Recursive Automatic Algorithm Selection for Inductive Learning

Carla E. Brodley
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003 USA
brodley@cs.umass.edu

COINS Technical Report 94-61
August 1994

RECURSIVE AUTOMATIC ALGORITHM SELECTION

FOR INDUCTIVE LEARNING

A Dissertation Presented

by

CARLA E. BRODLEY

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY
September 1994
Department of Computer Science

RECURSIVE AUTOMATIC ALGORITHM SELECTION

FOR INDUCTIVE LEARNING

A Dissertation Presented

by

CARLA E. BRODLEY

Approved as to style and content by:

Paul E. Utgoff, Chair

Andrew G. Barto, Member

Wendy Lehnert, Member

Victor R. Lesser, Member

Michael Sutherland, Member

W. Richards Adrion, Department Head
Department of Computer Science

This dissertation is dedicated to Antony Lloyd Hosking

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Paul Utgoff for his time, energy and humor in teaching me how to conduct research. His high standards in research, writing and professional conduct have made an excellent role model. I will miss our research discussions and our marathon Hearts games.

I thank the members of my thesis committee, Andy Barto, Wendy Lehnert and Victor Lesser for their insightful comments and suggestions. Mike Sutherland's statistical expertise was a great resource throughout my time here.

The machine learning lab has been a great environment for exploring ideas. I thank Jeff Clouse for always being available and interested in discussing new ideas with me, and for his friendship and support over the last five years. Sharad Saxena, Jamie Callan, Tom Fawcett, Margie Connell and Neil Berkman have each provided me with valuable feedback and interesting discussions over the years.

I thank Paula Magdich for suggesting that I take my first computer course. She and Mark Friedl have been a constant source of encouragement both at McGill and in graduate school. I would like to thank Malini Bhandaru, Carol Broverman, Adele Howe, Marty Humphrey, Susan Lander, and Dorothy Mammen for their friendship and advice throughout my years in graduate school.

I thank Emily Alston-Fallonsbee for always being there when I needed to express joy or doubt. My parents I thank for providing an environment in which pursuing my goals was made easy. Together with my sister they have been a unflagging source of encouragement and support that instilled in me the confidence to realize my dreams. Finally and most importantly, I thank my husband Tony Hosking for his support, encouragement, and his ability to help me see situations in a calm and rational fashion. His belief in me made finishing my degree a reality.

This research was supported by the National Science Foundation under Grant No. IRI-9222766, and by the National Aeronautics and Space Administration under Grant No. NCC 2-658.

ABSTRACT

RECURSIVE AUTOMATIC ALGORITHM SELECTION
FOR INDUCTIVE LEARNING
SEPTEMBER 1994
CARLA E. BRODLEY
B.A., MCGILL UNIVERSITY
M.S., UNIVERSITY OF MASSACHUSETTS AMHERST
PH.D., UNIVERSITY OF MASSACHUSETTS AMHERST
Directed by: Professor Paul E. Utgoff

The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a selective superiority; each is best for some but not all tasks. Selective superiority arises because each learning algorithm searches within a restricted generalization space, defined by its representation language, and employs a search bias for selecting a generalization in that space. Given a data set, it is often not clear beforehand which algorithm will yield the best performance. The problem is complicated further because for some learning tasks, different subtasks are learned best using different algorithms. In such cases, the ability to form a hybrid classifier that combines different representation languages will produce a more accurate classifier than employing a single representation language and search bias.

This dissertation presents an approach to overcoming this problem by applying knowledge about the biases of a set of learning algorithms to conduct a recursive automatic algorithm search. The approach permits classifiers learned by the available algorithms to be mixed in a recursive tree-structured hybrid, thereby allowing different subproblems of the learning task to be learned by different algorithms. The Model Class Selection System (MCS), an implementation of the approach, combines decision trees, linear discriminant functions and instance-based classifiers in a tree-structured hybrid classifier. Heuristic knowledge about the characteristics that indicate one bias is better than another is encoded in the rule base that guides MCS's search for the best classifier. An empirical evaluation illustrates that MCS achieves classification accuracies equal to or higher than the best of its primitive learning components for a variety of data sets, demonstrating that domain-independent knowledge about the biases of machine learning algorithms can guide an automatic algorithm selection search.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A fundamental problem in machine learning is how to form a generalization from a set of examples. Given a set of examples, each described by a set of features and labelled with a class name, the goal of a classifier construction algorithm is to form a generalization of the examples that can be used to classify previously unobserved objects with a high degree of accuracy.

Several dozen classifier construction algorithms have been developed in the last few decades, including various versions of perceptron, DNF cover, version space, decision tree, instance-based, and neural-net algorithms. The results of empirical comparisons of existing algorithms illustrate that each algorithm has a *selective superiority*; it is best for some but not all tasks. Selective superiority arises because each learning algorithm searches within a restricted generalization space, defined by its representation language, and employs a search bias for selecting a generalization in that space.

Given a data set, it is often not clear beforehand which algorithm will yield the best performance. In such situations, someone wanting to find a classifier for the data will be confused by the myriad choices, and will need to try many of them in order to be satisfied that a better classifier will not be found easily. The problem is complicated further because for some learning tasks, different subtasks are learned best using different algorithms. In such cases, the ability to form a hybrid classifier that combines different representation languages will produce a more accurate classifier than employing a single representation language and search bias.

In this dissertation we present an approach to overcoming this problem by applying knowledge about the biases of a set of learning algorithms to conduct a recursive automatic algorithm search. The approach permits classifiers learned by the available algorithms to be mixed in a recursive tree-structured hybrid, thereby allowing different subproblems of the learning task to be learned by different algorithms. In the remainder of this chapter we describe the selective superiority problem and outline its causes in more detail, we introduce our approach to recursive automatic algorithm selection, and we present an overview of the results of this dissertation.

1.1 The Problem of Selective Superiority

Several dozen inductive learning algorithms have been developed over the last few decades. In every case, the algorithm can boast one or more superior learning performances over the others, but none is always better. The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a selective superiority [Weiss & Kapouleas, 1989; Aha, Kibler & Albert, 1991b; Shavlik, Mooney & Towell, 1991; Salzberg, 1991]. This is because each algorithm is biased, leading to a good fit for some learning tasks, and a poor fit for others. Given a learning task, the maximum achievable accuracy depends on the quality of the data and on the ability of the learning algorithm to form the most accurate classifier for that data set. In this dissertation, we assume that the input representation and the data are given, and we focus instead on how learning is affected by the choice of learning algorithm.

An algorithm's success in finding a good generalization for a given data set depends on two factors. The first is whether the algorithm's representation space contains a good generalization. In statistics, this space is called the algorithm's *model class*. For example, the model class of a symbolic, univariate decision tree algorithm is the set of all possible hyper-rectangular regions that are separated by decision boundaries orthogonal to the feature axes. The model class of a perceptron learning algorithm is the class of linear discriminant functions. Because an algorithm's model class defines the space of possible generalizations, not even an exhaustive search strategy can overcome a poor choice of model class. The second factor of an algorithm's success is its search bias. Even algorithms that search the same model space have shown selective superiority due to different search methods. For example, if a set of instances is not linearly separable then the Least Mean Squares (LMS) training rule provides a better solution than the Absolute Error Correction rule (AECR) for learning the weights of a linear discriminant function [Duda & Hart, 1973]. The situation is reversed when the instances are linearly separable; LMS is not guaranteed to find a separating hyperplane, whereas AECR is.

In many cases, the choice of learning algorithm can be made on subject matter considerations. However, when such prior knowledge is not available, one needs a method for determining the learning algorithm that will produce the most accurate classifier for instances in the domain. Traditionally, this task has fallen on the shoulders of the data analyst. In this dissertation we present an approach to automatic algorithm selection, designed to solve the selective superiority problem.

The creation of systems that search multiple representation spaces increases the autonomy of learning machines. Ultimately, a human will no longer be required to select a learning algorithm. This increase in autonomy will make machine learning techniques accessible to a wider range of scientists. They will no longer need to understand the representational biases of the available learning algorithms, because this step in the analysis of data will now be automated.

1.2 Homogeneous versus Hybrid Classifiers

One can characterize inductive learning algorithms as searching either a *homogeneous* or *hybrid* hypothesis space. A homogeneous hypothesis space is one that contains a single representation language. For example, univariate decision trees and linear discriminant functions are each single representation languages. A hybrid hypothesis space is a one that combines different homogeneous hypothesis spaces; each hypothesis is expressed using one or more representation languages. For example, a representation language that mixes linear discriminant functions and univariate decision trees in a hypothesis defines a hybrid hypothesis space.

Selecting a learning algorithm that employs a homogeneous representation bias assumes that a classifier for the data set is best represented using a single representation language. For some data sets this may not be the case; different subspaces of the learning task may be best learned using different representation biases, indicating that forming a hybrid classifier will result in higher classification accuracy than a homogeneous classifier. Indeed, one goal of this research is to demonstrate that for some data sets, different subproblems are best learned using different learning algorithms.

Constructing a hybrid classifier requires a method for partitioning the data into useful subspaces and a method for choosing the best learning algorithm for each subspace. Therefore, the problem of automatically selecting the best algorithm for a set of data needs to be addressed when creating a hybrid classifier construction algorithm.

1.3 Automatic Algorithm Selection

Selecting the best algorithm for a data set in the absence of prior knowledge is a search problem. One well-known approach from statistics is to use cross-validation [Linhart & Zucchini, 1986], which performs an exhaustive search through the space of candidate methods. However, as the number of alternatives increases, the time required to search for the best algorithm may become impractical due to the computational expense of performing a cross-validation. One of the oldest methods in Artificial Intelligence for reducing search effort is to use knowledge about the problem domain.

Our objective is to solve the selective superiority problem through automatic algorithm selection. Here the problem domain is selection of the best learning algorithm for a given set of data. The search space is defined by the data set and the set of candidate algorithms. In this domain, knowledge stems from understanding the biases of the candidate algorithms and how this can be applied to guide the search for the best algorithm/model class for a given set of data.

1.4 Recursive Automatic Algorithm Selection

In this dissertation we study the problem of how to select the best learning bias for a data set, and we investigate the utility of combining different representations and search biases in a hybrid tree-structured classifier. Our approach recursively selects an algorithm (a representation and search bias), from a set of candidates, with which to construct each node of a hybrid tree-structured classifier. To select an algorithm our approach iteratively fits a classifier to the data using the representation and search bias currently considered best for the data set. Next it computes measures of how well the resulting classifier fits the data. The measures are used to decide whether the best classifier has been found or whether to search further, and if so which bias to try next.

The ability to perform an effective search relies on knowledge of how to recognize whether and why an algorithm is a poor choice, and on using this information to select a better one. We have encoded this knowledge into a set of heuristic rules that work together to guide a search for a best representation language and search bias. Our approach is based on the following hypothesis: *Domain independent knowledge about data characteristics in the form of feedback from learning can effectively guide an automatic algorithm selection search.*

Our approach to automatic algorithm selection is applied recursively, providing a mechanism for mixing the available model classes to form a hybrid classifier. The ability to choose a single model class is not lost, but the approach also permits hybrid classifiers. The hypothesis space of hybrid classifiers is larger than the hypothesis space of homogeneous classifiers defined by the union of the homogeneous representation languages contained in the hybrid space. By increasing the space of possible hypotheses we increase the probability that for a given data set, a good generalization will exist in the space searched. Increasing the search space does not ensure that a good generalization will be found; it depends on the search algorithm. Therefore, a second hypothesis of this research is: *Our knowledge-based search strategy for finding a hybrid classifier will produce a classifier that is never worse than, and for some data sets is better than, any homogeneous classifier produced by its primitive components.*

We have implemented the approach in a system named the Model Class Selection system (MCS). MCS combines three commonly used representations into a recursive tree-structured hybrid classifier. Given a data set, MCS selects the most appropriate of the three model classes, decision trees, linear discriminant functions and instance-based classifiers by applying a hill-climbing search guided by a set of heuristic rules. Our choice of these three representation languages stems from the results of empirical comparisons, which illustrate that these three languages produce classifiers of widely differing accuracy for different data sets; each of the three was selectively superior for some data sets.

For each model class, MCS can choose among one or more search biases for fitting a classifier to the data. After choosing a model class and search bias with which to

construct a classifier, MCS partitions the data set using the classifier and applies the search recursively to each resulting subset of the data that contains instances from more than one class. Once a classifier has been fit that classifies each instance in the training data correctly, MCS applies a heuristic pruning procedure to ensure that the resulting classifier does not overfit the training data at the expense of generalization.

1.5 Overview of the Results

In this dissertation we explore the advantages of a recursive heuristic approach to hybrid classifier construction. We examine the utility of applying knowledge about the biases of learning algorithms to guide a search, which uses feedback from the learning process. We demonstrate, both analytically and experimentally that some data sets are best learned by a hybrid classifier construction algorithm. Specifically, we demonstrate the following:

1. Hybrid classifiers subsume the homogeneous primitive algorithms that they combine. We provide both empirical and analytical evidence. The analytical evidence illustrates how the space of decision boundaries that can be represented by the hybrids increases the ability to select an accurate classifier for a larger set of learning tasks over selecting among the homogeneous primitive algorithms. Our empirical results illustrate that the classifiers produced by MCS are never less accurate than any of the primitive algorithms, and for some data sets are more accurate.

2. Domain independent knowledge about the biases of machine learning algorithms can guide the search for a best learning bias. The results of an empirical comparison illustrate that MCS is never less, and is sometimes more accurate, than the best of its primitive learning algorithms, demonstrating that MCS is a robust automatic algorithm selection system. Our evaluation of the heuristic rule set, we use data sets that were not used in rule development, demonstrating that general domain independent knowledge exists and can effectively guide an automatic algorithm selection system.

3. Using knowledge increases accuracy and reduces time over methods that form hybrid classifiers by exhaustively trying all possible algorithms at each node and selecting among them using one source of feedback. Indeed, the results of an empirical comparison demonstrate that these "knowledge-poor" methods perform worse than the best of the primitive algorithms for some data sets. In contrast, our knowledge-based approach performs equal to or better than each of the primitive algorithms, and equal to or better than the other hybrid methods.

4. An analysis of the decision boundaries that each model class can represent, of the decision boundaries that their combination into a hybrid can represent, and what types of boundaries the hybrid cannot represent.

In the past few years many researchers have studied the importance of learning bias to the success of finding an accurate classifier for a set of data. A multitude of empirical comparisons have illustrated that selective superiority is a problem that needs solving. In this dissertation, we illustrate that a recursive knowledge-based automatic algorithm selection system can subsume its primitive homogeneous learning algorithms, thereby solving the selective superiority problem for these algorithms. In addition, we demonstrate that for some data sets different subspaces require different learning biases, illustrating that hybrid classifiers perform better than homogeneous classifiers for these data sets.

1.6 Guide to the Dissertation

The remainder of this dissertation is organized as follows. In **Chapter 2** we discuss the selective superiority problem in more detail and describe existing approaches to automatic algorithm selection. In addition, we describe how knowledge about the biases of machine learning algorithms can guide a heuristic search for the best representation and search bias for a given set of data. In **Chapter 3** we discuss the issues that must be considered when designing a hybrid classifier construction algorithm, we describe existing hybrid classifier construction algorithms, and we present our recursive approach to combining different representations in a hybrid classifier. The MCS system, an implementation of the approach, is described in **Chapter 4**. Specifically, we describe MCS's model classes and search strategy, and we present a detailed example of MCS applied to a learning task. In **Chapter 5** we evaluate our approach. We first present experimental results comparing MCS to each of its primitive learning algorithms and to the traditional approach of cross-validation. We then present a comparison of MCS to three other hybrid algorithms that each apply all possible algorithms to construct each test node in the hybrid tree. The second half of Chapter 6 is devoted to analyzing the representational biases of MCS's model classes. We discuss the increase in representational power of the hybrids produced by MCS over the primitive model classes. In addition, we investigate what types of data sets the hybrids (and primitive model classes) can and cannot represent well. Finally, **Chapter 6** presents a summary of the dissertation, its contributions and the issues for future investigation uncovered by this research.

CHAPTER 2

AUTOMATIC ALGORITHM SELECTION

Given a set of data, the goal of an inductive learning algorithm is to form a generalization of the data that classifies previously unseen instances accurately. The maximum accuracy achievable depends on the quality of the data and on the ability of the learning algorithm to form the most accurate classifier for that data set. In this dissertation, we assume that the input representation (the features describing the instances) and the data are given and we focus instead on how learning is affected by the choice of learning algorithm.

For any given data set, exhaustively searching the entire space of possible hypotheses is computationally intractable. In order to find an accurate hypothesis in a reasonable amount of time, learning algorithms employ a restricted hypothesis space bias and a preference ordering bias for hypotheses in that space [Dietterich, 1990]. Empirical comparisons among algorithms illustrate that no single bias exists that is best for all learning tasks. [Weiss & Kapouleas, 1989; Aha, Kibler & Albert, 1991b; Shavlik, Mooney & Towell, 1991; Salzberg, 1991]. The manifestation of this problem is the selective superiority that we observe for each learning algorithm; each algorithm is best for some, but not all tasks [Brodley, 1993];

An example of the selective superiority problem is shown in Table 2.1. Each row shows the accuracies of the classifiers formed by three different learning algorithms. The accuracy for each method is the average across ten runs for the test set; for each run the data was split randomly into a training set (90%) and a test set (10%). The data sets are described in Chapter 5 and the algorithms are described in Chapter 4. The problem that no single algorithm is best for all tasks is apparent: for the Glass Recognition data set the decision tree is best; for the Vowel Recognition data set the k-nearest neighbor is best; and for the Waveform data set the linear discriminant is best.

