.

Figure 11: A decision tree constructed using Policy 8

4.2.9 Policy 0: Best of Policies 1-8

This final policy is special in that it is the best of policies 1-8 for a given dataset. The idea behind this policy is that it is unlikely for one policy to outperform, in terms of classification accuracy, all other policies on all datasets. Hence, trying all policies and picking the most suitable one for a given dataset produces a more accurate decision tree. While all the previous policies operate on the node level and are used to pick the most suitable subdivision method at that node, policy 0 operate on a dataset level and is used to pick the most suitable decision tree for that dataset.

Algorithm 4.9 describes policy 0.

```
bestAccuracy = 0

for each policy p in policies 1 to 8

    use p to construct a decision tree t on the given
    dataset

    classificationAccuracy = classification accuracy of t
    on a test dataset

    if classificationAccuracy > bestAccuracy

        bestAccuracy = classificationAccuracy

        bestTree = t

output t as the classification decision tree to be used
with the given dataset.
```

Algorithm 4.9: Pseudo code describing how policy 0 generates a decision tree for a given dataset.

Experimental Setup

Two classifiers, one adaptive and one non-adaptive, were implemented to evaluate the advantage of using adaptive policies in constructing decision trees. The adaptive classifier uses the policies described in the previous chapter while the non-adaptive classifier uses the defaults of the subdivision algorithm as its fixed method for splitting dataset subsets at all internal nodes of the decision tree. The two classifiers are otherwise identical.

A set of classification runs was performed on publicly available datasets using each of the two classifiers and the various policies in the adaptive case. In addition, the same set of experiments was performed using a well-known commercial classifier to provide a baseline for our experimental results. These results are reported and compared in the next chapter.

The remainder of this chapter describes in detail the setup for the above-mentioned experiments; it includes implementation decisions that can affect the resulting accuracy such as the subdivision algorithm and the pruning algorithm as well as the choice of the datasets used.

5.1 Subdivision Algorithms

Chinneck's [2003] separating surface placement via the maximum feasible subsystem heuristic is used to subdivide the datasets. The implementation of the heuristic provides a set of algorithms or methods that the calling routine can select from by varying a few parameters. This feature makes it easy to change the subdivision algorithm at runtime. Any

subdivision algorithm (even axis-parallel methods) can be used as long as its interface provides a degree of control over the placement of the separating surface. Table 3 gives a summary of the methods provided by Chinneck's algorithm - for a detailed discussion of this algorithm and its surface placement methods please refer to chapter 2.

Table 3: Chinneck's algorithm method summary

N°	Method	Category	Threshold
0	basic	population-accuracy control	n/a
1	Balance the population fractions	population-accuracy control	0.1
2	Balance the population accuracies (version 1)	population-accuracy control	n/a
3	Balance the population accuracies (version 2)	population-accuracy control	n/a
4	Balance the population accuracies (version 3)	population-accuracy control	n/a
5	Balance the zone accuracies (version 1)	zone-accuracy control	n/a
6	Balance the zone accuracies (version 2)	zone-accuracy control	n/a
7	Balance the zone accuracies (version 3)	zone-accuracy control	n/a
8	Favour the type 0 zone accuracy (version 1)	zone-accuracy control	0.9
9	Favour the type 1 zone accuracy (version 1)	zone-accuracy control	0.9
10	Favour the type 0 zone accuracy (version 2)	zone-accuracy control	0.9
11	Favour the type 1 zone accuracy (version 2)	zone-accuracy control	0.9
12	Prevent degenerate hyperplane placement	population-accuracy control	n/a

Our adaptive classifier relies on its predefined policies to choose one of the subdivision methods listed in Table 3 at each internal node of the decision tree it is constructing. Where applicable a fixed threshold is chosen and tied to a method. When deciding on a threshold value we had two requirements in mind. The first requirement is to retain the special characteristics of each method; coupling methods 8-11 with too low thresholds or method 1 with too high threshold values will render these methods indistinguishable from the basic method on many datasets. The second requirement is allow for the diversification of methods used in constructing a decision tree; setting the threshold level of methods 8-11 to the maximum value (i.e. 1.0) or the threshold of method 1 to its

minimum value (i.e. 0.0) will cause these methods to fail on most datasets or to produce splits that over-fit the training dataset, which would end-up getting pruned in most decision trees. We executed several trials and concluded that the threshold values in table 3 satisfy our requirements.

As noted in chapter 2, the degree of control provided by Chinneck's algorithm extend beyond the choice the subdivision method to be used. Specifically, the algorithm allows for choosing one of three heuristics and for specifying one of three secondary objectives. Our early experiments showed that changing these two parameters had little or no effect on the overall classification accuracy of the final decision trees produced. We chose the maximum-product heuristic because it is the fastest and we specify the minimization of the total slack of the removed points as our secondary objective because some classifiers, such as OC1, use this as their primary objective. It is worth pointing out here that the accuracies reported in Chinneck's publications [1998, 2001, and 2003] refer to the ability of a single separating surface to split the training dataset in a meaningful way while the accuracies of interest in this research are those representing the ability of a complete decision tree to classify data points that were not used in its construction (i.e. the test dataset).

5.2 Stopping Criterion

A stopping criterion that would prevent the decision tree from reaching its full extent constitutes a form of pruning referred to as pre-pruning. As mentioned in chapter 2, the literature advises against this form of pruning as it could cut short the development of branches whose presence may be advantageous in describing the underlying class structure of the dataset or to improve the classification accuracy.

In all experiments, with the exception of those involving Policy 4, the decision trees were fully grown. In other words, the instances at the leaf-nodes belonged to the same class or, for a given experiment, none of the subdivision algorithms used in that experiment were able to place a separation surface between them.