In this dissertation we present a knowledge-based automatic algorithm selection system designed to solve the selective superiority problem. The problem of selecting an appropriate learning bias is complicated further because for some learning tasks, different subtasks are learned best using different concept representation languages. In such cases, a hybrid representation language bias is better than selecting a single representation language. In Chapter 3 we will discuss hybrid classifiers. In this chapter we first discuss automatic algorithm selection as a solution to the selective

Table 2.1 An illustration of the selective superiority problem

Data Set	k-Nearest Neighbor	Linear Discriminant	Decision Tree
Glass Recognition	68	56	**75**
Vowel Recognition	**50**	37	39
Waveform	66	**83**	69

superiority problem, outlining the desired properties of such a solution and presenting a knowledge-based approach that has these properties. We discuss the existing approaches in this context, breaking our discussion into two parts based on how an approach chooses an algorithm. Section 2.2 describes "one-shot" approaches to selection; these methods gather any information that they need and then make a one-time decision. Section 2.3 describes approaches that may change the algorithm based on some type of feedback from the learning process. Finally, in Section 2.4 we discuss the strengths and limitations of the various existing approaches.

2.1 Solving the Selective Superiority Problem

To solve the selective superiority problem an automatic algorithm selection system must select the learning algorithm that will produce the most accurate classifier for previously unseen instances. A learning algorithm has two components that affect this ability: its representation language (model class) and its search bias. A representation language defines the set of possible generalizations for a data set. An algorithm's search bias determines how the generalization space defined by its representation language is searched (the order in which generalizations are considered and how search is terminated). For example the Widrow-Hoff algorithm [Duda & Hart, 1973] has a representation language of linear-discriminant functions and uses the least-mean squares error correction rule to search the generalization space defined by linear-discriminant functions of the data. A change in either of these two components can change the algorithm's ability to produce an accurate classifier.

In this section we first describe a categorization of automatic algorithm selection systems, defining precisely what we mean by automatic algorithm selection. We then discuss the desired properties of an automatic algorithm selection system that would solve the selective superiority problem. Finally, we describe our knowledge-based approach to automatic algorithm selection.

2.1.1 A Categorization of Automatic Algorithm Selection Methods

We can categorize an automatic algorithm selection system by the type of selection it performs. There are three general types of selection:

1. Select among different representation languages, but use the same search bias for each representation.

2. Retain the representation language, but select among different search biases.

3. Select among learning algorithms that differ in both representation language and search bias.

An example of a change in only the representation language is the addition of higher order terms to a linear discriminant function. The same search bias for learning the weights of the function is retained but the addition of new terms changes the search space for a generalization of the data. The second type of selection chooses among several possible candidate search biases, retaining the same representation. For example, selecting between the Widrow-Hoff procedure and the Absolute Error Correction Rule [Nilsson, 1965] for determining the weights of a linear discriminant function is a change in search bias but not representation. Finally, the last type of selection chooses among algorithms that differ in both representation language and search bias. For example, choosing between a linear discriminant algorithm and a univariate decision tree algorithm is an example of both types of change. In our discussion of existing approaches we indicate one or more of these types of selection that each approach is designed to handle. Hereafter when we call a learning method an automatic algorithm selection system we mean a method that performs any of these three types of selection.

2.1.2 Desired Properties

In creating an automatic algorithm selection system one desires that it have the following properties:

1. That it select a best of the set of candidate learning algorithms for each learning task.

2. That it not require a prohibitive amount of time to select a best algorithm for a set of data.

3. That it choose among candidate algorithms that are each suited to different distributions of data.

Given a learning task, an automatic algorithm selection system should select a *best* algorithm for that task from the set of candidate algorithms.[1] We define a best algorithm from a set of learning algorithms to be one that when used to classify previously unseen instances does so with no more errors than any of the other candidate algorithms.

An automatic algorithm selection system should not take a prohibitive amount of time. By selecting among a set of algorithms one enlarges the search space for finding a classifier for a set of data over the space searched by any individual algorithm in the set. With an increase in search space one desires an efficient search algorithm to ensure a computationally feasible search.

Finally, there is no reason to choose among search biases or representations that would each produce the same classifier for all sets of data. Ideally, each algorithm would possess different strengths. For example, a univariate decision tree is best for concepts represented by a set of hyper-rectangles that are orthogonal to the feature axes; a linear discriminant function is best for concepts represented by a single hyperplane. By choosing between the two, one strictly increases the number of learning tasks for which accurate classifiers can be learned, over either individual algorithm. Systems that select among different search biases should also have this property. For example, the Least Mean Squares (LMS) and Absolute Error Correction Rule (AECR) [Duda & Hart, 1973] each have different strengths. When the instances are linearly separable AECR is guaranteed to find the separating hyperplane, whereas LMS is not. When the data are not linearly separable LMS is preferable to AECR, because the error corrections of AECR will not cease, and the classification accuracy of the linear discriminant function will be unpredictable [Duda & Hart, 1973].

2.1.3 Heuristic Search Using Knowledge

To select a learning bias from a set of possibilities, one can examine the data set to find characteristics for which we know one bias is better than another. For example, if the instance space is linearly separable, then we know that a linear discriminant function is a good hypothesis space bias (representation language) and that the absolute error correction rule [Duda & Hart, 1973] will guarantee that a best linear discriminant function is found. Our approach to automatic algorithm selection relies on knowledge about the characteristics that indicate that a particular bias is appropriate, and on the ability to determine whether a data set has those characteristics.

There are many measurable characteristics of data sets. For example, one can characterize data sets by whether the instances are described by numeric or symbolic (nominal) features, or by a measure of correlation among the features and the class labels. The challenge is in uncovering characteristics that will indicate the best bias, and further, whether such characteristics can be computed. Rendell and Cho

[1] Two or more algorithms may be equally suited to a task.

(1990) point out that "data character" plays an extensive role in determining the behavior of learning algorithms. Their view of concepts as functions over the instance space led them to define geometric characteristics such as concept size (proportion of positive instances) and concentration (a characterization of the distribution of positive instances through the instance space), which they subsequently illustrate have a large effect on learning.

A different way to characterize a data set is by constructing a classifier using a particular bias and then examining the resulting classifier to determine whether the bias was appropriate, and if not, what would be a better bias. For example, the model class of a univariate decision tree is a poor choice when the features are related linearly. In such cases, the features will be tested repeatedly along a path in the decision tree, giving evidence that a series of tests are being used to approximate a non-orthogonal partition of the data that is not easily represented by a series of hyper-rectangles. This characteristic indicates that a better bias would be to form a linear discriminant functions.

Our approach to automatic algorithm selection computes characteristics of a data set using feedback from a search through the space of available representation and search biases. The approach iteratively fits a classifier to the data using the representation language and search bias currently considered best for the data set. Next it computes measures of how well the resulting classifier fits the data. The measures are used to decide whether a best classifier has been found or whether further search is required, and if so which bias to try next.

The ability to perform an effective search relies on knowledge of how to recognize whether and why an algorithm is a poor choice, and on using this information to select a better one. We have encoded this knowledge into a set of heuristic rules that work together to guide a search for a best representation language bias. Our approach is based on the following hypothesis: *Domain independent knowledge about data characteristics in the form of feedback from learning can effectively guide an automatic algorithm selection search.*

Indeed, a recent focus of research in machine learning is to understand the tasks for which a particular algorithm will perform better than some specified set of alternatives [Feng, Sutherland, King, Muggleton & Henry, 1993; Aha, 1992; Shavlik, Mooney & Towell, 1991]. Prior to this focus on knowledge-based approaches to automatic algorithm selection, the strategy was to try all candidate methods and choose one based on estimates of their accuracies. Of course these methods are also using a characteristic of the data set to make a selection; the accuracy of a particular algorithm is a descriptive characteristic of the data set. An important difference between the traditional approach and the approach presented in this dissertation is that traditional approaches apply all candidate algorithms to the data before making a selection, whereas our approach may choose an algorithm without trying all of them.

In Chapter 4, we describe a set of heuristic rules used to guide the automatic algorithm selection search. The heuristic rules form an expert system for selecting an appropriate learning bias for a set of data. In the remainder of this chapter we describe

existing approaches to automatic algorithm selection and discuss their potential to be a general solution to the problem of selective superiority.

2.2 One-shot Selection

We define one-shot selection methods to be those that select a learning algorithm based on some criterion that is computed once. In Section 2.2.1 we describe a one-shot selection method that uses an estimate of the classification accuracy of each algorithm to select among a set of candidate algorithms. In Section 2.2.2 we describe four approaches that each use characteristics (other than an estimate of accuracy) of the data to select the most appropriate learning algorithm.

2.2.1 Using Accuracy Estimates

One well-known approach to automatic algorithm selection from statistics is to use cross-validation [Linhart & Zucchini, 1986]. Given a set of data, an n-fold cross-validation splits the data into n equal parts. Each candidate algorithm is run n times; for each run, $n-1$ parts of the data are used to form a classifier, which is then evaluated using the remaining part. The results of the n runs are averaged and the algorithm that produced classifiers with the highest average classification accuracy is selected. Recently, Schaffer (1993) applied this idea to selecting a classification algorithm. The results of an empirical comparison of a cross-validation method (CV) to each algorithm considered by CV, illustrated that on average, across the test-suite of domains, CV performed best. Cross-validation is a general method that can be used to select among algorithms that differ in representation language or search bias.

2.2.2 Using Other Characteristics

An algorithm that performs an initial selection attempts to determine which learning algorithm, from a finite set of possible algorithms, appears most promising for the given data set; the choice is made before the actual learning begins. In this section we describe three such approaches and we include a fourth algorithm ACR [Saxena, 1991b], which although not originally designed to perform initial algorithm selection, could be modified to perform this task.

The approach taken by the VBMS system [Rendell, Seshu & Tcheng, 1987] is to select a learning algorithm based on problem space characteristics, such as the number of features and the number of examples. VBMS is a system that tries to learn the situations for which the bias of each candidate learning algorithm is appropriate. VBMS clusters algorithms based on their performance. Given a new learning task, the system selects an algorithm based on what it has learned and then runs it. If the resulting performance is not at the level desired, then VBMS tries the next best

algorithm. This process continues until satisfactory performance is reached or the system runs out of algorithms to try. When this process is finished, VBMS updates its statistics that relate problem characteristics to algorithm performance.

A recent focus of research in machine learning is to understand the tasks for which a particular algorithm will perform better than some specified set of alternatives [Feng, Sutherland, King, Muggleton & Henry, 1993; Aha, 1992; Shavlik, Mooney & Towell, 1991; Weiss & Kulikowski, 1991]. The knowledge resulting from such efforts can be used to form a heuristic search procedure for automatic algorithm selection. In the STATLOG project, sixteen algorithms were compared across twelve data sets (Feng, et al. 1993). One of the goals of the project was to discern what characteristics of data sets suit particular algorithms. Twelve statistical characteristics were uncovered that they think will be useful for predicting which of the algorithms will perform the best for a given task. However, the heuristics about these characteristics that were put forth have not at this point been evaluated on any data sets not used in the original comparative study.

Aha (1992) presents a method that generalizes case studies of algorithms to generate rules characterizing when differences in performances among learning algorithms will occur. The method takes a data set, for which performance of the various algorithms is different, and then models the data set to create an artificial data set whose characteristics are similar to the actual data set's characteristics. Next the set of algorithms is run on several versions of the artificial data; each version has a different setting of the parameters for generating the data set. The algorithms are evaluated on the different versions of the data set, and a rule is derived to summarize when the performance differences occur. Rules are extracted by CN2 [Clark & Niblett, 1989] from the performance results of the algorithms on the artificial data sets. Aha points out that although the rules derived from the method are highly constrained, they are more useful than results of most empirical comparisons of algorithms, which merely tell you which algorithm performs the best for a set of data.

The third method for selecting an algorithm, the ACR algorithm, was developed to select which of two instance representations permits a learning algorithm, call it X, to induce the better generalization [Saxena, 1991b]. ACR selects which of the two given representations enables X to represent the entire set of instances in fewer bits. The algorithm is based on the *Minimum Description Length Principle (MDLP)* which states that the best hypothesis to induce from a data set is the one that minimizes the length of the code needed to represent the data. The codelength of a hypothesis is the number of bits needed to represent the hypothesis plus the number of bits needed to represent the error vector resulting from using the hypothesis to predict the data [Rissanen, 1989].

ACR uses the *compressibility* of the data as a measure of the suitability of an instance representation for a learning algorithm. By estimating the number of bits required to describe or code a finite set of examples, ACR is able to rank different instance representations without estimating the accuracies directly. Empirical results show that using ACR to rank representations is more time efficient and produces

better results than estimating directly the performance of the learning algorithm with the different representations.

Although originally created to determine which of two input representations is better suited to the learning task, Saxena suggests that ACR could be applied to automatic algorithm selection [Saxena, 1991a]. Given one input representation, ACR would select which of two learning algorithms will produce a better compression of the data. However, to ensure a fair comparison, the coding schemes for assigning the number of bits to a hypothesis must not be biased to favor the hypotheses produced by one learning algorithm over another, which may be difficult because finding provably optimal codings is unsatisfiable.

2.3 Using Feedback during Learning

In this section we describe approaches to automatic algorithm selection that use feedback from the learning process to change the representation language or search bias. The approaches can be divided into five categories, based on the strategy used to search for a best algorithm:

1. **Iterative change based on concept form:** These methods use the form of the learned classifier to change the learning algorithm's underlying representation language.

2. **Iterative change based on measures of previous classifiers:** These methods use numeric measures of classifiers learned using previous choices of representation/search bias to guide the search for the best learning algorithm.

3. **Fixed-order search:** Search through the space of algorithms is conducted in a pre-specified order.

4. **Optimization:** This approach optimizes novelty and performance in the space of algorithms (the bias space) defined by the user-specified learning and performance measurement strategies.

5. **User-specified strategy:** This approach allows the user to specify the strategy for searching for the best learning bias.

This categorization allows us to examine the differences in the control strategies for automatic algorithm selection. In the discussion of each system we answer the following questions:

- What are the candidate representation languages and search biases?
- What type of feedback does the method use to guide search?

- Is the method restricted to one type of selection (i.e., does it select the representation language but fix the search bias)?
- How does the system determine when to stop searching?

2.3.1 Iterative Change Based on Concept Form

One approach for changing the underlying representation language of a learning system is to evaluate the form of a classifier induced from a set of training instances, described in an initial vocabulary, and then change the representation language with respect to this evaluation. In general, methods that employ this approach perform the following cycle until some criterion is satisfied: induce a classifier using instances described in language L_i, use the induced classifier to modify L_i to produce L_{i+1} and then repeat using L_{i+1}. By changing the instance description language, the system changes the underlying representation language of the learning system.[2] These methods have been called data-driven approaches to constructive induction [Michalski, 1983]. The systems described in this section have a cyclical refinement approach to model selection. Once the instance representation has been changed, the previous classifier is discarded and a new classifier is induced using the new instance representation.

A system starts with an initial feature set, F_1, equal to the set of input features. On each successive iteration of a select-fit cycle, new features are created by forming Boolean combinations of the current feature set, F_c. (Initially $F_c = F_1$.) The system decides which features to combine by examining the form of the classifier induced using F_c. Because the combination operator used by these systems is binary, a Boolean feature comprised of n of the initial features requires n iterations of the select-fit cycle. Although the order in which features are generated depends on the current classifier, the search bias of these systems is fixed to search from simple to complex feature sets. A feature set F_{i+1} is more complex than F_i if $\sum_{j=1}^{|F_{i+1}|} |feature_j| > \sum_{j=1}^{|F_i|} |feature_j|$, where $|F_i|$ is the number of features in feature set F_i and $|feature_j|$ is the number of features from F_1 that are combined to form $feature_j$.

FRINGE constructs Boolean combinations of the initial set of features to overcome the *tree replication problem* [Pagallo & Haussler, 1990]. FRINGE executes the following cycle until no new features are generated: induce a decision tree and then generate new features by forming conjunctions of pairs of features that occur at the *fringe* of the tree. Given a leaf labeled positive, a new feature is formed by taking the conjunction of the two tests immediately above the leaf. Empirical results show that FRINGE consistently outperforms a decision tree algorithm that forms tests based on only the initial Boolean input features.

Several extensions to the original FRINGE algorithm have been implemented. Because a small Conjunctive Normal Form (CNF) concept may not have a small DNF

[2]Note that the data has not changed; no new measures of the domain have been added.

representation, the dual-FRINGE algorithm was created to generate features useful for CNF concepts [Pagallo, 1990]. For each leaf labeled negative in the tree, dual-FRINGE forms a new feature by taking the disjunction of the two tests immediately above the leaf. Symmetric-FRINGE is the result of combining both the FRINGE and dual-FRINGE algorithms. The behavior of dual-FRINGE on CNF concepts was similar to that of FRINGE on DNF concepts, however the symmetric version was slightly less effective.

A third extension to FRINGE was introduced by Yang et al. (1991). Their system, DC-FRINGE, is similar to symmetric-FRINGE. The difference lies in the use of context to decide if a proposed feature should be generated. DC-FRINGE restricts the construction of disjunctions to situations in which the sibling of the leaf is also a leaf and the sibling of the parent is a positive leaf. DC-FRINGE performs better than symmetric-FRINGE for the test cases. The authors attribute this to the fact that the extra disjunctions produced by symmetric-FRINGE cause the decision tree algorithm to overfit to the training instances, thereby causing a decrease in the predictive accuracy.

A fourth extension, an algorithm called LINT, is a strict generalization of FRINGE. LINT learns (0,1) DLF functions (i.e., disjunctions of linear threshold functions such that each non-zero weight has a magnitude 1.0). The cyclic feature generation process is identical to FRINGE's, except that the features at the fringe of the tree are used to define a linear function. LINT sets the weight corresponding to a feature tested at a node, whose children are both leaves, to 1. If one child is leaf and the other is a test-node, then the weight is set to 1 if the leaf is labeled negative and to -1 if it is labeled positive. For each linear combination LINT generates the complete set of linear threshold functions. This set is obtained by varying the threshold. Because all of the independent variables of the function are Boolean, the cardinality of this set is finite and is equal to the range of the function. LINT was tested for *r-of-k* linear threshold functions, small disjunctions of *r-of-k* functions and on small random linear threshold functions. The results show that while LINT works well for small (0,1)DLF concepts, it does not do well for linear threshold functions with arbitrary weights.

CITRE [Matheus, 1990] uses the form of a decision tree induced from a set of instances, described by a set of Boolean features, to construct new terms. CITRE is similar to FRINGE, but the feature formation heuristic is slightly different. The system creates new terms by forming a conjunction for each pair of features found on a path to a positive-labeled leaf of the constructed decision tree. It uses two domain specific feature pruning techniques for the domain of tic-tac-toe and then generalizes the remaining features by changing common constants to variables. The resulting feature set is pruned using an information theoretic to retain only 27 features during each iteration; it retains the 9 initial features and the 18 best-ranked constructed features. A new tree is grown and the process repeats until only one positive-labeled leaf remains in the decision tree. Results show that for the domain of tic-tac-toe

CITRE realizes a 15% increase in accuracy over the accuracy obtained by applying ID3 [Quinlan, 1986] to a set of instances described by only the nine initial features.

2.3.2 Iterative Change Based on Measures of Previous Classifiers

In this section we describe systems in which a classifier (or model for the data) is selected, or a new one is proposed, in response to computed measures of the previously learned classifiers.[3] Specifically, we describe approaches to representation selection in statistics, numerical discovery methods that use mathematical relationships among the features to guide search, feature construction methods for learning polynomial functions, and incremental algorithms that adjust the representation language during learning.

Representation Selection in Statistics: Existing automatic algorithm selection systems in statistics are designed to select among different representation languages. These methods fit a model (or classifier) to the data, perform statistical tests to determine the fit of the model, and then may change the representation language or halt depending on the results of diagnostic statistical tests. There are several methods for selecting an appropriate representation and all require specification of the entire set of terms to be considered for inclusion in the model. Term selection is a special case of representation selection. Before discussing the control strategies for term-selection, we first describe the general approach to model formation in statistics.

Model selection in statistics refers to the process of estimating the relationships among the variables of a given data set [Chatterjee & Price, 1977]. Given one dependent variable, y, and k independent variables, $x_1, x_2, ..., x_k$, the goal of statistical model selection is to find a functional relationship, $y = f(x_1, x_2, ..., x_k, \varepsilon)$, which explains or predicts the data. The x_i are often called *predictor* variables because the predicted value of y, $\hat{y}$, depends on the values of the x_i. The disturbance terms, $\varepsilon_i, i = 1, .., k$, are unobserved and represent the underlying disturbance process of the data. One of the most widely used tools to estimate this functional relationship is *regression analysis.*

Regression analysis is a set of data analysis techniques used to help a human data analyst understand the interrelationships among variables in a particular environment. Using regression analysis requires that one make the assumption that the data represent some stochastic process. There has been significant research on selecting and fitting *linear models* to the available data, and our discussion will focus on these techniques. A linear model is any function that is linear in its parameters. The value of the highest predictor variable in the model is called the *order* of the model. For example, $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$ is a second-order linear regression

[3] A similar approach is taken by the IPUS system [Lesser, Nawab & Klassner, to appear March 1995], which iteratively adjusts the parameters of its signal processing algorithms based on feedback from the results of previous settings.

model. The general linear model for the variables $x_1, x_2, ..., x_k$ is written in the form: $y = \beta_0 Z_0 + \beta_1 Z_1 + ... + \beta_p Z_p + \varepsilon$, where $Z_0 = 1$ is a dummy variable (usually unity) and each Z_i, $i \in \{1, 2, .., p\}$, is a general function of $x_1, x_2, .., x_k$ and can take on any form. The β_i are the true population parameters of the model. The goal of regression analysis is to find estimates, b_i, for the β_i.

The most popular method for finding estimates, b_i, for the true population parameters of a linear model is the *ordinary least squares (OLS)* method. Its popularity is due to the fact that it provides minimum variance unbiased estimates of variable interactions that can be expressed additively. This is true only if the underlying stochastic process has the following two properties: the disturbance terms of the variables, $\varepsilon_i, i = 1, .., p$ are random quantities ($\varepsilon = \sum \varepsilon_i$), and they are independently distributed with mean zero and constant variance, σ^2.

The least squares method was invented independently by C. F. Gauss and A. M. Legendre [Draper & Smith, 1981]. The method finds estimates, b_i (for the β_i) by minimizing the sum of the squared residuals, $e_i, i = 1, .., n$, where n is the number of observed data points. The residual error, e_i, of the regression equation, $y = b_0 + b_1 Z_1 + b_2 Z_2 + ... + b_p Z_p + e$, measures the difference in the value predicted for the dependent variable, $\hat{y}_i$, and the observed value y_i, for $i = 1, .., n$.

If the disturbance terms of the independent variables are not independently distributed, then one must use the method of *general least squares (GLS)* (also called weighted least squares) [Draper & Smith, 1981]. GLS requires a specification of a matrix of weights, Ω, which captures all systematic information about the disturbance process.

Examination of the fit of a particular regression equation to the data requires a further assumption about the residuals: for a small data set the e_i must be normally distributed [Chatterjee & Price, 1977]. (Due to the Central Limit Theorem this is always true for large data sets.) The two assumptions, that the residual is a random variable with mean zero and constant variance σ_2, follow from the assumptions about the disturbance terms (i.e, that the $\varepsilon_i, i = 1, ..k$ are random variables, with zero mean and an unknown constant variance, σ^2) [Draper & Smith, 1981]. To measure the fit of a regression equation to the data, we can measure the proportion of total variation about the mean $\bar{Y}$ explained by the regression. The proportion is computed by $R^2 = \sum(\hat{Y}_i - \bar{Y})^2 / \sum(Y_i - \bar{Y})^2$. R^2 falls between 0 and 1 and we can measure its significance using an f-test [Chatterjee & Price, 1977].

With the advent of computers, automated representation selection procedures have been created. There are several methods for selecting the appropriate representation, and all require specification of the entire set of terms Z_i to be considered for inclusion in the model. The terms (called features in machine learning) can be taken from the set of initial variables, $x_i, i = 1, 2, ...k$, and any function of these variables. We present the three most commonly used procedures for selecting the appropriate terms. A more comprehensive list can be found in Draper and Smith (1981).

The first procedure evaluates all possible subsets of the terms to form a model [Chatterjee & Price, 1977]. This method gives the analyst the maximum amount of information available concerning the relationships between y and the Z_i's. This can be intractable if there are many terms to be considered for inclusion in the equation; given p terms, the total number of equations to be considered is 2^p. To evaluate the different regression equations one of several different criteria can be used. The choice of which criterion to use should be related to the intended use of the regression [Hocking, 1986]. For example, if the objective is to obtain a good description of the independent variable and the OLS method is used, then the R^2 statistic is a good choice [Hocking, 1986]. Another commonly used criterion is the residual mean square error, $RMS_p = SSE_P/(n-p)$, where SSE_p is the residual sum of squares for a p-term equation given n data points. Given two equations, the one with the smaller RMS is preferred. For a detailed discussion of RMS see Chatterjee and Price (1977). For a description of other criteria and their relationship to the intended use of the regression see Hocking (1986).

Because of the computational intractability of evaluating all possible regression equations, various methods have been proposed for evaluating only a small number of subsets by adding or deleting terms one at a time according to a specific criterion. The two most common are *stepwise forward selection (SFS)* and *stepwise backward elimination (SBE)* [Draper & Smith, 1981]. SFS begins with no terms in the model and adds one term Z_i at a time until either all terms are in the model, or until a stopping criterion is satisfied. The first term to be included is the term that has the highest simple correlation with the dependent variable y. If the regression coefficient of this term is significantly different from zero, then it is retained in the equation and a search for the second term proceeds. The second term to enter the equation is the term that has the highest correlation with y, after y has been adjusted for the effect of the first term, i.e., the term that has the highest correlation with the residuals from the first step.

SBE begins with all possible terms in the model and eliminates them one by one. The term to be eliminated is the one that makes the smallest contribution to the reduction of the error sum of squares as measured by its *t-ratio* (the ratio of the regression coefficient to the standard error of its coefficient). If all the t-ratios are significant, then the full set of terms is retained in the model. After each term is eliminated, the equation is re-fit and the regression coefficients in the new equation are examined. The procedure terminates if all t-ratios are significant or only one term remains.

In addition to examining the significance of the coefficients of the individual variables to determine which model to use, one can use either the R^2 or the RMS criterion to judge each regression equation computed by the SFS and the SBE procedures. These criteria can be used to terminate the stepwise procedures. Finally a word of caution about the stepwise regression methods is needed. Use of both the *t*-test and the *f*-test assumes that the data are from independent samples. The manner in which these tests are used in the stepwise procedures violates this assumption.

Therefore, these procedures are typically used with caution by the human analyst. Their purpose is to give the analyst a rough idea of the form of the model, rather than to select a model without human intervention.

Linear models can represent a wide variety of relationships. However there are situations in which a non-linear model is appropriate. For example, suppose information about the form of the relationship between the dependent and the independent variables indicates that it is non-linear. Any model not of the form given above is called a non-linear model. For example, $y = e^{\beta_1+\beta_2 t^2+\epsilon}$ and $y = (\beta_1/\beta_2)(e^{\beta_1 t} - e^{\beta_2 t}) + \epsilon$ are two non-linear models. Note that the first example is *intrinsically linear* and can be transformed into a linear model by taking the *log* of both sides of the equation, however, there is no transformation to make the second example linear. If a model is intrinsically linear, then it should be transformed and the regression coefficients estimated using the resulting linear model. A non-linear model can be solved using the least squares method for the non-linear case, and Draper and Smith (1981) describe this method. Little work has been done on automatic representation selection for non-linear models, because if a data analyst decides to use a non-linear model, then he or she usually knows the form of the model.

Measuring Numeric Relationships Among Variables In this section we describe learning systems that search for a mathematical function of a data set by looking for numeric relationships among the variables. The goal of such systems is to find a mathematical function, $y = f(x_1, x_2, ..., x_n)$, that explains or predicts the data. Specifically, they search for a mathematical function of the independent variables, $x_1, x_2, ..., x_n$, that predicts the the value of the dependent variable, y, to some pre-specified degree of accuracy. Such systems are performing numerical function approximation. These systems use the following iterative control strategy: find numeric relationships among the existing set of terms and then test these relationships to see if any of them can predict the independent variable, y. If all of the proposed relationships fail to predict y, then add these relationships to the existing set of terms and begin again. The initial set of terms is the set of input variables.

The BACON programs [Langley, 1986] perform a depth first search for a polynomial function of the independent variables. The search is guided by the repeated application of heuristic production rules to find relationships among the current set of terms. A term is either an initial variable or a polynomial function of the initial variables. At each node in BACON's search tree, the system holds $n-1$ of the n terms constant and varies only one term, call it x_1. It uses the heuristic production rules to analyze the function $y = f_1(x_1)$, when $x_2, ..., x_n$ are held constant. The heuristics search for one of two relationships: $y = cx_1$ or $y = c/x_1$. If either relationship is found, then the system adds a new dependent term, $y/x_1 = c$ or $yx_1 = c$ respectively, to the set of terms at that node in the search tree. In addition, if the system detects a linear relationship, $y = c_1x_1 + c_2$, then it adds $(y - c_2)/x_1 = c_1$ to the list of dependent terms. At each level in the search, the system examines each independent variable's relationship to each of the dependent terms. If any relationships are found, then the system continues searching in a depth-first order, otherwise it backs up a

node in the tree and tries the next relationship that was found at that node. The system terminates its search when a dependent term is found that has a constant value or if the search exceeds a user specified search depth. In summary, BACON uses a depth-first search to find the first equation (called a law in BACON) that is able to predict the value of the user specified dependent variable.

ABACUS is a system that discovers multiple equations for numeric data [Falkenhainer & Michalski, 1986]. The system first discovers a set of equations that explain the data and then uses the A^q algorithm [Michalski & Chilausky, 1980] to generate rules for when each equation should be used. Unlike other quantitative discovery systems, ABACUS does not require the user to specify the dependent variable(s). ABACUS searches a potentially infinite space of polynomial functions. It forms a *proportionality graph* by looking for qualitatively proportional relationships among the input variables. Variable x is qualitatively proportional to variable y if, for some subset of the data, x increases when y increases or x decreases when y increases. In a proportionality graph a vertex represents a term and an edge represents the qualitative proportional relationship between two terms. A term is either an input variable or a polynomial combination of a subset of the input variables. The combinations are constrained by the use of dimensional analysis. To find an equation the system first performs a *proportionality graph search* and if that fails, it then performs a *suspension search.* The proportionality graph search performs a depth-first search of each bi-connected component of the initial proportionality graph according to the equation formation heuristics. The search continues until a new term has a level of constancy greater than a pre-specified threshold or until all combinations have been exhausted. The suspension search is a beam search, in which for each level in the generated search tree the nodes are divided into active and suspended nodes. Suspended nodes are those whose constancy is less than a preset threshold. A new level in the search tree is created by testing the proportionality the active nodes have among themselves and to all active nodes at earlier levels. The search continues until an equation is found or the user specified search depth is reached. If this depth is reached, then the system backtracks one level and re-activates the suspended nodes at that level. When an equation is found, all of the data points that it covers are removed from the data set, and the search begins again using the remaining data points, until all data points are covered by some equation or the systems deems the remaining points miscellaneous. The model space of all polynomial functions is infinite and ABACUS limits its search through user specified parameters. The search through the space is a beam search from simple to complex models. Because the system has the ability to backtrack, in the worst case the suspension search degenerates into a simple breadth-first search.

COPER [Kokar, 1986] takes a different approach to quantitative discovery from the ABACUS and BACON systems. COPER uses dimensional analysis to constrain the search for a polynomial equation of the input variables. The system first finds the relevant arguments of the function and then finds the functional form. To find the relevant arguments, COPER performs the following cycle: it first generates all

dimensionless products of the current set of terms (called the independent arguments) and then selects a *base* set from the independent arguments. A base set is a maximal set of the arguments that cannot be expressed in terms of any of the other arguments. COPER then re-expresses the remaining independent arguments in terms of the base arguments. To determine if this set of arguments is complete, COPER divides the data points into *orbits*. An orbit is a set of points for which the derived values remain constant when the base arguments are varied. The system then calculates the predicted values of the function in each orbit and compares these values to the observed values. If the difference is significant, then the system searches for a new descriptor that reduces this difference by considering all possible ways of combining the existing arguments while obeying the laws of dimensional analysis. After the set of relevant arguments has been found, COPER iterates over each base to search for a simple polynomial formula. If none of these give satisfactory results, then it reconsiders each base using more complex polynomials. COPER can potentially search an exponential number of formulae, because both the number of bases and the number of polynomial functions considered, can be exponential in the number of variables.

Feature Construction for Learning Polynomial Functions: Sutton and Matheus (1991) present a method for learning higher order polynomial functions from examples using linear regression and feature construction. Their system begins by performing a regression on the training set to learn a regression equation for the model $y = k_0 + k_1x_1 + ... + k_px_p$, where p is the number of input variables. If the residual error is approximately zero, then the system halts; otherwise the feature construction process is triggered. For each iteration of the algorithm, one new feature is generated by taking the product of two of the current features. To determine these two features the system calculates the *potential* of each feature in the current set. A feature's potential is the regression of the squared error of the current function over the squared values of the feature. There are n^2 ways to choose a new feature, where n is the size of the current feature set. To constrain the search, the $m << n^2$ most promising pairs of the current features are selected, as judged by the magnitude of sum of the two features' potentials. For each of the m pairs, the system then computes the joint potential of the pair by regressing their product onto the squared error. The pair with the highest joint potential is included in the equation and the system regresses the new equation onto the training data. This process continues until the residual error of the current system is approximately zero. Note that this system relies on the regression to zero the coefficients of irrelevant features. This assumption is correct if the noise in the data set is systematic, however if it is not, then the coefficients of irrelevant features will be close to zero and therefore the features will remain in the equation.