5.3 Pruning Algorithm

In their comparative analysis, Esposito, Malerba, and Semeraro [1997] demonstrated that none of the six pruning algorithms they looked at consistently outperforms all other algorithms on all datasets.

We implemented the three best pruning algorithms out of the six that Esposito et al analyzed: Reduced Error Pruning (REP), Pessimistic Error Pruning (PEP), and Error Based Pruning (EBP). Esposito et al did not rank the algorithms they compared. However, one of their experimental results is a table titled: “Average Error Rates of Trees Obtained with Different Pruning Methods” that compares the pruning algorithms on fifteen datasets. We used the above-mentioned table to help us shorten the list of pruning algorithms to implement.

We chose to use the Pessimistic Error Pruning algorithm in our experiments because it produced better results more often than the other two algorithms.

5.4 Accuracy Calculation

In all the experiments, the accuracy is defined to be the average overall classification accuracy, calculated from a ten-fold cross-validation [Witten and Frank 1999] with 20% of the points reserved for testing.

In our implementation, each dataset is randomly divided into ten subsets (folds) within one point difference in cardinality. Two consecutive folds (e.g. 1 and 2) are reserved for testing while the rest are used for training and pruning. The process is then repeated with the next pair of test folds (e.g. 2 and 3) until all ten combinations are processed.

5.5 Software Failures

Depending on the dataset and the subdivision method requested, the subdivision algorithm sometimes fails to produce a separating hyperplane. These software failures are specific to the subdivision algorithm. The behaviour of our classifiers is as follows:

- If the error occurs at the root node, then the experiment is halted and a 0 classification accuracy is reported
- If, on the other hand, the error occurred on any internal node other than the root node, that node is converted to a leaf that bares the label of the class with higher cardinality

5.6 Non-Adaptive and Baseline Classifiers

The objective of our experiments is to compare the classification accuracy of the adaptive classifier against that of the non-adaptive classifier. The non-adaptive classifier in our experiments uses the *basic* method as its fixed subdivision method. The basic method is the default in Chinneck's algorithm and it provides the highest classification accuracy in the fixed-subdivision-method (i.e. non-adaptive) classifier. The classifier used as a base line is a commercial version of a well-know axis-parallel decision-tree classifier. All of the parameters of the baseline classifier were left at default values.

5.7 UCI Datasets

The University of California Irvine machine-learning repository [2004] contains a collection of datasets that can be used to evaluate classifiers and learning algorithms. The datasets in the repository are contributed by researchers around the world and, with few exceptions, are publicly available for download. The UCI machine-learning dataset repository is referenced in many publications in the field of classification. There is a total of 113 datasets in the repository encompassing a much diversified set of domains including medicine, science, engineering, and many more. All the datasets were downloaded from the ftp site of the repository (<ftp://ftp.ics.uci.edu/pub/machine-learning-databases/>). Some of the datasets are not suitable for our experiments and thus are excluded.

5.7.1 Datasets Excluded

A dataset is excluded from the experiments if one or more of the following criteria apply to it. Most of the exclusions are unavoidable for example when the access to the dataset is restricted or when the class is continuous. A few datasets were excluded for practical reasons as in the case eliminating a dataset because it takes too long to finish a classification run or when there are too many attributes to allow for easy massaging of the dataset. However, only seven datasets could in theory have been included but were, instead, excluded for practical reason. Out of the 113 datasets, these constitute only a small percentage and their exclusion is not expected to cause any bias in the results.

Restricted:

In the repository there are three datasets that are not publicly available.

Categorical:

If all or most of the attributes in the dataset were categorical, the database would not be suitable for our experiments because the separating algorithm requires numerical attributes.

It is possible to convert categorical to numerical attributes by splitting the categorical column into n numerical columns where n is the total number of distinct categories. However, this is only done if the dataset contains one or two categorical attributes, each with few distinct categories. The splitting is done by placing a 1 in the column that matches a given categorical value and 0 in the rest. The following example illustrates splitting a categorical column into 3 numerical columns:

Table 4: Converting attribute values from categorical to numerical.

Categorical	Numerical		
Color	Color Red	Color Green	Color Blue
green	0	1	0
red	1	0	0
green	0	1	0
blue	0	0	1

Twenty-seven datasets were excluded because all or most of their attributes were categorical.

Dataset too small:

Some of the datasets from the UCI repository are relatively small in size. As the dataset size become smaller so does the size of the test dataset. Accuracies calculated from a small test dataset tend to have low statistical confidence and as such are not suited for

comparative analysis. Datasets with less than 200 points are excluded. Nine databases are excluded due to this criterion.

Dataset too big:

Four databases were excluded because our classifiers took too long to construct decision trees on them.

Continuous class:

Decision trees are used to classify datasets with discrete classes. Three datasets are excluded because they have continuous classes.

Too many missing attribute values:

The separating-surface placement algorithm requires that all the attributes of the data points be present. Therefore, any instance with missing attributes has to be removed from the dataset before the latter can be input to the classifier. If a significant portion of the dataset has to be removed due to missing attributes, then the remaining dataset may end up too small and thus is excluded. One dataset is excluded due to this reason.

Too many attributes:

Each dataset is processed using a spreadsheet application to eliminate rows with missing attributes and, where feasible, to convert categorical attributes to numerical values. If a dataset has too many attributes to fit into a spreadsheet it is excluded for practical reasons. Three datasets are excluded because they have more than 256 attributes.

Non-standard datasets:

Not all the downloadable material from the UCI repository adheres to the conventional definition of a dataset, which is a set of points or instances that can be represented in a tabular form. Examples of materials in the UCI repository that were not usable as a dataset for our experiment include rule-based representations of certain domains, programs, and various textual descriptions that cannot be readily put in the tabular form that our classifier requires for its input. Twenty-five datasets are excluded because they are not given in tabular format.

Classifier error:

One dataset was excluded because it caused error on all the policies of the adaptive classifier as well as the non-adaptive classifier.

5.7.2 Datasets used in the experiments

Table 5 below lists all the datasets that are used in our experiments with relevant information about each. The “net pts” column contains the total number of data point after cleanup, which is the removal of records with missing attribute values.

Table 5: A list of the datasets used in the experiments

	dataset	net pts	type 0 pts / category	type 1 pts / category	attributes	Discrete attributes
1	abalone (9 v/s others)	4177	3488 / others	689 / 9	8	1
2	allbp (negative v/s others)	2643	2515 / negative	128 / others	21	15
3	allhyper (negative v/s others)	2643	2568 / negative	75 / others	21	15
4	allhypo (negative v/s others)	2643	2427 / negative	216 / others	21	15
5	allrep (negative v/s others)	2643	2568 / negative	75 / others	21	15
6	ann-thyroid (3 v/s others)	7200	6666 / 3	534 / others	21	15
7	balance-scale (B v/s others)	625	576 / others	49 / balanced	4	0
8	breast-cancer-wisconsin	683	444 / benign	239 / malignant	9	0
9	bupa	341	142 / 1	199 / 2	6	0
10	cmc (1 v/s others)	1473	629 / 1	844 / others	8	3
11	dermatology (1 v/s others)	358	247 / others	111 / 1	34	0
12	dis (negative v/s others)	2643	2596 / negative	47 / others	21	15
13	ecoli (cp v/s others)	336	193 / others	143 / CP	7	0
14	glass	214	163 / window	51 / non-window	9	0
15	haberman	282	203 / 1	79 / 2	3	0
16	hd_hungarian (0 v/s others)	261	163 / 0	98 / others	10	6
17	heart	270	150 / 1	120 / 2	12	3
18	house-votes-84	232	124 / republican	108 / democrat	16	16
19	hypothyroid	2000	1878 / negative	122 / positive	18	12
20	ionosphere	351	126 / bad	225 / good	33	0
21	letter-recognition (U v/s V+W)	2329	1516 / V or W	813 / U	16	0
22	monks-1	556	278 / 0	278 / 1	6	0
23	monks-2	601	395 / 0	206 / 1	6	0
24	monks-3	554	266 / 0	288 / 1	6	0
25	new-thyroid (normal v/s other)	215	150 / normal	65 / others	5	0
26	optdigits (3 v/s others)	5620	5048 / others	572 / 3	64	0
27	page-blocks (test v/s others)	5473	4913 / others	560 / text	10	0
28	pima-indians-diabetes	768	500 / 0	268 / 1	8	0
29	sat (2 v/s others)	6435	5732 / others	703 / 2	36	0
30	segmentation (path v/s others)	2310	1980 / others	330 / path	18	0
31	sick (negative v/s others)	2643	2431 / negative	212 / others	21	15
32	sick-cuthyroid (negative v/s others)	2000	1762 / negative	238 / others	18	12
33	spambase	4601	2788 / not spam	1813 / spam	57	0
34	tic-tac-toe	958	332 / negative	626 / positive	9	9
35	vehicle (bus v/s others)	846	628 / others	218 / bus	18	0
36	wdbc	569	357 / benign	212 / malignant	30	0
37	yeast (CYT v/s others)	1484	1021 / others	463 / CYT	8	0

Chapter 6

Results and comparisons

6.1 Results

Table 6 summarizes the results of all the experiments carried out in this research. It lists the total classification accuracy of an adaptive classifier (column headings “Policy 1” ... “Policy 8”) against those of the non-adaptive classifier (column heading “None”). The adaptive classifier uses the eight policies described in chapter 4 while the non-adaptive uses a fixed method of subdivision, which in our case is the basic or default method. In addition to the results of the two classifiers implemented during this research, table 6 also lists the classification accuracy of the commercially available classifier (column heading “Comm.”) executed on the same datasets with its default settings. The commercial software results were included to provide a reference point as we compare our results.

Blank cells in table 6 indicate software failures during execution; a software failure on at least one fold in the 10-fold cross validation runs would be considered as a failure of the entire experiment. This shows one of the advantages that adaptive classifiers have over non-adaptive classifiers. Within one policy, an adaptive classifier is less likely to fail due to the fact that it tries several methods of subdivision at each internal node of a decision tree thus having a better chance of finding a separation surface than would a classifier that relies on only one method for its subdivisions. Across many policies, adaptive classifiers

have an even bigger advantage. The standard deviation of the 10-fold cross validation results is also included in the table.

For a given dataset, the highest observed classification accuracy for the adaptive and the non-adaptive classifiers is shown in boldface.