Incremental Algorithms that Adjust the Representation In this section we describe two incremental concept learning algorithms, STAGGER [Schlimmer, 1987] and IB3-CI [Aha, 1991a], which adjust the representation language during the learning process. An incremental learning algorithm updates the concept description

after each new instance is observed, such that after each update the current concept description can be used to classify all previously *observed* instances with a high degree of accuracy. Both systems can adjust the representation language in response to just one instance: a change in the representation is triggered by an incorrect classification.

STAGGER's objective is to learn a set of numerically weighted features called *elements*: an element is a Boolean function of the attribute values. STAGGER associates two weights with each element: the logical necessity (LN) and the logical sufficiency (LS) of the element. Each time STAGGER observes a new instance, it matches each element against the instance and uses the elements' weights to predict whether the instance is a positive or negative example of the concept. STAGGER then updates the LN and LS weights. If STAGGER predicts the incorrect class for an instance it creates new features by applying the Boolean operators AND, OR and NOT to the existing set of features using a set of heuristics to propose combinations. New features are retained as long as their predictive performance stays above that of their component elements. STAGGER has an initial bias toward linearly separable concepts, but can shift its bias by creating new elements if the performance of the classifier is inadequate. The order in which STAGGER searches for new features is from simple to complex, but unlike other systems described that have this ordering, STAGGER has the ability to backtrack.

IB3-CI is an instance-based learning (IBL) algorithm that constructs new features in response to classification errors. IBL algorithms represent concept descriptions with a set of stored instances, and update the concept description after each instance is processed. IB3-CI integrates IB3 [Aha & Kibler, 1989] with STAGGER. Like STAGGER, IB3-CI uses LN and LS weights to guide its feature formation process. Its objective is to reduce the similarity between two instances (the new instance and the stored instance judged most similar) when a misclassification occurs. Features that match the positive and mismatch the negative instances are paired by their increasing summed LN weights. A new feature is formed by taking the conjunction of the first logically unique pair. It replaces a previously constructed feature with the lowest max(LS,1/LN) value with the new feature if the *store* is full. The size of the store, the maximum number of features permitted in the current set, is a system parameter. It then repeats the process with the pairs ordered by their decreasing LS values. IB3-CI was applied to the tic-tac-toe endgame problem [Matheus, 1990]. The results show that feature construction improves the learning performance over IB3 substantially (approximately 20%) for this data set.

2.3.3 Fixed-order Search

In this section we describe methods for automatic algorithm selection that search the space of algorithms using a fixed order. We first describe the perceptron tree algorithm [Utgoff, 1989], which selects between two algorithms that differ in both representation and search bias. We then describe the E^* algorithm [Schaffer, 1990] which conducts an ordered search through six functional forms to fit to the data.

Finally, we describe fixed-order search methods for selecting an appropriate net architecture.

A *perceptron tree*, combines a univariate decision tree with linear threshold units (LTUs) [Utgoff, 1989]. Utgoff defines a perceptron tree to be either an LTU or an attribute test, with a branch to a perceptron tree for each value of the attribute; a decision tree in which every leaf node is an LTU. The depth-first, recursive tree growing procedure chooses between two types of classifiers, a symbolic attribute test or an LTU, to place as a test at each node in the tree. At each subspace, or node, as determined by the partially formed tree, the algorithm first trains an LTU. If the subspace is linearly separable, as determined by a heuristic measure, then the LTU is retained as a Boolean test at that node. If the subspace is not linearly separable, then the space is split via a symbolic attribute selected using an information-theoretic measure. The control strategy for the perceptron tree algorithm employs a fixed order selection strategy; it first tries an LTU, if that fails, it then grows a symbolic decision tree node.

There are two successor systems to the original perceptron tree algorithm, PT2 [Utgoff & Brodley, 1990] and LMDT [Utgoff & Brodley, 1991]. Both removed the explicit choice between the two learning algorithms. Each node in a tree created by PT2 or LMDT is a linear discriminant function (for LMDT each node is a linear machine [Nilsson, 1965]). PT2 and LMDT each use a variant of Sequential Backward Elimination [Kittler, 1986] to select the terms to include at each node. SBE was described in Section 2.3.2.

Schaffer's E^* algorithm searches a finite space of candidate models [Schaffer, 1990]. It implements a fixed order search through six possible functional forms, $y = k_0 + k_1 x^n$, where $n \in \{-2, -1, -.5, .5, 1, 2\}$. The motivation for the order in which the functional forms are considered is to have E^* emulate the way a statistician would search for a model relating the two variables, y and x. The tests that E^* uses to select the best model are: a measure of fit, $MF = 1/(1 - R^2)$, of each equation, to select the best model and the *t-test* to determine if the coefficients for an equation should be zero. The thresholds for these tests were chosen to emulate the choices a statistician would make. The system can also output the null answer, "no relationship found" if no equation passes the tests.

Adaptive neural net algorithms change the representation language used to learn a classifier to overcome the difficulty of specifying the correct net architecture for a given learning task. If we restrict learning in artificial neural networks to adjustments of the weights at each node, then a good generalization can be learned only if the network designer picks the appropriate network architecture for the problem at hand. The number of input and output units are typically specified by the task. Selecting an architecture consists of picking the number of hidden units and specifying the connections among the input, output and hidden units. It is a well known problem that a network with too many hidden units overfits the training instances and in the worst case results in rote learning, whereas a network with too few hidden units

overgeneralizes. Both cases decrease the ability of a trained network to classify previously unseen instances correctly.

A neural net algorithm that chooses its own architecture frees the human architect from the trial and error process often required to find a best architecture. There are two basic approaches for changing the net architecture dynamically. The first approach starts with a large net and removes hidden units/connection until further removal appears as if it will result in overgeneralization [Mozer & Smolensky, 1989; Hanson & Pratt, 1989; Karnin, 1990; Le Cun, Denker & Solla, 1990] The second approach starts with a small net and adds units as deemed necessary for learning the concept [Gallant, 1986; Honavar & Uhr, 1988; Ash, 1989; Frean, 1990b; Fahlman & Lebiere, 1990].

There are two approaches for removing units and/or connections. After the network has been trained, algorithms that use the first approach compute the importance of each unit/connection for keeping the error rate low, and then eliminate some number of the least important units. Training then continues on the resulting network [Karnin, 1990; Le Cun, Denker & Solla, 1990; Mozer & Smolensky, 1989]. The second approach for reducing the size of the net is to modify the actual weight training algorithm such that unnecessary connections or units have zero weight or output after training. *Weight decay* achieves this end: the weight on each connection is decremented toward zero by a certain factor at each update. The weights corresponding to important connections move away from zero and unimportant ones move toward zero as learning proceeds. Alternatively, weight decay can be performed implicitly by changing the error function. Hanson and Pratt's (1989) method adds terms to the error function to penalize hidden units that have small outputs.

There are several algorithms that start with a small net and add units/connections to decrease the classification error of the net. These methods differ in when and where new units are added to the net during training. Honavar and Uhr's (1988) *generation* method grows links and adds units whenever the net's performance is not improving. The process halts when the net converges. Ash's (1989) *dynamic node creation* algorithm trains networks that have only one hidden layer. If the rate of decrease in the error falls below a preset threshold, a new fully-connected unit is added. The *upstart* algorithm uses binary units and grows a binary tree-structured network [Frean, 1990b]. Two new children are added if a parent cannot classify the instances correctly; the children correct the output of the parent. The *cascade correlation* algorithm starts with one layer [Fahlman & Lebiere, 1990]. If the required mapping can not be learned by the one layer, then a hidden unit is added as a hidden layer and trained while the previously trained weights are frozen. More hidden units are added until the correct mapping is learned. The new unit is fully connected to the input layer and to the unit added previously. Gallant (1986) proposes three constructive network architectures: the tower, the inverted pyramid and the distributed construction. New units are added if the current net is incapable of classifying the instances to a specified degree of accuracy. When a new unit is added, the coefficients of the existing units are frozen.

2.3.4 Optimization

Tcheng, Lambert, Lu and Rendell (1989) created a system that couples induction with optimization to search the *inductive bias space.* This space is defined by the observed examples and a subset of the system's set of learning algorithms (chosen by the user). The system contains two components: the *Competitive Relation Learner (CRL)* and the *Induce and Select Optimizer (ISO).* CRL is a generalized recursive splitting algorithm that produces decision trees in which each internal node is either a univariate test or an arbitrary hyperplane (generated randomly). Each leaf node can be a univariate test, a neural network, a k-nearest-neighbor classifier or a regression model (linear, quadratic, logarithmic or exponential). CRL applies all user specified learning algorithms to the set of instances observed at a node in the partially formed tree. CRL evaluates each resulting classifier using the accuracy for the training set, the accuracy of an independent test set or a n-fold crossvalidation (the user specifies the choice of evaluation measurement procedure). The recursive tree formation strategy continues until the error is less than T_1, the number of examples at a node is less than T_2 or the time consumed is greater than T_3. The choice of stopping criterion, T_i, and its value are specified by the user.

Because trying all of the learning algorithms is computationally expensive, the second component of the system, ISO, optimizes the search for the correct inductive bias. To select which bias to apply (which learning strategy) ISO balances *novelty* and *performance* in the *bias space.* The bias space is defined by the user specified learning and error measurement strategies. The novelty measure, which is set by the user, directs search to points in the bias space far away from strategies that have already been tried. The chosen performance measure directs search to points that have a low error rate. Note that this approach assumes that one can structure the bias space such that learning algorithms that are similar in bias and performance are close to one-another.

The system begins by probing randomly in the bias space to produce n optimization data points, (X_i, O_i), where X_i represents a bias point and O_i measures its credibility as measured by X_i's error. ISO uses these data points to train its optimization function. From this point on, the search uses the optimization function to select the next point in the bias space to explore. The next point can either be a partially formed classifier (a point already visited) or a new learning strategy. How long the system spends with one learning strategy is a user controlled parameter. The system halts when CRL's stopping criteria are met.

2.3.5 User-specified Strategy

The SBS testbed [Provost & Buchanan, 1994] allows the user to choose an *inductive policy* for forming a generalization. An inductive policy is a search strategy for searching the inductive bias space. It includes the search algorithm (called bias transformation operators) and a performance evaluation function. Typically, an

evaluation function will specify a tradeoff between accuracy and the reduction of some cost (eg., time, space, or complexity of the concept description). Note that the evaluation function could also be maximization of accuracy.

This approach requires an explicit representation of the bias space. SBS is built around a multiclass version of the RL learning system [Provost & Buchanan, 1992]. RL is a learning method that allows many different biases to be specified explicitly, such as the beam search width and the complexity limit on rules. SBS repeats the following cycle until a stopping criterion is met: choose candidate biases using the bias transformation operators; run RL with each bias; compare biases (rule sets) using the chosen evaluation function; and then choose the best bias. Three basic search strategies are: select one bias at random, exhaustively search bias space, and choose among several biases selected at random from the bias space.

Provost and Buchanan(1994) give three extensions to these basic policies. Firstly, one can add structure to the bias space by adding domain knowledge or knowledge about biases to order biases or restrict search from considering some biases. Secondly, the results from searching with the first i biases can be used to determine the next appropriate bias to try. This extension assumes that new biases can be formed from existing biases in a useful manner. An example of this is the SFS algorithm described in Section 2.3.2. The final extension builds concept descriptions based on elements learned in several different biases. This strategy assumes that partial concept descriptions can be used to construct coherent concept descriptions and can be evaluated for "goodness".

2.4 Discussion

In this section we discuss our knowledge-based approach and the other existing approaches' potential to be a general solution to the automatic algorithm selection problem. In Table 2.2 we show the type of selection that each approach handles with an "X". Methods that can perform all types of automatic algorithm selection are general approaches – they can be used for any set of candidate algorithms. Note that we do not mean the specific implementation, but rather the general approach behind the implementation.

Our knowledge-based approach is a general approach to automatic algorithm selection; it can select among different representations or search biases. Our approach incorporates many of the features of existing systems. Like the methods described in Section 2.3 it uses feedback during learning to guide the search for the best algorithm. Unlike many of these methods it does not exhaustively examine all candidate learning algorithms.

Crossvalidation, and ISO/CRL are general approaches that can be applied to any mix of representation and search bias. Both methods require exhaustive search through the space of candidates to ensure that the best algorithm has been selected.

Table 2.2 Catalogue of automatic algorithm selection systems

Approach/System	Representation Only	Search Bias Only	Both
Knowledge-Based Approach	X	X	X
Crossvalidation	X	X	X
VBMS		X	
STATLOG			
Generalize Case Studies		X	
ACR	X		
Fringe, CITRE	X		
SBE and SFS Term Selection	X		
BACON, ABACUS, COPER	X		
Feature Constr. for Poly. Fns.	X		
STAGGER, IB3-CI	X		
Perceptron Trees			X
E^*	X		
Adaptive Neural Nets	X		
ISO/CRL (Optimization)	X	X	X
SBS Testbed	X	X	X

Neither of these approaches use information about why a particular algorithm is unsuited to the given data set, and therefore must search randomly. In addition, in the case of ISO/CRL, the system gives the same time limit to each learning strategy. Some algorithms require more time than others. Therefore, if a particular algorithm produces a classifier with a high error rate it is unclear whether this is due to insufficient training time.

ACR has the potential to be a general approach, but has only been applied to initial feature selection. In using ACR to select among different representation languages, one must ensure that the coding schemes for assigning the number of bits to a hypothesis not be biased to favor the hypotheses produced by one learning algorithm over another. This is difficult because finding provably optimal codings is unsatisfiable.

The ideas behind STATLOG are promising for a general approach to automatic algorithm selection. However, thus far the heuristics resulting from STATLOG have not been incorporated (tested) in an automatic algorithm selection system. Using one set of learning problems to derive a decision process for selecting among algorithms based on characteristics of the data was shown to be promising in Aha's method for generalizing case studies. The difficulty is in coming up with characteristics of data sets that allow one to distinguish among different learning algorithms. The work in this area is preliminary, but hopefully will continue over the next few years.

Considering candidate algorithms in a fixed order is a general selection method. However fixed-order selection methods implement biases that may be inappropriate for some learning tasks. In the perceptron tree algorithm, effort is wasted determining that a space is not linearly separable; training an LTU requires significantly more time than selecting a single attribute test. In addition it may be that a subset of the input variables would produce a test superior to either a univariate test or a test based on all of the input variables.

Allowing the user to specify the inductive policy is a general approach. However, to pick a policy that does not result in exhaustive search, one requires significant knowledge about the biases of the learning system to structure the bias space.

The remainder of the methods described in this chapter select among representation languages. None of these methods is a general approach to automatic algorithm selection system. This does not mean that the ideas incorporated in each approach could not be used for part of an automatic algorithm selection system. Indeed the idea of using the performance results of algorithms already tried to help select the next candidate algorithm is a promising one and in Chapter 4 we will see many of these ideas incorporated into a new general approach.

CHAPTER 3

HYBRID CLASSIFIERS

One can characterize learning algorithms as searching either a *homogeneous*, *heterogeneous* or *hybrid* hypothesis space. A homogeneous hypothesis space is one that contains hypotheses expressed in a single representation language. For example, univariate decision trees and linear discriminant functions are each single representation languages. A heterogeneous hypothesis space contains hypotheses from several different representation languages; each individual hypothesis is expressed in one of the representation languages contained in the heterogeneous space. For example, a hypothesis space that contains hypotheses expressed as *either* a linear discriminant function *or* a univariate decision tree is a heterogeneous hypothesis space. Finally, a hybrid hypothesis space is a space that combines different homogeneous hypothesis spaces; each hypothesis is expressed using one or more representation languages. For example, a representation language that mixes linear discriminant functions and univariate decision trees in the same hypothesis defines a hybrid hypothesis space.

Traditionally, machine learning researchers have created inductive learning algorithms that form homogeneous classifiers. Restricting a learning algorithm to one that employs a homogeneous representation bias assumes that a classifier for the data set is represented best using a single representation language. (This is also true of heterogeneous hypothesis spaces; although one can select among different homogeneous biases, the end result is still a classifier expressed in a single representation language.) However, for some data sets this may not be the case; different subspaces of the learning task may be learned best by employing different representation biases, indicating that forming a hybrid classifier will result in higher classification accuracy than a homogeneous classifier. For example, consider the following concept: heart attack caused by a weight problem. A heart attack can be caused by two extremes, obesity or anorexia.[1] Given a set of positive and negative training examples described by three attributes, sex, height and weight, a concept learner needs to learn the subconcepts "underweight" and "overweight". These two subconcepts are different depending on whether the patient is a woman or a man. A single linear threshold unit (LTU) is not appropriate for the entire instance space because the space is not linearly

[1]Anorexia patients typically have a potassium deficiency due to lack of food, which in turn may cause heart failure.