Table 6: Experimental Results

Dataset		None	Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6	Policy 7	Policy 8	Comm.
abalone (9 v/s others)	Avg		68.88%	73.74%	73.74%		72.41%	72.54%	68.88%	72.37%	58.63%
	SD		1.62%	1.04%	1.04%		1.97%	1.83%	1.62%	1.11%	
allbp (negative v/s others)	Avg	96.39%	93.93%	94.27%	96.06%	95.72%	94.51%	93.21%	93.93%	94.32%	95.23%
	SD	1.32%	1.22%	1.30%	0.94%	1.23%	0.96%	1.26%	1.22%	1.38%	
allhyper (negative v/s others)	Avg	96.88%	96.86%	96.80%	97.79%	97.29%	97.14%	96.97%	96.86%	97.14%	98.41%
	SD	0.69%	0.54%	1.08%	0.80%	0.68%	1.06%	1.22%	0.54%	1.20%	
allhypo (negative v/s others)	Avg	98.01%	97.88%	98.07%	98.39%	98.17%	97.92%	98.20%	97.88%	98.49%	99.32%
	SD	0.35%	0.34%	0.56%	0.67%	0.54%	0.49%	0.49%	0.34%	0.72%	
allrep (negative v/s others)	Avg	97.65%	98.03%	98.09%	98.52%	98.07%	98.15%	97.71%	98.03%	97.81%	99.39%
	SD	0.51%	0.87%	1.07%	0.39%	0.97%	0.83%	1.11%	0.87%	1.13%	
ann-thyroid (3 v/s others)	Avg	98.40%	98.51%	98.35%	98.58%	98.28%	98.54%	98.14%	98.51%	98.34%	99.40%
	SD	0.44%	0.34%	0.46%	0.26%	0.44%	0.24%	0.59%	0.34%	0.47%	
balance-scale (B v/s others)	Avg		91.84%	91.68%	93.03%		92.56%	91.44%	91.84%	92.00%	63.36%
	SD		1.68%	2.61%	2.64%		2.79%	1.61%	1.68%	0.96%	
breast-cancer-wisconsin	Avg	95.90%	95.31%	95.02%	95.53%	95.75%	95.31%	96.12%	95.31%	95.53%	95.90%
	SD	2.05%	1.55%	1.65%	2.12%	1.58%	1.67%	2.05%	1.55%	2.06%	
bupa	Avg	71.10%	63.93%	65.82%	65.68%	66.25%	64.65%	60.55%	63.93%	65.53%	68.04%
	SD	6.68%	5.77%	7.51%	7.39%	6.74%	5.45%	5.57%	5.77%	6.34%	
cmc (1 v/s others)	Avg	68.43%	67.28%	66.60%	66.33%	67.11%	65.34%	65.92%	67.28%	65.04%	70.94%
	SD	2.58%	2.07%	2.08%	1.95%	1.77%	2.82%	2.16%	2.07%	2.14%	
dermatology (1 v/s others)	Avg	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	99.16%
	SD	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	
dis (negative v/s others)	Avg		96.10%	96.69%	97.60%	97.31%	97.31%	96.50%	96.10%	96.24%	97.58%
	SD		1.40%	0.55%	0.65%	0.95%	0.95%	1.27%	1.40%	1.04%	
ecoli (cp v/s others)	Avg	95.97%	95.97%	94.80%	94.79%	95.39%	95.68%	94.49%	95.97%	93.90%	94.05%
	SD	2.15%	2.15%	2.10%	1.45%	2.02%	1.66%	2.32%	2.15%	2.02%	
glass	Avg	89.98%	90.45%	91.87%	91.64%	91.87%	90.92%	91.87%	90.45%	91.87%	93.93%
	SD	4.93%	4.98%	3.88%	3.55%	3.88%	3.07%	3.88%	4.98%	3.88%	
haberman	Avg		68.78%	68.98%	68.98%		68.26%	67.90%	68.78%	68.44%	69.15%
	SD		5.81%	6.75%	6.75%		5.57%	6.44%	5.81%	3.07%	
hd_hungarian (0 v/s others)	Avg	70.71%	71.68%	78.16%	77.96%	76.62%	71.48%	72.24%	71.68%	78.35%	78.93%
	SD	5.00%	5.76%	4.26%	4.07%	3.85%	6.30%	6.68%	5.76%	4.43%	

Dataset		None	Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6	Policy 7	Policy 8	Comm.
heart	Avg	68.70%	68.33%	70.74%	69.07%	69.81%	69.26%	70.00%	68.33%	71.30%	80.00%
	SD	6.62%	7.73%	7.45%	9.12%	6.88%	6.25%	7.19%	7.73%	7.16%	
house-votes-84	Avg	91.81%	91.81%	91.81%	91.81%	92.67%	91.81%	91.81%	91.81%	91.81%	96.98%
	SD	4.40%	4.40%	4.40%	4.40%	4.36%	4.40%	4.40%	4.40%	4.40%	
hypothyroid	Avg	97.48%	96.45%	97.10%	97.83%	96.90%	97.23%	96.95%	96.45%	97.58%	98.05%
	SD	0.77%	1.07%	1.01%	0.62%	0.89%	1.54%	0.90%	1.07%	1.01%	
ionosphere	Avg	82.49%	81.36%	84.78%	84.78%	84.93%	83.63%	84.78%	81.36%	84.78%	90.03%
	SD	5.72%	6.36%	8.81%	8.81%	9.57%	6.57%	8.81%	6.36%	8.81%	
letter-recognition (U v/s V+W)	Avg	98.73%	98.75%	98.63%	98.71%	98.60%	98.71%	98.65%	98.75%	98.63%	98.41%
	SD	0.59%	0.61%	0.52%	0.69%	0.46%	0.60%	0.53%	0.61%	0.52%	
monks-1	Avg	84.45%	95.69%	98.66%	98.66%	85.34%	99.28%	98.75%	95.69%	99.01%	88.67%
	SD	6.63%	3.48%	1.81%	1.81%	7.21%	1.69%	1.65%	3.48%	1.43%	
monks-2	Avg		80.11%	83.69%	82.61%		82.78%	82.11%	80.11%	83.27%	95.84%
	SD		3.55%	2.67%	3.47%		2.65%	3.68%	3.55%	3.68%	
monks-3	Avg	88.82%	96.66%	97.20%	97.29%	88.82%	97.48%	97.11%	96.66%	97.11%	98.92%
	SD	3.44%	2.56%	2.30%	2.33%	3.44%	1.98%	2.32%	2.56%	2.32%	
new-thyroid (normal v/s other)	Avg	91.86%	92.09%	91.40%	91.86%	91.63%	92.32%	91.63%	92.09%	91.86%	91.63%
	SD	4.79%	5.03%	4.37%	3.81%	4.53%	4.77%	4.11%	5.03%	3.84%	
optdigits (3 v/s others)	Avg	97.72%	97.80%	97.88%	97.80%	97.81%	97.84%	97.88%	97.80%	97.88%	97.62%
	SD	0.50%	0.39%	0.35%	0.40%	0.38%	0.38%	0.35%	0.39%	0.35%	
page-blocks (1 v/s others)	Avg	96.73%	96.56%	96.57%	96.29%	96.40%	96.53%	96.41%	96.56%	96.72%	96.36%
	SD	0.37%	0.63%	0.53%	0.51%	0.70%	0.57%	0.72%	0.63%	0.55%	
pima-indians-diabetes	Avg	75.79%	67.70%	68.42%	68.81%	70.77%	70.25%	65.10%	67.70%	68.23%	73.83%
	SD	4.27%	4.22%	3.32%	3.30%	2.56%	3.03%	2.87%	4.22%	4.79%	
sat (2 v/s others)	Avg	98.18%	98.18%	98.24%	98.15%	98.45%	98.29%	98.30%	98.18%	98.26%	98.94%
	SD	0.39%	0.39%	0.32%	0.24%	0.34%	0.38%	0.36%	0.39%	0.33%	
segmentation (path v/s others)	Avg	99.46%	99.46%	99.46%	99.46%	99.50%	99.46%	99.46%	99.46%	99.46%	99.74%
	SD	0.37%	0.37%	0.37%	0.37%	0.31%	0.37%	0.37%	0.37%	0.37%	
sick (negative v/s others)	Avg	94.18%	93.63%	93.98%	94.36%	93.34%	94.12%	93.11%	93.63%	93.87%	97.16%
	SD	1.88%	0.94%	1.51%	1.18%	1.05%	1.09%	1.41%	0.94%	1.21%	
sick-euthyroid (negative v/s others)	Avg	95.58%	91.33%	92.68%	93.73%	93.58%	92.83%	92.03%	91.33%	92.93%	96.05%
	SD	0.74%	1.15%	0.82%	0.85%	1.59%	0.75%	1.85%	1.15%	1.63%	
spambase	Avg	89.19%	89.83%	91.63%	91.64%	91.91%	90.14%	89.75%	89.83%	91.94%	92.28%
	SD	1.69%	1.29%	0.64%	0.62%	0.78%	1.46%	1.48%	1.29%	0.78%	
tic-tac-toe	Avg	96.30%	96.66%	96.40%	97.03%	96.61%	95.83%	96.40%	96.66%	96.40%	93.00%
	SD	1.42%	1.50%	1.26%	0.55%	0.52%	1.78%	1.26%	1.50%	1.26%	
vehicle (bus v/s others)	Avg	95.03%	95.33%	95.92%	95.68%	95.68%	94.50%	95.92%	95.33%	95.92%	95.27%
	SD	1.45%	1.41%	0.90%	0.97%	1.65%	1.55%	0.90%	1.41%	0.90%	
wdbc	Avg	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.15%
	SD	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	
yeast (CYT v/s others)	Avg		67.28%	69.78%	70.59%		67.96%	69.17%	67.28%	70.05%	69.47%
	SD		1.58%	2.57%	3.45%		1.77%	2.02%	1.58%	2.10%	

6.2 Comparisons

This section uses performance profile [Dolan and Moré 2002] plots to first compare the adaptive classifier classification accuracies versus those of the non-adaptive classifier and then to compare the classification accuracies for each individual adaptive policy with those of the non-adaptive classifier. The plots are spread across more than one chart for clarity. All charts include a performance profile plot of the commercial classifier classification accuracy to establish an external reference.

6.2.1 Performance Profiles

Dolan and Moré use the term *performance profile* for “the [cumulative] distribution function of a performance metric”. In our case, the performance metric is the ratio of the total classification accuracy of a given classifier on a certain dataset to the highest observed classification accuracy on the same dataset. If we define $A_{c,d}$ to be the classification accuracy of classifier c on dataset d , then the performance metric of classifier c on dataset d is:

$$r_{d,c} = \frac{A_{c,d}}{\max\{a_{c,d} : c \in C\}}$$

and the performance profile is:

$$\rho_c(\tau) = \frac{1}{n_d} \text{size}\{d \in D : r_{d,c} \leq \tau\}$$

Where C is the set of classifiers that we used; this set includes the adaptive classifier, the non-adaptive classifier, and the commercial version of the baseline classifier. D is the set of

datasets used in our experiments, n_d is the number of datasets in D and τ is a real number between 0 and 1.

In other words, the performance profile of a classifier c is the probability that the classification accuracy of c is within a factor τ of the best observed accuracy. For example, in Figure 12, the probability of having the accuracy of the non-adaptive classifier to be within 90% of the accuracy of the adaptive classifier using policy 0 is approximately 0.8. If a certain classifier had the highest accuracy in all cases, it would show a single point at the upper left corner of the graph. In general, curves that are closest to the upper left corner indicate superior results.

6.2.2 Adaptive Versus Non-Adaptive

Figure 12 shows the performance profile of the adaptive classifier (running policy 0) versus that of the non-adaptive classifier. Recall that policy 0 gives the maximum classification accuracy from the eight different policies.

Figure 12: Adaptive Policy 0 versus Non-Adaptive Performance Profiles

Figure 12 demonstrates that adaptive change or modification of the subdivision method used by a classifier as it constructs a decision tree can significantly improve the classification accuracy of that classifier.

The performance profile of the adaptive classifier is better than that of the non-adaptive classifier for all values of τ .