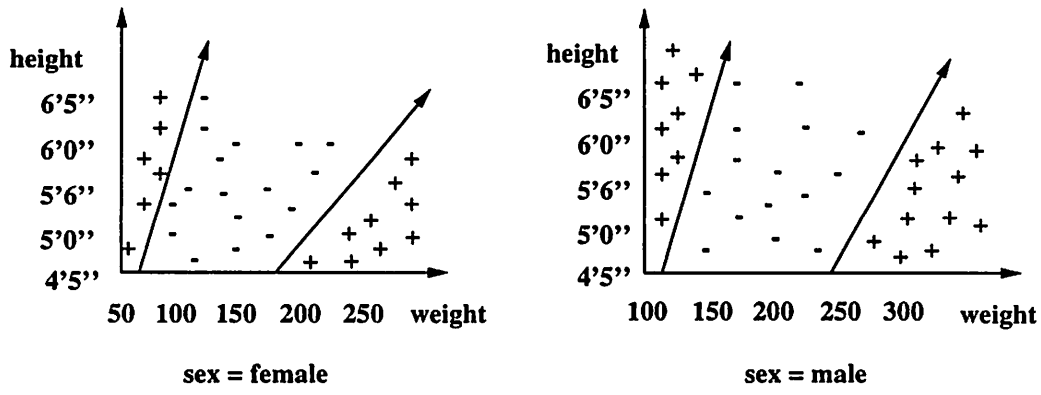


Figure 3.1 The concept heart attack; "+": positive instance, "-": negative instance.

separable, as can be seen in Figure 3.1. Moreover, a symbolic decision tree algorithm would have difficulty learning a compact generalization for this concept because it would need to approximate the hyperplanes with a series of splits orthogonal to each of the axes.

A better solution (shown in Figure 3.2) is a hybrid formalism that combines LTUs and decision trees. This example illustrates that choosing initially between a decision tree and an LTU will not yield as succinct and accurate a classifier as combining both representation languages and employing a control strategy to choose between them, for each subspace of the instance space. For such cases one would like to mix different representations and search biases to create a hybrid classifier.

We define a hybrid classifier to be one that combines different homogeneous classifiers. There are three general approaches. In the first approach a hybrid classifier consists of a set of n homogeneous classifiers, one for each of n distinct regions (subspaces) of the instances space. This requires a method for determining how to partition the space into n distinct regions and how to form a homogeneous classifier for each region. The second type of hybrid classifier combines the classification decisions of several different homogeneous classifiers. For each of these two approaches each homogeneous learning algorithm in the hybrid learns a classifier for a set of data without the influence of any other algorithm; what set or subset of the data is given to each homogeneous learning algorithm and how the resulting classifiers are combined differs depending on the hybrid classifier formation algorithm. Finally, like the second approach, the third approach combines the classification decisions of different homogeneous classifiers, but the classifiers are not trained independently.

The need for hybrid classifiers is based on the hypothesis that for some data sets different parts of the instance space are learned best using different learning algorithms. In this chapter we first outline the desired properties of a hybrid classifier construction algorithm. We then describe existing approaches, pointing out the limitations of each. We focus our discussion on algorithms that *explicitly* choose

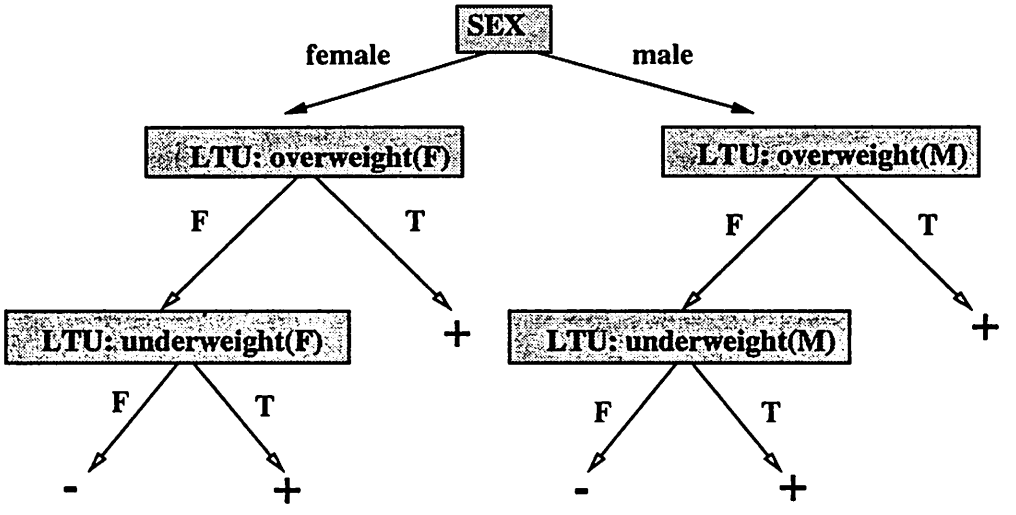


Figure 3.2 Classifier induced for the concept heart attack using a hybrid formalism.

among and mix different learning algorithms. Examples of systems that contain various properties of different algorithms can be found in Schlimmer (1986), Brodley and Utgoff (1992), Clark and Niblett (1989), and Towel, Craven and Shavlik (1991). We conclude this chapter with a description of our approach to hybrid classifier construction that is designed to have the properties outlined in Section 3.1.

3.1 Desired Properties

In creating a hybrid classifier construction system one desires that it have the following properties:

1. That it subsume its primitive components.
2. That it partition the data set into subspaces that each are learned best by one of the candidate homogeneous classifiers.
3. That it select a best classifier (hybrid or homogeneous) for each subspace of the instance space.
4. That it combine algorithms that are each suited to different distributions of data.

A hybrid classifier construction algorithm should subsume its primitive components; for any given data set it should produce a classifier at least as accurate as the classifier produced by each of its primitive homogeneous algorithms. If our

hypothesis that for some data sets different subspaces are learned best by different algorithms is true, then for these data sets the hybrid classifier should be more accurate than classifiers produced by each of the homogeneous algorithms. This requires that the hybrid classifier formation algorithm partition the instance space into subspaces that each are learned best by one of the primitive homogeneous algorithms. The ability to combine different representation languages in a single hybrid structure allows a space of classifiers that is strictly richer than the union of its constituent homogeneous components. In designing a search strategy for a hybrid classifier construction algorithm, one does not want to lose the option of picking a single homogeneous classifier, which requires an ability to recognize whether the best classifier is a homogeneous or hybrid classifier for a given set of instances.

As discussed in Chapter 2, there is no reason to combine algorithms that would each produce the same classifiers for each set of data. The algorithms included in the set of candidates considered by the hybrid classifier construction algorithm should be complementary [Utgoff, 1989]. There are two components of each algorithm that must be considered: the algorithm's representation language and its search bias for selecting a generalization. Concepts or subconcepts that are difficult to represent well in one language may be easy to represent in another. Indeed, a system that can combine different representation languages and search biases can represent accurately a larger class of concepts.

3.2 Existing Approaches

An important distinction among hybrid classifier construction algorithms is the way in which they combine the different primitive algorithms. There are three basic approaches to constructing a hybrid classifier. The first is to assign explicitly a different classifier for each mutually exclusive subset of the data [Tcheng, Lambert, C-Y Lu & Rendell, 1989; Utgoff, 1989], which requires a method for splitting the data into mutually exclusive subsets. Given an instance, a decision procedure is used to decide to which subspace it belongs and the classifier for that subspace is used to classify the instance. A second approach is to apply each of several different learning algorithms to the entire set of data and then to combine their outputs [Wolpert, 1992; Breiman, 1992; Zhang, Mesirov & Waltz, 1992]. A third approach uses different subsets of the training data to train each of a set of classifiers and then classifies previously unseen instances using some scheme to combine the outputs of the set. In Sections 3.2.1, 3.2.2 and 3.2.3 we describe each of these approaches in detail.

3.2.1 Explicit Combination

Methods that explicitly assign different classifiers to different subspaces of the instance space require a procedure to determine whether one of the homogeneous

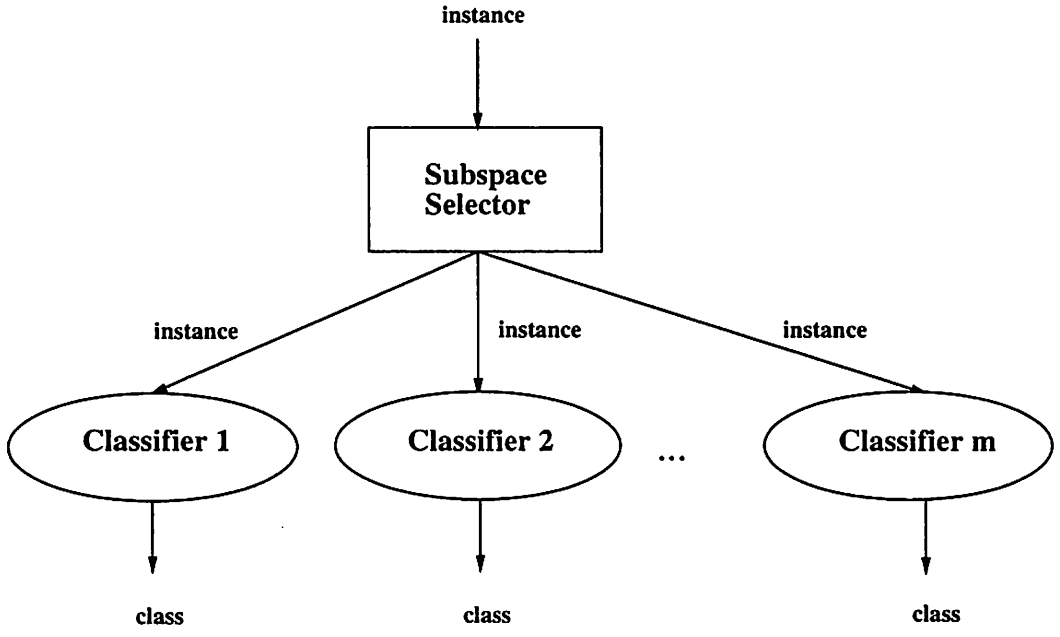


Figure 3.3 A hybrid classifier that explicitly assigns a classifier to each subspace.

classifiers is ideal for a given space (or subspace) or whether the space should be split into a new set of subspaces. Once further splitting is deemed necessary, the next step is to find a partition of a space that creates subspaces that are each learned well by one of the homogeneous learning algorithms. The ability to partition the instance space into homogeneous subspaces, that are each ideal for one or more of the candidate homogeneous learning algorithms, allows a hybrid algorithm to find the best hybrid classifier for the entire instance space.

In Figure 3.3 we show a hybrid classifier that assigns explicitly a different classifier for each of m mutually exclusive subspaces of the instance space. To classify a previously unseen instance, it first selects the correct subspace and then returns the class label assigned by the classifier for that subspace. Note that the procedure used to assign the instance to a subspace is also a classifier; it classifies the instance as belonging to a particular subspace.

In designing an algorithm that forms a tree-structured hybrid classifier, one must decide whether classifiers of each representation language can be at any test node in the tree or whether classifiers from certain representation languages are only permitted in some parts of the tree. For example, the Perceptron Tree algorithm [Utgoff, 1989] is a recursive hybrid-classifier construction algorithm that combines decision trees and linear threshold units. Utgoff defines a perceptron tree to be a decision tree in which each leaf node is a perceptron and each test node is a univariate symbolic test. In terms of the organization shown in Figure 3.3, the subspace selector

is a univariate decision tree and each of the m classifiers is a perceptron. The search strategy of Perceptron Trees was discussed in Chapter 2.

In the AIMS system [Yerramareddy, Tcheng, Lu & Assanis, 1992], the model formation component, CRL [Tcheng, Lambert, C-Y Lu & Rendell, 1989], forms a recursive hybrid structure. CRL partitions the instance space recursively by selecting among the user-specified decomposition strategies (univariate tests and arbitrary hyperplanes). After CRL determines that further decomposition (partitioning) is not desirable, it searches for the best of a user-specified subset of a univariate test, a neural network, a k-nearest-neighbor classifier or a regression model (linear, quadratic, logarithmic or exponential). CRL restricts the types of tests permitted at internal nodes of the tree to decomposition strategies; in the context of Figure 3.3 the subspace selector is a decision tree consisting of univariate tests and arbitrary hyperplanes.

A different approach to forming a hybrid classifier is taken by Jacobs, Jordan, Nowlan and Hinton (1991). Their goal is to reduce the interference effects that occur when training a single multilayer network to perform different subtasks on different occasions; interference effects lead to slow learning and poor generalization. Jacobs et al. reduce such interference effects by using a system composed of several different "expert" networks plus a gating network that decides which of the experts should be used for each training case. During training, the gating network allocates a new case to one or a few experts, and if the output is incorrect the weight changes are localized to these experts (and the gating network). Therefore each expert can specialize in different cases and there is no interference among the weights of different experts. Each expert is a feedforward network and all experts receive the same input and have the same number of outputs. The gating network is also feedforward and receives the same input as the expert networks and has one output unit for each of the n experts. The gating network makes a stochastic decision about which single expert to use on each occasion. The selector acts like a multiple-input, single-output stochastic switch; the probability that the switch will select the output from expert j is p_j, where p_j is the output of unit j of the gating network.

3.2.2 Stacked Generalization

The second general approach to forming hybrid classifiers is to apply each of several different learning algorithms to the entire set of data and then to combine their outputs [Wolpert, 1992; Breiman, 1992; Zhang, Mesirov & Waltz, 1992]. Such approaches have been called *Stacked Generalization* [Wolpert, 1992]. To construct such a classifier requires specifying the primitive components and selecting a combination scheme. Ideally, the primitive components are chosen such that each works well on different data sets (as discussed in Section 3.1).

Stacked generalization can be seen to be a more sophisticated version of crossvalidation, which is a "winner takes all" strategy for combining the individual classifiers. In stacked generalization, the combination scheme can be simply a weighted average or another learning algorithm can be applied to determine how the outputs of the

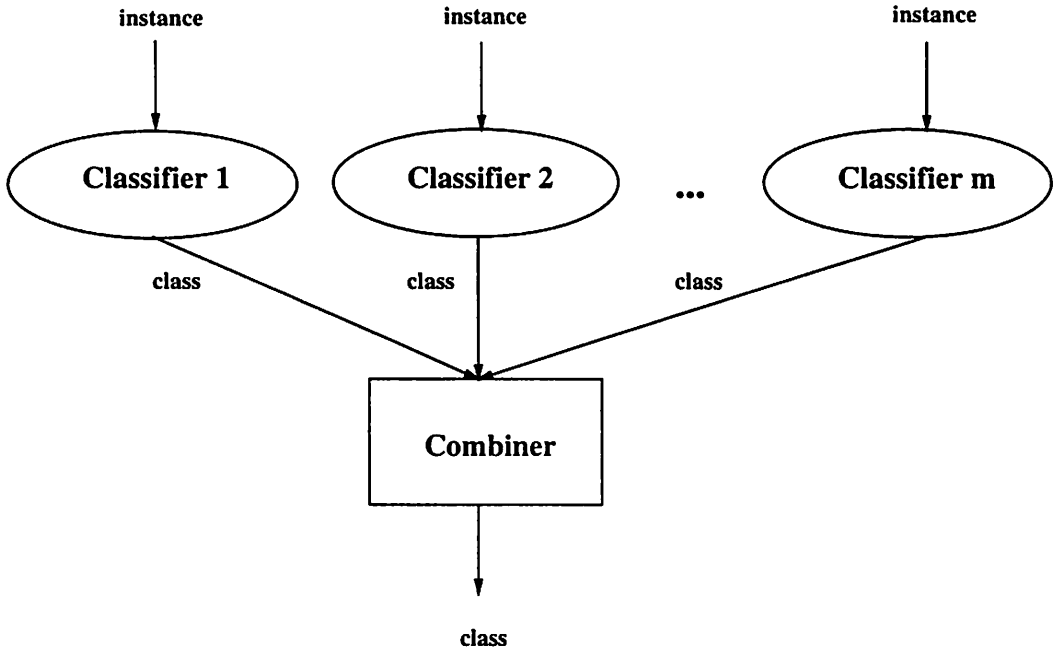


Figure 3.4 A hybrid classifier that combines the classification decisions of m classifiers.

primitive classifiers should be combined. In Figure 3.4 we show the structure of a hybrid classifier constructed using stacked generalization. A previously unseen instance is classified by applying each of the m classifiers to assign m class labels and then using a combination strategy to output a single label. The "combiner" takes as input the outputs of the m classifiers. Note that it could also input the feature values of the instance to be classified.

Wolpert (1992) presents a general method for forming a stacked generalization of m classifiers.[2] For each of the n instances, construct all m classifiers using the remaining $n-1$ instances. Classify the n^{th} instance using each of the m classifiers to create a vector of the predictions, $guess_m$. Couple $guess_m$ with the true class label to create a training instance of the "level-one" data. This process creates n instances of level-one data that are used to train the combiner. Note that one can add other information to each level-one data point, such as the values of the features of the instance [Wolpert, 1992]. After the combiner is trained, the full set of data is used to construct each of the m classifiers.

[2]Stacked Generalization can also be used to adjust for the biases of a single classifier, but we restrict our discussion here to the hybrid case. The reader is referred to Wolpert (1992) for non-hybrid uses of stacked generalization.

In a crossvalidation selection of the single best classifier, the level-one data would be used to select the best classifier. Wolpert's idea is that the level-one data has more information in it and can be used to construct good combinations of the guesses of the primitive algorithms. Wolpert shows that using level-one data to form nonlinear combinations of three nearest neighbor classifiers achieves a performance improvement of 20% over selection of the best single classifier by crossvalidation for the Nettalk data. However, each of the three nearest neighbor classifiers was given access to a single input letter slot (slot 3, slot 4 and slot 5). Therefore, Wolpert states that it is possible that either backprop or a nearest neighbor classifier that had access to all seven slots might achieve better performance for this data set. For more details on how the experiment was performed see Wolpert (1992).