Figure 12 also shows that the classification accuracy of the adaptive classifier implemented in this research is comparable to that of a commercially available classifier. The performance profiles of the two classifiers are close for high values of τ . For values of τ

less than about 95%, the performance profile of the adaptive classifier is actually better than that of the commercial classifier. The plots also show that, for the datasets used in the experiments, the adaptive classifier is guaranteed to have a classification accuracy of around 85% or higher of the best observed accuracies while the classification accuracy of the decision trees produced by the commercial classifier is only guaranteed to be 65% or higher of the observed classification accuracies.

6.2.3 Individual Policies Versus Non-Adaptive

Figures 13, 14, and 15 below show the performance profiles of individual policies compared to the performance profile of the non-adaptive classifier.

The plots demonstrate that even when used individually, the adaptive profiles usually provide an improvement in classification accuracy over non-adaptive decision tree construction. The performance profiles of all policies are better than that of the non-adaptive classifier.

Figure 13: Performance Profiles of Adaptive Policies 1, 2, and 3

Figure 14: Performance Profiles of Adaptive Policies 4, 5, and 6

6.2.4 Execution time

While the commercial classifier took anywhere between 1 and 10 minutes to construct a decision tree on any of the datasets, the range for the execution time of our classifiers was much larger. The time required by our classifier to construct a decision tree ranged from few minutes to 6 hours for the non-adaptive classifier and up to 30 hours for the adaptive classifier. The analysis of the execution time of the classifiers is outside the scope of this research.

While fast construction of decision trees is always a desirable feature, it is not an essential one considering that, once constructed, a decision tree can be used for years as long as the

dataset used to construct it is still representative of the data points that it is classifying. Obtaining a better decision tree may very well be worth running a classifier over-night.

Figure 15: Performance Profiles of Adaptive Policies 7 and 8

6.2.5 Performance profiles with software failures eliminated

Part of the improvement in the classification accuracy of the adaptive classifier is due to its robustness. The performance profiles in this section are based on the datasets that did not cause software failures in any method. Table 7 below shows only the models in which

every method succeeded. Software failures were observed in 6 out of the 37 models used for the experiments, which leaves table 7 with 31 models.

Table 7: Experimental results with software failures eliminated

Dataset		None	Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6	Policy 7	Policy 8	Comm.
allbp (negative v/s others)	Avg	96.39%	93.93%	94.27%	96.06%	95.72%	94.51%	93.21%	93.93%	94.32%	95.23%
	SD	1.32%	1.22%	1.30%	0.94%	1.23%	0.96%	1.26%	1.22%	1.38%	
allhyper (negative v/s others)	Avg	96.88%	96.86%	96.80%	97.79%	97.29%	97.14%	96.97%	96.86%	97.14%	98.41%
	SD	0.69%	0.54%	1.08%	0.80%	0.68%	1.06%	1.22%	0.54%	1.20%	
allhypo (negative v/s others)	Avg	98.01%	97.88%	98.07%	98.39%	98.17%	97.92%	98.20%	97.88%	98.49%	99.32%
	SD	0.35%	0.34%	0.56%	0.67%	0.54%	0.49%	0.49%	0.34%	0.72%	
allrep (negative v/s others)	Avg	97.65%	98.03%	98.09%	98.52%	98.07%	98.15%	97.71%	98.03%	97.81%	99.39%
	SD	0.51%	0.87%	1.07%	0.39%	0.97%	0.83%	1.11%	0.87%	1.13%	
ann-thyroid (3 v/s others)	Avg	98.40%	98.51%	98.35%	98.58%	98.28%	98.54%	98.14%	98.51%	98.34%	99.40%
	SD	0.44%	0.34%	0.46%	0.26%	0.44%	0.24%	0.59%	0.34%	0.47%	
breast-cancer-wisconsin	Avg	95.90%	95.31%	95.02%	95.53%	95.75%	95.31%	96.12%	95.31%	95.53%	95.90%
	SD	2.05%	1.55%	1.65%	2.12%	1.58%	1.67%	2.05%	1.55%	2.06%	
bupa	Avg	71.10%	63.93%	65.82%	65.68%	66.25%	64.65%	60.55%	63.93%	65.53%	68.04%
	SD	6.68%	5.77%	7.51%	7.39%	6.74%	5.45%	5.57%	5.77%	6.34%	
cmc (1 v/s others)	Avg	68.43%	67.28%	66.60%	66.33%	67.11%	65.34%	65.92%	67.28%	65.04%	70.94%
	SD	2.58%	2.07%	2.08%	1.95%	1.77%	2.82%	2.16%	2.07%	2.14%	
dermatology (1 v/s others)	Avg	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	98.88%	99.16%
	SD	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	1.10%	
ecoli (cp v/s others)	Avg	95.97%	95.97%	94.80%	94.79%	95.39%	95.68%	94.49%	95.97%	93.90%	94.05%
	SD	2.15%	2.15%	2.10%	1.45%	2.02%	1.66%	2.32%	2.15%	2.02%	
glass	Avg	89.98%	90.45%	91.87%	91.64%	91.87%	90.92%	91.87%	90.45%	91.87%	93.93%
	SD	4.93%	4.98%	3.88%	3.55%	3.88%	3.07%	3.88%	4.98%	3.88%	
hd_hungarian (0 v/s others)	Avg	70.71%	71.68%	78.16%	77.96%	76.62%	71.48%	72.24%	71.68%	78.35%	78.93%
	SD	5.00%	5.76%	4.26%	4.07%	3.85%	6.30%	6.68%	5.76%	4.43%	
heart	Avg	68.70%	68.33%	70.74%	69.07%	69.81%	69.26%	70.00%	68.33%	71.30%	80.00%
	SD	6.62%	7.73%	7.45%	9.12%	6.88%	6.25%	7.19%	7.73%	7.16%	
house-votes-84	Avg	91.81%	91.81%	91.81%	91.81%	92.67%	91.81%	91.81%	91.81%	91.81%	96.98%
	SD	4.40%	4.40%	4.40%	4.40%	4.36%	4.40%	4.40%	4.40%	4.40%	
hypothyroid	Avg	97.48%	96.45%	97.10%	97.83%	96.90%	97.23%	96.95%	96.45%	97.58%	98.05%
	SD	0.77%	1.07%	1.01%	0.62%	0.89%	1.54%	0.90%	1.07%	1.01%	
ionosphere	Avg	82.49%	81.36%	84.78%	84.78%	84.93%	83.63%	84.78%	81.36%	84.78%	90.03%
	SD	5.72%	6.36%	8.81%	8.81%	9.57%	6.57%	8.81%	6.36%	8.81%	
letter-recognition (U v/s V+W)	Avg	98.73%	98.75%	98.63%	98.71%	98.60%	98.71%	98.65%	98.75%	98.63%	98.41%
	SD	0.59%	0.61%	0.52%	0.69%	0.46%	0.60%	0.53%	0.61%	0.52%	

Dataset		None	Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6	Policy 7	Policy 8	Comm.
monks-1	Avg	84.45%	95.69%	98.66%	98.66%	85.34%	99.28%	98.75%	95.69%	99.01%	88.67%
	SD	6.63%	3.48%	1.81%	1.81%	7.21%	1.69%	1.65%	3.48%	1.43%	
monks-3	Avg	88.82%	96.66%	97.20%	97.29%	88.82%	97.48%	97.11%	96.66%	97.11%	98.92%
	SD	3.44%	2.56%	2.30%	2.33%	3.44%	1.98%	2.32%	2.56%	2.32%	
new-thyroid (normal v/s other)	Avg	91.86%	92.09%	91.40%	91.86%	91.63%	92.32%	91.63%	92.09%	91.86%	91.63%
	SD	4.79%	5.03%	4.37%	3.81%	4.53%	4.77%	4.11%	5.03%	3.84%	
optdigits (3 v/s others)	Avg	97.72%	97.80%	97.88%	97.80%	97.81%	97.84%	97.88%	97.80%	97.88%	97.62%
	SD	0.50%	0.39%	0.35%	0.40%	0.38%	0.38%	0.35%	0.39%	0.35%	
page-blocks (1 v/s others)	Avg	96.73%	96.56%	96.57%	96.29%	96.40%	96.53%	96.41%	96.56%	96.72%	96.36%
	SD	0.37%	0.63%	0.53%	0.51%	0.70%	0.57%	0.72%	0.63%	0.55%	
pima-indians-diabetes	Avg	75.79%	67.70%	68.42%	68.81%	70.77%	70.25%	65.10%	67.70%	68.23%	73.83%
	SD	4.27%	4.22%	3.32%	3.30%	2.56%	3.03%	2.87%	4.22%	4.79%	
sat (2 v/s others)	Avg	98.18%	98.18%	98.24%	98.15%	98.45%	98.29%	98.30%	98.18%	98.26%	98.94%
	SD	0.39%	0.39%	0.32%	0.24%	0.34%	0.38%	0.36%	0.39%	0.33%	
segmentation (path v/s others)	Avg	99.46%	99.46%	99.46%	99.46%	99.50%	99.46%	99.46%	99.46%	99.46%	99.74%
	SD	0.37%	0.37%	0.37%	0.37%	0.31%	0.37%	0.37%	0.37%	0.37%	
sick (negative v/s others)	Avg	94.18%	93.63%	93.98%	94.36%	93.34%	94.12%	93.11%	93.63%	93.87%	97.16%
	SD	1.88%	0.94%	1.51%	1.18%	1.05%	1.09%	1.41%	0.94%	1.21%	
sick-euthyroid (negative v/s others)	Avg	95.58%	91.33%	92.68%	93.73%	93.58%	92.83%	92.03%	91.33%	92.93%	96.05%
	SD	0.74%	1.15%	0.82%	0.85%	1.59%	0.75%	1.85%	1.15%	1.63%	
spambase	Avg	89.19%	89.83%	91.63%	91.64%	91.91%	90.14%	89.75%	89.83%	91.94%	92.28%
	SD	1.69%	1.29%	0.64%	0.62%	0.78%	1.46%	1.48%	1.29%	0.78%	
tic-tac-toe	Avg	96.30%	96.66%	96.40%	97.03%	96.61%	95.83%	96.40%	96.66%	96.40%	93.00%
	SD	1.42%	1.50%	1.26%	0.55%	0.52%	1.78%	1.26%	1.50%	1.26%	
vehicle (bus v/s others)	Avg	95.03%	95.33%	95.92%	95.68%	95.68%	94.50%	95.92%	95.33%	95.92%	95.27%
	SD	1.45%	1.41%	0.90%	0.97%	1.65%	1.55%	0.90%	1.41%	0.90%	
wdbc	Avg	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.59%	93.15%
	SD	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	1.65%	

As expected, eliminating the effect of robustness on the classification accuracy narrows the gap between the adaptive and the non-adaptive performance profiles. Figure 16 below shows that while the performance profile of adaptive policy 0 is still superior to that of the non-adaptive classifier, the difference is less pronounced than it is in the plots where robustness plays a role in improving the classification accuracy. The performance profiles in figure 16 show that, in the reduced set of experiments, the classification accuracy of the adaptive policy 0 was within 10% of the best observed accuracy on each of the datasets,

the classification accuracy of the non-adaptive classifier is within 15% of the best observed accuracy, and that of the commercial classifier is now within 10% of the best observed accuracy. The significant improvement in the performance profile of the commercial classifier suggests that the commercial classifier exhibits its poorest performance on the eliminated datasets.