Breiman (1992) illustrates that stacking regressions can improve performance.[3] Breiman states: "If the vector of independent features is high dimensional and the sample size N is not large compared to the number of variables, then it is known and agreed that the ordinary least squares predictor has too much variance and needs modification." A difficulty is that the two solutions to this problem, variable subset selection and ridge regression, each work on opposite ends of the "coefficient spectrum." Subset selection works well if there are a few large coefficients and the rest are zero or nearly zero; whereas ridge regression works well if there are many small non-zero coefficients but no large ones. Breiman shows empirically that combining regression predictions using stacked generalization results in more accurate predictors of the dependent variable, y, than the traditional strategy of picking the best one (typically by crossvalidation). In the particular cases that were tried, the linear combination, $\sum_{i=1}^{k} \beta_i P_i$, of the k predictors P_i, did not always result in a prediction error lower than the best of the individual regression equations. However, when the coefficients, β_i, were constrained to be non-negative, the linear combination did illustrate lower prediction error than each of the individual regression equations. Indeed, in some cases the improvement was substantial.

Zhang, Mesirov and Waltz (1992) developed a hybrid system to predict the secondary structures of proteins. The hybrid's performance was better than each of the individual classifiers in the hybrid and than all previously reported methods applied to this domain. Each of a memory-based reasoning system, a statistical method (which uses Bayes theorem) and a neural network were trained on the same half of the data. The outputs of these trained "experts" on the second half of the data were used as inputs to train the combiner, which is a neural network. After the combiner is trained, each expert was then retrained on the entire data set.

One problem with Stacked Generalization is that these methods can result in a hybrid classifier that is less accurate than one of the primitive components if a bad combination scheme is used [Wolpert, 1992]. Indeed, it is an open problem how to form a procedure for choosing or combining the outputs of the primitive learning components. Much of the success of these efforts is due to finding a good

[3]For a description of regression analysis see Chapter 2 Section 2.3.2.

representation and search bias for forming the classifier used as the combiner. In the case of stacked regression [Breiman, 1992] the researcher needed to discover that non-negative coefficients led to a performance improvement, whereas unconstrained coefficients could lead to a performance decrease over the best individual regression.

3.2.3 Boosting

A third approach to forming hybrid classifiers originates from Schapire (1990). The general idea is to use different subsets of the data to train different classifiers. To classify an instance, the outputs of the classifiers are combined using a particular combination scheme. The algorithm for training proceeds as follows. Train the first classifier using a set of N training examples, then flip a fair coin. If the coin is heads, pass new instances through the first classifier until it misclassifies an instance, and add this instance to a second training set. Otherwise if the coin is tails, pass instances though the first classifier until it finds an instance that it classifies correctly and add this instance to the second training set. Repeat this process until enough patterns have been collected and then train the second classifier with this new training set. The first two classifiers are then used to produce the training set for a third classifier in the following manner: classify new instances using the first two trained classifiers. If the classifiers disagree on the classification, add this instance to a third training set. Continue this process until enough instances are generated to form a third training set. Then train the third classifier. Once the third classifier has been trained, the entire set is used to classify instances.

To classify an instance, each of the three classifiers is applied to the instance, and a class label is assigned using the following voting scheme: if the first two classifiers agree, then that is the class label. Otherwise assign the class label as classified by the third classifier. Boosting has been applied to improve the performance of neural networks, but there is no reason that it could not be applied to other types of learning algorithms. When neural networks are used, one useful modification to the basic boosting algorithm is to add together the three sets of outputs from each of the three networks to obtain one set of outputs [Drucker, Schapire & Simard, 1993]. This modification reduced the error rate by 0.5

3.3 A General Recursive Hybrid Algorithm

In this section we describe a new approach to hybrid classifier construction. Like the systems described in Section 3.2.1 our approach creates a tree-structured hybrid classifier. Unlike Perceptron Trees [Utgoff, 1989] and CRL [Tcheng, Lambert, C-Y Lu & Rendell, 1989], our method makes no distinction between decomposition and learning strategies; any of the primitive learning algorithms can be used to form a subspace selector or classifier. We do not enforce any restriction because there

Table 3.1 Algorithm for forming a tree-structured hybrid

```
Form Classifier(instances)

IF instances from a single class
   THEN return(class)
   ELSE select the best algorithm from the set of candidates to create a classifier
            partition the instances using the classifier
            for each partition call Form Classifier(partition)
```

is no evidence that the best decomposition of a given data set can be found with orthogonal partitions (univariate tests) or arbitrary hyperplanes. Indeed specifying that the decomposition partitions must be orthogonal to the feature axes restricts the way in which subspaces can be formed and may preclude finding a partition for which each subspace is ideally suited for one of the algorithms. Furthermore, partitioning the space with an arbitrary hyperplane is in conflict with the goal of decomposing the space into learnable subspaces. By chance one may be found, but our approach searches explicitly for partitions of the data that create subspaces easily learned by one of the candidate homogeneous learning algorithms.

Given a data set, the goal of our approach is to select the most appropriate of a set of candidate learning algorithms. The approach applies a hill-climbing search guided by a set of heuristic rules. Next, the data set is partitioned using the classifier formed by the chosen algorithm and the search is applied recursively to each resulting subset of the data that contains instances from more than one class. Once a classifier has been fit that classifies each instance in the training data correctly, the next step is to apply a heuristic pruning procedure to ensure that the resulting classifier does not overfit to the noise in the training data. The general recursive algorithm for forming a hybrid classifier is shown in Table 3.1.

Constructing a hybrid classifier requires a method for partitioning the data into useful subspaces and a method for choosing the best learning algorithm for each subspace. Therefore, the problem of automatically selecting the best algorithm for a set of data needs to be addressed when creating a hybrid classifier construction algorithm.

The space of hybrid classifiers is strictly larger than the space of homogeneous classifiers defined by the homogeneous representation languages contained in the hybrid space. By increasing the space of possible hypotheses we strictly increase the probability that for a given data set, a good generalization will exist in the space searched. Increasing the search space does not ensure that a good generalization will be found; it depends on the search algorithm. Therefore, a hypothesis of this research is: *Our knowledge-based search strategy for finding a hybrid classifier will*

produce a classifier that is never worse than, and for some data sets is better than, any homogeneous classifier produced by its primitive components.

Another consideration is the cost of searching a larger space. Our empirical results, reported in Chapter 5, illustrate that merely increasing the search space by permitting hybrid classifiers will not lead to an increase in performance; one needs a search strategy that will not be misled by the larger number of possibilities. In addition, a comparison of our method to several other hybrid classifier construction algorithms illustrates that applying knowledge about the biases of the homogeneous algorithms to search this larger space can substantially reduce the time needed over knowledge-poor methods.

CHAPTER 4

A RECURSIVE AUTOMATIC ALGORITHM SELECTION SYSTEM

We have implemented our recursive automatic algorithm selection approach, producing the Model Class Selection system (MCS). Given a set of data, MCS builds a classifier using a set of heuristic rules to guide a hill-climbing search for the best representation and search bias from which to form a test for each node in a hybrid classifier. Figure 4.1 shows an example of a hybrid classifier that MCS might construct. The root of the tree is a linear combination test, the left subtree is an instance-based classifier and the right subtree is a two-node univariate decision tree. Each leaf node is labeled with one of three classes (A, B, or C). In this chapter we first describe the design goals of our implementation. We then describe MCS's model classes, search strategy, and the data-set characteristics, computed during search, that lead MCS to prefer one bias over another. We conclude with a detailed example of MCS applied to a learning task.

4.1 Design Goals

The implementation of our recursive automatic algorithm selection approach was designed with several goals in mind. In Chapters 2 and 3 we outlined the desired properties of an automatic algorithm selection system and a hybrid classifier construction algorithm, respectively. By combining these properties we derive the following list of abilities that an implemented system should have:

1. Choose among and combine candidate algorithms that are each suited to different distributions of data.
2. Subsume its primitive components.
3. Select a best classifier (hybrid or homogeneous) for each subspace of the instance space.
4. Avoid a prohibitive amount of time to select a best algorithm for a set of data.

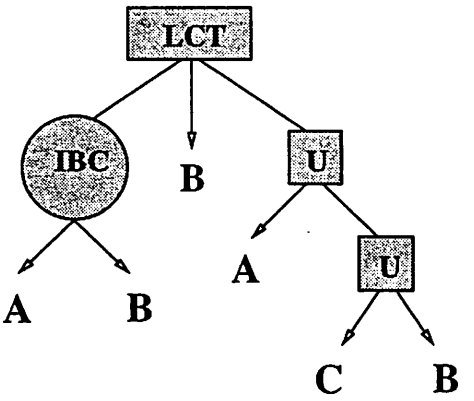


Figure 4.1 Example of a hybrid tree-structured classifier.

We chose the following three primitive representation languages: linear discriminant functions, decision trees and instance-based classifiers. Our choice of these three representation languages stems from the results of empirical comparisons, which illustrate that these three languages produce classifiers of widely differing accuracy for different data sets; each of the three was selectively superior for some data sets. An analytical evaluation of the three representation languages provided further evidence that for many data sets the three would form widely differing classifiers. In Sections 4.2 and 5.2 we discuss in more detail the differences and similarities among these three languages.

The remaining three abilities, listed above, drove the development of the rule base and search strategy of MCS. In order to avoid a prohibitive amount of time to select a best test for each node in the hybrid classifier, the rules were designed to cut off unpromising avenues of search. For example, there is no need to explore various linear combinations of the features if there are strong indications that a single univariate test is a better test.

To arrive at an implementation that subsumes its primitive components and selects a best classifier for each subspace of the instance space requires careful generation of the rule set. This trial and error process starts with a basic knowledge of the strengths and weaknesses of each of the included primitive algorithms. The representation languages included in MCS had the added benefit of being three of the most extensively used languages in both machine learning and statistics. This meant that a body of knowledge about their biases existed prior to the implementation of MCS. We were able to draw on this body of knowledge to begin the formation of the rule base. Building on this basic knowledge we then began a trial and error process for generating and debugging rules that we describe in Section 4.3.2.

4.2 Model Classes

MCS combines three primitive representation languages (model classes): linear discriminant functions, decision trees and instance-based classifiers. For each model class there are many different algorithms for searching for the classifier that best fits the data. In this section we first describe the algorithms included in MCS. Specifically, we describe each representation language, its corresponding search bias, and how different types of data (numeric and symbolic) and missing values are handled. We conclude with a discussion of the differences and similarities among the three model classes.

4.2.1 Univariate Decision Trees

A univariate decision tree is either a leaf node containing a classification or an attribute test, with for each value of the attribute, a branch to a decision tree. To classify an example using a decision tree, one starts at the root node and finds the branch corresponding to the value, for the attribute tested, observed in the example. This process repeats at the subtree rooted at that branch until a leaf node is reached. The resulting classification is the class label of the leaf.

There are many different decision tree algorithms [Moret, 1982; Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1986]. During tree construction, at each node, one wants to select a *test* that best divides the instances into their classes. There are many different partition-merit criteria that can be used to judge the "goodness of a split"; the most common appear in the form of an entropy or impurity measure. Breiman, et al. (1984), Quinlan (1986), Mingers (1989), Safavian and Landgrebe (1991), Buntine and Niblett (1992), and Fayyad and Irani (1992b) discuss and compare different partition-merit criteria.

MCS's approach to constructing decision trees is to use information theory to select a best attribute to place as a test at a node. An information theoretic metric measures the gain in information if attribute A_i is used to form a partition of the instances observed at a node. Such metrics aim to reduce the *impurity* of each resulting partition. The impurity of a partition is at a minimum if it contains elements of only one class. The impurity is at a maximum if all classes are equally represented in the partition [Breiman, Friedman, Olshen & Stone, 1984]. To find a best univariate test for a set of instances, MCS chooses a test that maximizes the Information-Gain Ratio metric [Quinlan, 1986]. In Appendix A we describe how to compute this measure.

One desires that a decision tree algorithm be able to handle both unordered (symbolic) and ordered (numeric) features. Univariate decision tree algorithms require that each test have a discrete number of outcomes. To meet this requirement, each ordered feature A_i is mapped to a set of unordered features by finding a set of Boolean tests of the form $A_i > b$, where b is in the observed range of A_i. MCS finds the value of b that maximizes the Information-Gain Ratio. To this end, the observed

values for A_i are sorted and the midpoints between class boundaries are evaluated [Quinlan, 1986; Fayyad & Irani, 1992].

An issue that must be addressed for any learning algorithm that handles real-world tasks is how to handle missing values in the data. During tree construction and when using the tree to classify an instance, MCS fills in missing values for a univariate test as follows: if a numeric attribute's value is missing it substitutes the sample mean [Sutton, 1988]; and if a symbolic attribute's value is missing then it substitutes the most frequently observed value [Quinlan, 1989].

4.2.2 Linear Discriminant Functions

To construct a linear combination test for a set of data, MCS represents the test as a linear threshold unit for two-class tasks and as a linear machine for multiclass tasks [Nilsson, 1965]. A linear threshold unit (LTU) is a binary test of the form $W^TY \geq 0$, where $\mathbf{Y}$ is an instance description (a pattern vector) consisting of a constant 1 and the n features that describe the instance. $\mathbf{W}$ is a vector of $n+1$ coefficients, also known as weights. If $W^TY \geq 0$, then the LTU infers that Y belongs to one class A, otherwise the LTU infers that Y belongs to the other class B. When an LTU is a test node in a decision tree it has two branches, one for each class.

A linear machine (LM) is a set of R linear discriminant functions that are used collectively to assign an instance to one of the R classes [Nilsson, 1965]. Let Y be an instance description (a pattern vector) consisting of a constant 1 and the n features that describe the instance. Then each discriminant function $g_i(Y)$ has the form W_i^TY, where W_i is a vector of $n+1$ coefficients. A linear machine infers instance Y belongs to class i if and only if $(\forall j, j \neq i)\ g_i(Y) > g_j(Y)$. For the rare case in which $g_i(Y) = g_j(Y)$ some arbitrary decision is made: our implementation of an LM chooses the smaller of i and j in these cases. When an LM is a test node in a decision tree it has R branches, one for each observed class in the instances.

Each feature included in a linear combination test must be numeric. To include symbolic features in linear combination tests, MCS maps each symbolic (unordered) feature to one or more numeric features [Hampson & Volper, 1986; Utgoff & Brodley, 1990]. For a two-valued symbolic feature, MCS simply assigns 1 to one value and -1 to the other. If the feature has more than two observed values, then each feature-value pair is first mapped to a propositional feature, which is TRUE if and only if the feature has the particular value in the instance [Hampson & Volper, 1986]. The two-valued propositional feature is then mapped to a numeric feature, where a value of TRUE is mapped to 1 and a value of FALSE is mapped to -1. This mapping avoids imposing any order on the unordered values of the feature. With this encoding, one can create linear combinations of both ordered and unordered features.

An LTU (LM) is biased toward concepts that are linearly separable. If, however, the space is not linearly separable then an LTU (LM), trained using the absolute error correction rule [Duda & Hart, 1973], can not represent the concept and as a result will misclassify some percentage of the instances. There are several training

procedures aimed at finding a good LTU (LM) even when the space is not linearly separable. One such procedure is the thermal training rule, which enables a linear threshold unit to converge to a set of boundaries using an annealing coefficient [Frean, 1990a]. Utgoff and Brodley (1992) have adapted this idea to a linear machine and we use their method to train linear machines in MCS. Our choice of this method is based on the results of an empirical comparison that illustrated that the thermal training rule performed better than several other alternatives across a range of different tasks [Brodley & Utgoff, in press].

In MCS, we add a new modification to this procedure to address the problem that the weights found by this rule depend on the order in which the instances are presented; a bad ordering can lead to an inaccurate classifier. To ameliorate this problem we apply the thermal training procedure ten times, ordering the instances differently each time. This produces ten LTUs (LMs), each with a different set of weights. MCS chooses the LTU (LM) that maximizes the Information-Gain Ratio metric. In Appendix A we describe how to compute this metric for linear combination tests.

In addition to finding a good set of weights, one wants to eliminate noisy and irrelevant features with the goal of increasing predictive accuracy. For most data sets it will be impossible to try every possible subset of the set of features because the number of possible combinations is exponential in the number of features. Therefore, some type of heuristic search procedure must be used. To select the terms to use with a linear discriminant function, MCS applies one of three search procedures: Sequential Backward Elimination (SBE) [Kittler, 1986], a variation of SBE, Dispersion Sequential Backward Elimination (DSBE), which uses the form of the function to determine which terms to eliminate [Utgoff & Brodley, 1991], and Sequential Forward Selection (SFS) [Kittler, 1986]. The choice of which of these search biases to apply is determined dynamically during learning, depending on the hypotheses that have already been formed.

A Sequential Backward Elimination search is a top-down search method that starts with all of the features and tries to remove the feature that will cause the smallest decrease of some partition-merit criterion that reflects the amount of classification information conveyed by the feature [Kittler, 1986]. Each feature of a test either contributes to, makes no difference to, or hinders the quality of the test. An SBE search iteratively removes the feature that contributes least to the quality of the test. It continues eliminating features until a specified stopping criterion is met. To determine which feature to eliminate, the coefficients for i linear combination tests, each with a different feature removed, are computed. MCS selects the subset of the features that maximizes the Information-Gain Ratio. MCS searches as long as the accuracy of the current test based on i features is either more accurate or is not more than 10% less accurate than the accuracy of the best test found thus far, and two or more features remain to be eliminated. This heuristic stopping criterion is based on the observation that if the accuracy drops by more than 10%, the chance of finding a better test based on fewer features is remote [Brodley & Utgoff, in press].