Figure 16: Adaptive Policy 0 vs. Non-Adaptive Performance Profiles with the effect of robustness eliminated.

Figures 17, 18, and 19 below show the performance profiles of the individual adaptive policies. Eliminating the datasets that cause software failures on the non-adaptive classifier

resulted in performance profiles that very close to each other but adaptive policies still provide an improvement in classification accuracy for the classifier that uses them.

Figure 17: Performance Profiles of Adaptive Policies 1, 2, and 3 with the effect of robustness eliminated

Figure 18: Performance Profiles of Adaptive Policies 4, 5, and 6 with the effect of robustness eliminated

Figure 19: Performance Profiles of Adaptive Policies 7 and 8 with the effect of robustness eliminated

Conclusions

This thesis investigates an approach for improving the classification accuracy of decision trees by adaptively varying the method of splitting internal nodes while the decision tree is constructed. This is in contrast to the approach of current classifiers where a fixed method is used throughout the construction phase of their decision trees. We demonstrate through empirical experimental results that this approach does in fact lead to a significant improvement in the classification accuracies of the produced decision trees. More specifically, we observed in chapter 6 that:

- An adaptive classifier using policy 0 produces much better decision trees than a similar but non-adaptive classifier.
- The decision trees produced by the adaptive classifier are comparable in terms of classification accuracy with those generated by a commercial implementation of one of the best available classifiers.

Our investigation relied on the availability of a segregation algorithm that provided enough control over its behaviour to allow effective variations on the subdivision method.

As mentioned in the introduction, the main contributions of this work are:

- Developing a set of algorithms that can be used by a classifier to adaptively change the subdivision method at each node of the decision tree it is constructing
- Demonstrating that using these policies result in an improvement of the classification accuracy of the produced decision trees.

7.1 Research directions

In this thesis we put forth the idea of changing the subdivision method to adapt to varying dataset subsets as a decision tree is constructed. There are many research topics that can be pursued following this one. Chief among these is developing more sophisticated tree construction algorithms (a.k.a. policies) that use some form of analysis of the data at a node to choose the best subdivision method to apply at that node. At the classifier level, the possibility exists for devising a way of picking the most suitable tree construction policy (or a small set of policies) for a given dataset without having to try all the available policies. This may, for example, depend on the number of discrete attributes or some other characteristic of the dataset.

Pruning is an important phase in the construction of a decision tree. Experimenting with as many as possible of the existing pruning algorithms or even developing a new pruning algorithm may prove to be a worthwhile endeavor as it is expected to result in further improvement in the classification accuracy of the adaptive classifier.

On the practical aspects of the research, it is recommended that some effort go into improving the execution speed of the classifier. This makes it easier to do more experiments and to include larger and more complex datasets. Moreover, it is worthwhile

investing some time in building customized utilities for dataset pre-processing and for automating the experiments.

Bibliography

- K. Bennett and O. Mangasarian. Robust Linear Programming Discrimination of Two Linearly Inseparable Sets. *Optimization Methods and Software*, vol. 1, pp 23-34 1994.
- K. Bennett and E. Bredensteiner. A Parametric Optimization Method for Machine Learning. *INFORMS Journal on Computing*, vol. 9:3, pp. 311-318 1997.
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.
- J. Chinneck. Improved Linear Classification via LP Infeasibility Analysis. Technical Report, Department of Systems and Computer Engineering, Carleton University, 1998.
- J. Chinneck. Fast heuristics for the Maximum Feasible Subsystem Problem. *INFORMS Journal on Computing*, vol. 13:3, pp. 210-223, 2001.
- J. Chinneck. Controlling Classifier Accuracy via a Maximum Feasible Subsystem Heuristic. *Systems and Computer Engineering*, Carleton University, 2003.
- E. Dolan and J. Moré. Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming – Series A*, vol. 91, pp. 201-213, 2002.
- R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

F. Esposito, D. Malerba, and G. Semeraro. A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 467-491, May 1997.

E. Frank. Pruning Decision Trees and Lists, PhD dissertation. University of Waikato, January 2000.

D. Hand. *Discrimination and Classification*. Wiley, 1981.

I. Markham, R. Mathieu, and B. Wray. A Comparison of Neural Networks and Decision Trees for the Kanban Setting Problem. *Proceedings of the DSI 1998 annual meeting* - November 21-24 1998, Las Vegas, Nevada.

J. Mingers. Expert Systems - Rule Induction With Statistical Data, *Journal of the Operations Research Society*, vol. 38, no. 1, pp. 39-47, 1987.

S. Murthy, S. Kasif, and S. Salzberg. A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*, vol. 2, pp. 1-33, 1994.

S. Murthy. On Growing Better Decision Trees from Data. PhD dissertation. John Hopkins University - Baltimore, Maryland, 1997.

T. Niblett and I. Bratko. Learning Decision Rules in Noisy Domains, *Proceedings of Expert Systems*, pp. 25-34, 1986. Cambridge University Press.

J. Quinlan. Induction of Decision Trees. *Machine Learning*, vol. 1, pp. 81-106, 1986.

J. Quinlan. Simplifying Decision Trees, International Journal of Man-Machine Studies, vol. 27, pp. 221-234, 1987.

J. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers 1993.

University of California at Irvine, UCI Machine Learning Repository,

<http://www.ics.uci.edu/~mlearn/MLRepository.html>, 2004

I. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers 1999.

K. Xu. How Has the Literature on Gini's Index Evolved in the Past 80 Years? Department of Economics Dalhousie University Halifax, Nova Scotia. April 2003.