A Sequential Forward Selection search is a bottom-up search method that starts with zero features and tries to add the feature that that will cause the largest increase of some partition-merit criterion. An SFS search iteratively adds the feature that results in the most improvement of the quality of the test. It continues adding features until the specified stopping criterion is met. During the process of adding features, the best linear combination test with the minimum number of features is saved. When feature addition ceases, the test for the decision node is the saved linear combination test. MCS applies the SFS search until either no more features remain to be added or the accuracy of an LTU (LM) decreases by more than 10% when a new feature is added. The resulting LTU is the LTU with the largest Information-Gain Ratio observed during the SFS search.

The Dispersion-Guided Sequential Backward Elimination (DSBE) search is a variation of the SBE algorithm, which uses the weights of the LTU (LM) to determine which features to eliminate [Brodley & Utgoff, in press]. DSBE selects the feature to remove that contributes the least to discriminability based on the magnitude of the weights of the LTU (or LM). This reduces the search time by a factor of n; instead of comparing n linear combination tests when deciding which of the n features to eliminate, DSBE compares only two linear combination tests (T_i to T_{i-1}). To be able to judge the relative importance of the features by their weights, we normalize the instances before training using the standard normal form (i.e., zero mean and unit standard deviation).

For an LTU test, we measure the contribution of a feature to the ability to discriminate by the magnitude of its corresponding weight. We choose the feature corresponding to the weight of smallest magnitude as the one to eliminate. For an LM test, we evaluate a feature's contribution using a measure of the *dispersion* of its weights over the set of classes. A feature whose weights are widely dispersed has two desirable characteristics. Firstly, a weight with a large magnitude causes the corresponding feature to make a large contribution to the value of the discriminant function, and hence discriminability. Secondly, a feature whose weights are widely dispersed across the R linear discriminant functions (R is the number of classes) makes different contributions to the value of the discrimination function of each class. Therefore, one would like to eliminate the feature whose weights are of smallest magnitude and are least dispersed. To this end, DSBE computes, for each remaining feature, the average squared distance between the weights of the linear discriminant functions for each pair of classes and then eliminates the feature that has the smallest dispersion. This measure is analogous to the Euclidean Interclass Distance Measure for estimating error [Kittler, 1986].

To handle missing values for linear combination tests, MCS fills them in using the sample mean. After normalization the sample mean of each feature is equal to zero. In a linear combination test this has the effect of removing the feature's influence from the classification, because a feature with a value of zero does not contribute to the value of the linear combination. The normalization information is retained at

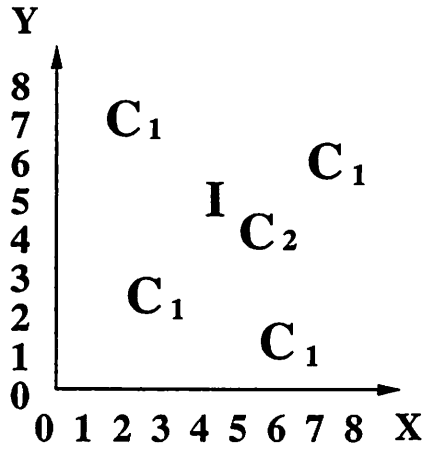


Figure 4.2 An instance-based classifier.

each node in the tree. During classification, this information is used to normalize and fill in any missing values of the instance to be classified.

4.2.3 Instance-Based Classifiers

An instance-based classifier (IBC) is a set of n distinct instances, each from one of m classes, that are used to assign an instance to one of the m classes. A simple IBC is the k-nearest neighbor (k-NN) classifier [Duda & Hart, 1973], which classifies an instance according to the majority classification of its k nearest neighbors. MCS employs a k-NN algorithm that stores each instance of the training data. To determine how near one instance is to another, MCS computes the Euclidean distance between the two instances. When an IBC is embedded in a decision tree as a test node, it has a branch for each of the observed classes.

Computing the distance between symbolic features, either requires a special procedure [Aha, 1990] or the conversion of the symbolic features to a numeric representation. To handle symbolic data, MCS encodes each symbolic variable as one or more propositional variables and then maps each propositional variable to a numeric variable as described in the previous section. In order that all variables contribute equally to classification decisions, MCS normalizes variables using standard normal form, i.e., zero mean and unit standard deviation.

In Figure 4.2 we show a simple instance-based classifier consisting of five instances, each described by two features (X and Y) and labelled as one of two classes (C_1 and C_2). If a 1-NN were used the resulting classification for the instance labelled I would be class 2. If a 3-NN were used then the IBC would label I as class 1. Our implementation of an instance-based classifier treats all features equally and assigns equal importance to the k nearest neighbors of the instance to be classified. (There

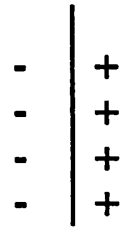


Figure 4.3 Representation similarities.

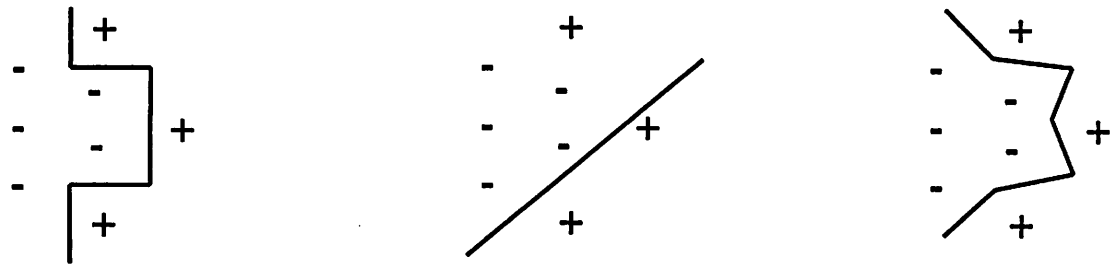
are methods for weighting the features in the distance calculation and for weighting the importance of the neighbor's votes [Duda & Hart, 1973; Aha, 1990].

To search for a best value of k, MCS estimates the classifier's accuracy with the following measure: for each instance in the training data, classify that instance using the remaining instances. The system selects the value of k that produces the highest number of correct classifications for the training data. The order in which values for k are evaluated is determined by the heuristic rules, which are described in the Section 4.3.2.

4.2.4 Differences and Similarities

Classifiers constructed from any one of these three languages each form a piecewise-linear partition of the given instance space. They differ in how (where) the boundaries may be placed in the space. A univariate test of feature F_i represents a decision boundary that is orthogonal to F_i's axis. A univariate decision tree defines a set of orthogonal decision boundaries that partition the instance space into a set of hyper-rectangular regions each labeled with a class name. A set of R linear discriminant functions defines a set of R regions in the instance space, separated by hyperplanes, each labeled with a different class name. An instance-based classifier defines a piecewise-linear partition of the instance space; the number of blocks is determined by the number and distribution of the instances, and the choice of k, which is the number of nearest neighbors to examine when classifying an unlabeled instance. The result is a set of regions, each labeled by a different class name, separated by piecewise-linear boundaries.

Given a data set, the placement of boundaries for a univariate test is restricted to being orthogonal, but which features are used and the placement of each boundary along a feature's axis is determined by the search bias. For a linear discriminant function, the search bias determines both the orientation and placement of the hyperplane decision boundary by learning the coefficients of each discriminant function. For an instance-based classifier, the orientation and placement of the piecewise-linear



a: Univariate decision tree partition

b: Linear discriminant function partition

c: k-NN classifier partition

Figure 4.4 Representation differences.

boundaries are determined by the distribution of the training instances and the choice of k.

In Figure 4.3 we show an instance space for which classifiers from each of the three model classes would define an identical partition of the instance space, given the goal of partitioning the instances into regions, each containing instances from only one class. For many data sets, classifiers from each of the three model classes will define different partitions. Figures 4.4a, 4.4b, and 4.4c illustrate the type of partition each model class might define for a simple instance space consisting of five negative and three positive examples of the concept to be learned. Which of these concept representations is best depends on where in the instance space the true concept boundary lies. MCS's rule-based search strategy addresses this problem by using feedback from the learning process to determine which of the three representation biases is best for the given instance space.

4.3 Search Strategy

MCS searches for a hybrid classifier from the available model classes using a hill-climbing search to select a test for each node in the tree-structured classifier. A set of heuristic rules is used to decide which model classes to try, which model classes to avoid, and to determine when a best test at each node has been found. The instance space is partitioned according to the chosen test, and the search is applied recursively to each resulting subset that contains instances from two or more classes. The general recursive procedure of MCS is shown in Table 4.1. After MCS has constructed a hybrid classifier that perfectly partitions the training instances into regions each labeled with a single class name, it applies a pruning algorithm to reduce the estimated error of the classifier as computed for an independent set of instances.

Table 4.1 General recursive algorithm

```
Form Classifier(instances)

IF instances from a single class
   THEN return(class)
   ELSE select the best algorithm to create a classifier
        partition the instances using the classifier
        for each partition call Form Classifier(partition)
```

One problem that can occur during MCS's search is when two tests (classifiers) appear equally good for the set of instances observed at a node in the tree. The current version of MCS differs from our original version [Brodley, 1993] in order to handle these situations better. If two tests appear equally good at a node, and do not perfectly partition the set of instances at that node, then MCS examines whether one defines a better *partition* of the data than the other. In a decision tree, test nodes have one of two functions, depending on their position in the tree. Test nodes whose children are each leaves serve as classifiers of the subspace defined by the tree above them. Internal test nodes (nodes for which at least one child is not a leaf node) partition the instance space into subspaces. For example, in Figure 4.1, the linear combination test (LCT) partitions the space into two subspaces and the IBC node serves as a classifier for one of the subspaces. The Information-Gain Ratio is a heuristic method judging the quality of a partition test. Another way to judge the quality of a partition test is to examine the accuracy of the subtree whose root is that test.

In cases for which two tests appear equally good, MCS builds a subtree, of depth one, for each of the two candidate tests. One *partition* test is judged better than another if its subtree's accuracy is higher than the other test's. MCS builds a subtree by constructing a classifier for each subset of the instances defined by the partition test. Which representation language and search biases are used to construct classifiers for each subspace depends on the context of the comparison. In our description of MCS's rules we detail how these choices are made. The two subtrees are then compared using the heuristic rules to determine which of the two partition tests to place at that node. In cases for which the two tests each partition the set of instances perfectly, MCS selects the simpler of the two tests. After one of the tests is selected, MCS discards its subtree.

Even with the addition of the depth-one search, MCS may still have difficulty selecting among tests that appear equally good. In particular, this can happen when the measures of the set of candidate tests yield conflicting results. For example, suppose that the information score of test T_1 is greater than that for T_2, but T_2's accuracy is higher than T_1's. In such cases it is unclear which test to choose. To

address these situations we have added a *global* model-class bias to MCS, which is determined automatically before MCS begins to construct a classifier. We describe how this global bias is selected in Section 4.3.1.

To address further the problem of not being able to distinguish which of a set of tests will result in a more accurate classifier, MCS retains the most accurate of the remaining alternative tests when it selects a test for a node in the tree. This alternative can be different from the selected test, because MCS's heuristic rules may select a test that is less accurate, but that makes the subspaces easier to learn. The decision of whether to replace the subtree rooted at that test in the tree with the best of the alternatives is performed during the pruning stage.

In the remainder of this section we first describe a method for choosing a global model-class bias for MCS. Next we describe the measures used in MCS's rules and how the rules were developed. We then describe MCS's heuristic rule base, focusing on a discussion of why certain data-set characteristics lead MCS to prefer one bias over another. Finally, we describe MCS's pruning strategy, which differs from traditional decision-tree pruning algorithms to take into account the fact that a hybrid decision tree has tests constructed from different representation languages.

4.3.1 Choosing a Global Model Class Bias

Before MCS begins its search for a hybrid classifier, it examines the data set to determine which of the homogeneous model classes leads to the most accurate classifiers for random subspaces of the data set. This model-class bias is used by MCS to help decide among a set of candidate tests when measures of the tests do not indicate clearly a best test.

To choose a global bias, we do the following ten times: randomly select one half of the training data and apply each of the primitive learning algorithms to the resulting subspace of the data. We evaluate each primitive algorithm using the remaining instances. If one of the model classes performs better than the rest, for the majority of the ten random subspaces, then we input this model class as the global bias for MCS. If the results of the analysis were mixed, then we do not specify a bias for MCS. In the description of the rules we describe how this global bias influences MCS's search strategy.

4.3.2 Heuristic Rule Base

The heuristic rules guide a hill-climbing search for a best test to place at a node in the hybrid classifier. The rules detect various characteristics of the data that lead MCS to prefer a test from one model class over other tests within the same or different model classes. At each stage during the search, MCS retains a set of candidate tests; initially this set is empty. During search, the heuristic rules determine when a best test has been found, whether further investigation of other model classes is needed, or whether to search further within one model class.

Each rule may compute one or more of the following measures to judge the quality of a candidate test: the Information-Gain Ratio, the accuracy, and whether a test *compresses* the data. The first two measures are computed for any test, whether it is a univariate test, a linear discriminant function, a k-NN classifier, or even an entire subtree. The Information-Gain Ratio and accuracy of a univariate test or a linear combination test are computed directly from the training instances. For a k-NN classifier, we employ a leave-one-out strategy for generating the class counts to compute the Information-Gain Ratio and accuracy of the classifier. Specifically, we classify each of the instances in the classifier using the remaining instances.

Our judgement of whether a test compresses the data is based on the Minimum Description Length Principle, which states that the best "hypothesis" to induce from a data set is the one that minimizes the length of the hypothesis plus the length of the exceptions [Rissanen, 1989]. The *codelength* of a classifier (the hypothesis) is the number of bits required to encode the classifier plus the number of bits needed to encode the error vector resulting from using the classifier to classify the instances. We say that a test *compresses* the data if the number of bits required to represent the test and its corresponding error vector is *less* than the number of bits required to represent the error vector of the instances. In the rules, we examine only univariate and linear combination tests for compression.[1] The details of how to compute the codelength of these two types of tests are described in Appendix B.

Before describing the rules in detail we first outline how the rule set was developed. Our method was a cyclical process of trial and error. We began with general ideas about the situations in which one model class would be preferred to another. These ideas were culled from the literature and from our own experience with these three model classes. For example, it is fairly well-known that instance-based classifiers perform well for data sets for which all the features are relevant. However, if many features are irrelevant then they are known to perform poorly [Aha, 1990]. Whenever possible we reference the original source of a rule in the description of the rules presented in Section 4.3.3.

We encoded our general heuristics into a set of rules such that no conflict resolution is required. We then used four data sets, Iris, Breast, Pixel and Heart Disease, which are described in Chapter 5 to debug the rules. MCS generates a detailed rule trace, which prints out the decision made at each step in the search at a node, all of the available measures of the set of data observed at the node, and measures of the classifiers that have already been tried. In Section 4.4 we present such a trace. After a run, we examined this detailed rule trace to decide whether MCS had made the correct decisions in the search. In particular, we looked for cases for which MCS produced a classifier that was less accurate than the best of its primitive components. We pinpointed where in the rule trace MCS made a decision that led it away from the best model class. We then altered the rules to ensure that the better decision

[1] Lack of compression is used to decide whether an instance-based classifier should be tried. Therefore the rules do not test whether an IBC compresses the data.

Table 4.2 Rule for when candidate set = {}:

IF (number instances < hyperplane capacity) OR (global-bias = U) THEN **fit**(U)
ELSE **fit**(LCT_n)

was made and re-ran MCS on that data set. Once we had corrected the problem for that data set, we ran MCS with the new rules across all four development data sets. If performance did not decrease for the other data sets then the rule(s) remained, otherwise we examined why the new rule(s) hindered performance and readjusted the rule set. This cyclical process involved many months of experimentation. In the following description of the rules we describe the general intuition behind each rule and in some cases give illustrative examples.

4.3.3 Description of the Rules

The description of the rules is organized by the model classes the search strategy has investigated, which we designate as the *candidate set.* In the description of the rules we use four symbols: **fit**(T) adds a new test, T, to the candidate set; **select**(T) terminates search and selects T from the candidate set to use as a test at that node; **delete**(T) removes T from the candidate set, because MCS has determined that it is not as good as the other candidates; and **examine-alt**(T) compares the accuracy of T to the alternative test (if it exists) and the chosen test, and retains the alternative with the highest accuracy. The rules are shown in Tables 4.2 through 4.10, and we spend the remainder of this section describing and motivating each rule.

Initially the candidate set is empty and MCS must decide where in the model space to begin the search. To this end, MCS examines the number of instances relative to the number of features that describe each instance. When the ratio of instances to features is small, either a univariate test (U) or an instance-based test (IBC) is preferred over a linear combination test (LCT). The rule shown in Table 4.2 determines whether to start the search with a univariate test or an LCT. (The rules shown in Table 4.3 address the case where an IBC would be preferred to a univariate test in the case of a small number of instances.) Specifically, if the number of training instances is fewer than twice the number of features used to describe each instance (the capacity of a hyperplane) [Duda & Hart, 1973] then MCS starts with a univariate test; otherwise the search begins with an LCT based on all n features. This rule is motivated by the observation that when there are too few instances relative to the number of features, then there are many possible orientations of the hyperplane that are consistent with the data, and not enough information to choose among the possibilities. This rule is overridden when the global model-class bias is toward univariate decision trees; in this case the first test formed is always a univariate test.

Table 4.3 Rules for when candidate set = {U}:

IF accuracy(U) = 100% THEN **select**(U)
ELSIF average difference in info of each feature to best is $< \epsilon$ THEN
  starting with U, **fit**(LCT_i) using SFS
ELSIF U does not compress the data THEN
  IF the number of instances < hyperplane capacity THEN **fit**(LCT_n)
  ELSE **fit**($IBC_{k=2}$)
ELSE **select**(U) AND recurse

This does not preclude the investigation of an LCT; it may however reduce search effort.

The rules shown in Table 4.3 determine whether the initial choice of a univariate test was appropriate or whether further exploration of other model classes is required. Recall that at this point the candidate set = {U}. If the univariate test perfectly partitions the data, then there is no need to search for a more complex test, and search halts. Otherwise MCS tries to determine if a univariate test representation is inappropriate by checking for the presence of one of two different data characteristics. The first characteristic is whether no single feature is superior to all the rest. In these cases a linear combination test may provide a better partition of the data. An example of such a situation is shown in Figure 4.5, which shows a two-dimensional instance space and the corresponding univariate decision tree, which approximates the hyperplane boundary, $x + y \leq 8$, with a series of orthogonal splits. In the figure, the dotted line represents the hyperplane boundary and the solid line represents the boundary defined by the univariate decision tree. This example illustrates the well known problem that a univariate test using feature F_i can only split a space with a boundary that is orthogonal to F_i's axis [Breiman, Friedman, Olshen & Stone, 1984].

In an earlier version of the rules we explicitly tested for such situations, but we found that by examining the the relative difference in the information score of the best feature to the other features we could catch such situations and save a substantial amount of time. If the average difference of the best U to each of the others is less than a threshold, ϵ, then MCS explores whether a linear combination test will do better. We have set ϵ to 0.20. To this end, MCS starts with a best feature (the one chosen for the univariate test) and performs an SFS search. Note that if the univariate test was initially chosen based on the global bias (indicated by the number of instances being greater than the capacity of a hyperplane), then the system fits an LCT based on all n features and uses the rules shown in Table 4.6 to decide where next to guide the search. The second characteristic is whether a univariate test compresses the data. It is known that the information-theoretic measure does not provide reliable results if the number of training instances is too small [Quinlan, 1987; Aha, 1990]. In these

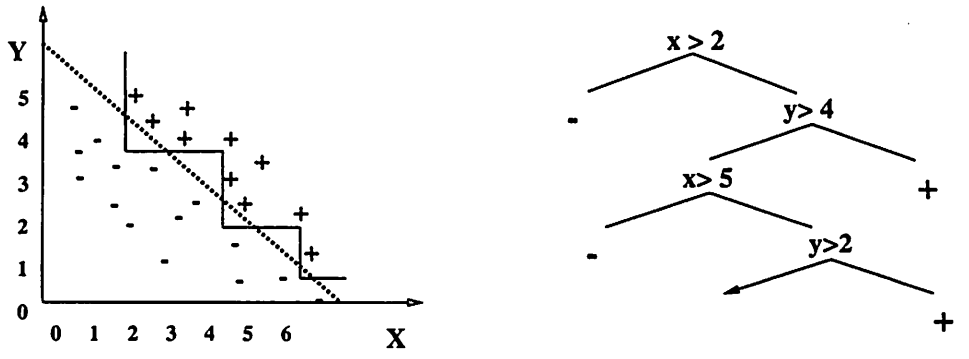


Figure 4.5 An example instance space; "+": positive instance, "-": negative instance and the corresponding univariate decision tree.

Table 4.4 Rule for when candidate set = $\{LCT_n\}$:

IF the instances are linearly separable THEN **fit**(LCT_{n-1}) using DSBE
ELSE **fit**(U) AND **examine-alt**(LCT_n)

cases, an instance-based classifier is more likely to be appropriate because there is no minimum number of instances required to form an IBC. If neither of the above two characteristics is observed then there is no indication that a better model class will be found, and the univariate test is selected.

If the first model class tried (as determined by the rule shown in Table 4.2) was an LCT based on all n of the input features, then MCS applies the rule shown in Table 4.4 to decide whether to continue searching within the class of linear combination tests or whether to explore other model classes. Linear combination tests are ideal for linearly-separable instance spaces. MCS looks for this characteristic by examining the accuracy of a linear combination test that has been fit to the data. If the LCT based on all n features is 100% accurate then we know that the space is linearly separable, and all that remains to be done is to search for the smallest set of features to include in the test while retaining linear separability. This is achieved by a sequential backward elimination search procedure (DSBE) that eliminates features one by one using the magnitudes of the corresponding weights to determine which features to eliminate [Brodley & Utgoff, in press].

Even if a test is not 100% accurate, the space still may be linearly separable if some of the features are removed from the linear combination test; removing noisy features from the LCT will increase its accuracy. However, given no certainty of this being the case, the system fits a univariate test to the data and then uses the additional information that this will provide to determine where next to direct the search.

Table 4.5 Rules for when candidate set = $\{LCT_i, LCT_{i-1}\}$:

IF there is more than one feature in LCT_{i-1} THEN
  IF accuracy(LCT_{i-1}) = 100% THEN **fit**(LCT_{i-2}) AND **delete**(LCT_i)
  ELSIF LCT_i compresses data THEN **select**(LCT_i) AND **examine-alt**(LCT_{i-1})
  ELSE **fit**(U) AND **delete**(LCT_{i-1})
ELSE **select**(LCT_1 OR LCT_2 based on accuracy)

Table 4.6 Rule for when candidate set = {U,LCT_n}:

IF info(U) $\geq$ info(LCT_n) AND accuracy(U) $\geq$ accuracy(LCT_n) THEN
  fit LCT_i using SFS, starting with U AND **delete**(LCT_n)
ELSE **fit**(LCT_i) using SBE

If the accuracy of an LCT based on all n features indicated that the instance space was linearly separable, then the application of the rule shown in Table 4.4 created the candidate set = $\{LCT_i, LCT_{i-1}\}$, where $i = n$. The rules in Table 4.5 determine whether further feature elimination should take place and if not, what to do next. MCS continues to eliminate features as long as there are more than two features in the smaller of the two linear combination tests and the accuracy of the smaller of the two is 100%. If the system eliminates features until there is only one remaining, then it selects this test. Otherwise it checks to see whether the best LCT_i does not compress the data, indicating that it may be too complex for the data. In this case the system fits a univariate test to see if it will compress the data. This rule is motivated by the observation that although an LCT may perfectly partition the *training data*, it may be overfitting to noise in the instance space. Building a subtree of univariate tests allows MCS to do more fine-grained pruning.

The rule shown in Table 4.6 handles the point in search where both a best univariate test and an LCT based on all n features have been added to the candidate set. If both the information score and the accuracy of the univariate test are higher than the LCT, then there is evidence that the best test is univariate, but MCS explores the option that a small LCT may be a better test, by constructing an LCT using SFS, starting with the feature in the univariate test. This rule is based on results of previous research that illustrate that the bias of an SFS search can lead to a better LCT than an SBE search for some data sets [Brodley & Utgoff, in press]. Otherwise, there is no reason to believe that a better LCT could not be found, and the system searches for one using SBE.

Table 4.7 Rules for when candidate set = {U,LCT_i}:

IF ((global-bias = Uni) AND
(info(U) $\geq$ info(LCT_i) OR accuracy(U) $\geq$ accuracy(LCT_i))) OR
((global-bias = lct) AND
(info(U) $\geq$ info(LCT_i) AND accuracy(U) $\geq$ accuracy(LCT_i))) OR
((global-bias = none OR ibc) AND (info(U) $\geq$ info(LCT_i)) THEN
examine-alt(LCT_i)
IF U compresses the data THEN **select**(U) AND recurse
ELSE **fit**($IBC_{k=2}$) AND **delete**(LCT_i)
ELSE **examine-alt**(U)
IF LCT_i compresses the data THEN **select**(LCT_i) AND recurse
ELSE **fit**($IBC_{k=2}$) AND **delete**(U)

If the candidate set contains a univariate test and an LCT, based on i features formed using either an SBE or SFS search (candidate set = {U,LCT_i}), then MCS's next action depends on whether there is a global model-class bias. If the bias is toward univariate decision trees, then the system forms a univariate subtree of depth one. This is because in many cases a single univariate test will not have as high an information score or accuracy as an LCT. Because an LCT is more complex than a U, a fairer comparison is an LCT to a univariate tree of depth one. In addition, MCS biases the information score and accuracies of the two tests by their complexity. Specifically it uses the following weight: $\frac{T-v}{T+v}$, where T is the number of instances and v is the number of features in the test [Quinlan, 1993]. This has the effect of penalizing the more complex test. MCS uses these weighted information scores and accuracies in the rules shown in Table 4.7. If the bias is not for univariate trees but the two tests are close in either information score or accuracy (within 10% of one another), then the system creates subtrees for both the LCT and the univariate test, for which each of the children can be classifier from any of the three model classes. Rather than choose erroneously between the two models, MCS is exploring which of the two tests make the subspaces easier to learn. To construct the subtree, MCS chooses a classifier for each subspace from the set of a best univariate test, an LCT test and a k-NN (k=1) test. The chosen test is the one that maximizes the Information-Gain Ratio.

MCS compares the information score and accuracy of the two tests (possibly subtrees at this point) to decide whether to select one of the tests or whether an instance-based classifier should be examined. MCS prefers U to LCT if one of the following three cases is true: the global bias is toward univariate decision trees and either the accuracy or the information score of U is better; the global bias is toward

Table 4.8 Rules for when candidate set = $\{LCT_i, IBC_k\}$:

IF accuracy(IBC_k) $\geq$ accuracy(LCT_i) THEN
  fit(IBC_{k+1}) AND **delete**(LCT_i)
ELSIF ((global-bias = IBC) OR
  (more than half the features remain in LCT_i) OR
  (accuracy(LCT_i) - accuracy(IBC_k) < 10%)) THEN
  Until accuracy(IBC_k) < accuracy(IBC_2) increase k
  IF accuracy(IBC_k) $\geq$ accuracy(LCT_i) THEN
    select(IBC_k) AND **examine-alt**(LCT_i) AND recurse
  ELSE **select**(LCT_i) AND **examine-alt**(IBC_k) AND recurse
ELSE **select**(LCT_i) AND **examine-alt**(IBC_k) AND recurse

linear combination tests and both the information score and accuracy of U are higher than the LCT; and if there is not global bias or it is toward instance-based classifiers, then only the information score is considered. If U is preferred, then the system examines whether U compresses the data. If it does then the search halts and U is selected. If U does not compress the training data, then MCS examines whether an instance-based classifier is a better model. Note that MCS retains the ability to select the LCT during pruning by examining whether it would make a good alternative. If the LCT is preferred over U, then the system performs the same compression analysis, making a decision of whether to select the LCT or examine an IBC using the same criteria as for the univariate test case.

If the candidate set = $\{LCT_i, IBC_{k=2}\}$, then MCS decides whether further exploration of IBC tests is required using the rules shown in Table 4.8. If the global bias is not for instance-based classifiers, then the accuracies of the LCT and IBC are weighted by their complexity as described above. If the accuracy of the IBC is higher than the LCT's, then MCS explores the IBC model class, by increasing the value of k (the number of nearest neighbors to examine during classification). In addition, MCS examines whether the LCT would make a good alternative. If the IBC's accuracy is not higher than the LCT's accuracy, but one of the following three cases is true then MCS explores higher values of k for the IBC: the global bias of MCS is toward an IBC; more than half the features remain in the LCT; or there is a less than 10% difference in the accuracies of the IBC and the LCT. The second case captures situations in which IBCs do well because all (or most) of the features are relevant. The third case catches situations for which a 1-NN is a poor choice but a k-NN is a good choice ($k > 2$). MCS evaluates an IBC for increasing values of k until the accuracy of the IBC drops lower than that of a 2-NN (equivalent to a 1-NN) classifier. At this point MCS selects between the IBC or the LCT. Finally, if the LCT is the better test, then it selects LCT and retains the IBC as a possible alternative, but does not investigate

Table 4.9 Rule for when candidate set = {U,IBC$_k$}:

IF accuracy(U) $\geq$ accuracy(IBC$_k$) THEN
  select(U) AND **examine-alt**(IBC$_k$) AND recurse
ELSIF accuracy(IBC$_k$) < 100% THEN **fit**(IBC$_{k+1}$) AND **delete**(U)
ELSE **select**(IBC$_k$) AND **examine-alt**(U) AND recurse

Table 4.10 Rule for when candidate set = {IBC$_k$,IBC$_{k+1}$}:

IF accuracy(IBC$_{k+1}$) > accuracy(IBC$_k$) THEN **fit**(IBC$_{k+2}$) AND **delete**(IBC$_k$)
ELSE **select**(IBC$_k$) AND recurse

other values of k. Note that only accuracy is used here, because in most cases when an IBC is selected as a test no subtree will be needed.

When search has led MCS to believe that the best candidate test is a univariate test or an IBC, MCS applies the rules shown in Table 4.9. If the global bias is for univariate tests, then MCS grows a subtree of depth one for the univariate test in which each node of the subtree is a univariate test. MCS then biases the univariate subtree and the instance-based classifier by their complexities if the global model-class bias is for univariate trees. The system selects U if its accuracy is higher than that of the IBC. Otherwise it explores the IBC model class. The test not chosen is examined to see whether it would make a good alternative.

To reach the point in the search where the two best candidate tests are each instance-based classifiers, the accuracy of IBC$_{k=2}$ was higher than either a univariate test, an LCT, or both. The rule shown in Table 4.10 handles the situation in which MCS has decided that the best model class is instance-based classifiers and is now searching for the value of k that leads to the best heuristic accuracy on the training data using the leave-one-out scheme described in Section 4.2 above.

4.3.4 Pruning Hybrid Classifiers

To address the problem of *overfitting* in the hybrid formalism, the system prunes back the classifier as computed for an independent set of instances [Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1987]. Overfitting occurs when the classifier overfits the training data at the expense of generalization, which can occur in both noisy and noise-free domains. In the case of domains that contain *noisy instances* (instances for which the class label is incorrect or some of the feature values are

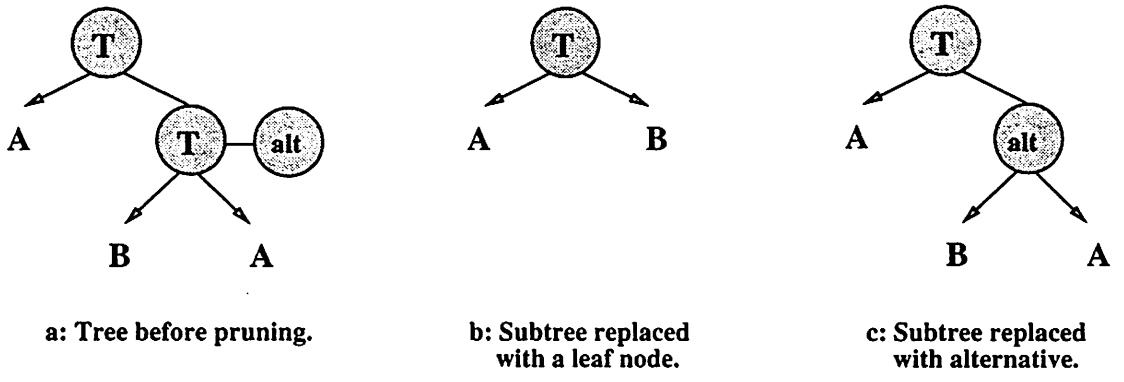


Figure 4.6 Choices of hybrid classifier pruning method.

incorrect) finding the classifier that maximizes the accuracy for the *training data* may overfit to the noise in the training data, and subsequently perform poorly for previously unseen instances.

Our approach to pruning a hybrid classifier differs from the traditional approach to pruning decision trees. Traditionally, each non-leaf subtree is examined to determine the change in the estimated classification error if the subtree were replaced by a leaf labeled with the majority class of the training examples used to form a test at the root of the subtree. The subtree is replaced with a leaf if this lowers the estimated classification error; otherwise, the subtree is retained.

A hybrid classifier that mixes different model classes has test nodes of varying complexity. This can cause problems when deciding whether to replace a test node with a class label; a complex test may overfit the training data, but removing it may decrease the accuracy of the classifier. In such cases, one wants to replace the test with one that is less complex. In addition, as described in Section 4.3, test nodes in a decision tree have one of two functions: to partition or classify the instances observed at that node. During the tree construction phase, MCS saves the most accurate candidate test if it was not chosen as the test for that node (Currently, MCS does not bias the storage of alternatives toward less complex tests. An issue for future research is to examine the utility of saving both the most accurate alternative and a less complex test for consideration during pruning.) During pruning, in addition to deciding whether to retain a subtree or replace it with a leaf, our approach examines whether replacing the subtree with the alternative would result in a lower estimated classification error than either of the two other choices. In Figure 4.6 we show these three options: Figure 4.6a illustrates the option of retaining the original tree (the alternative would be deleted); Figure 4.6b illustrates the option of replacing a subtree with a leaf node; and Figure 4.6c illustrates the option of replacing a subtree with the saved alternative.

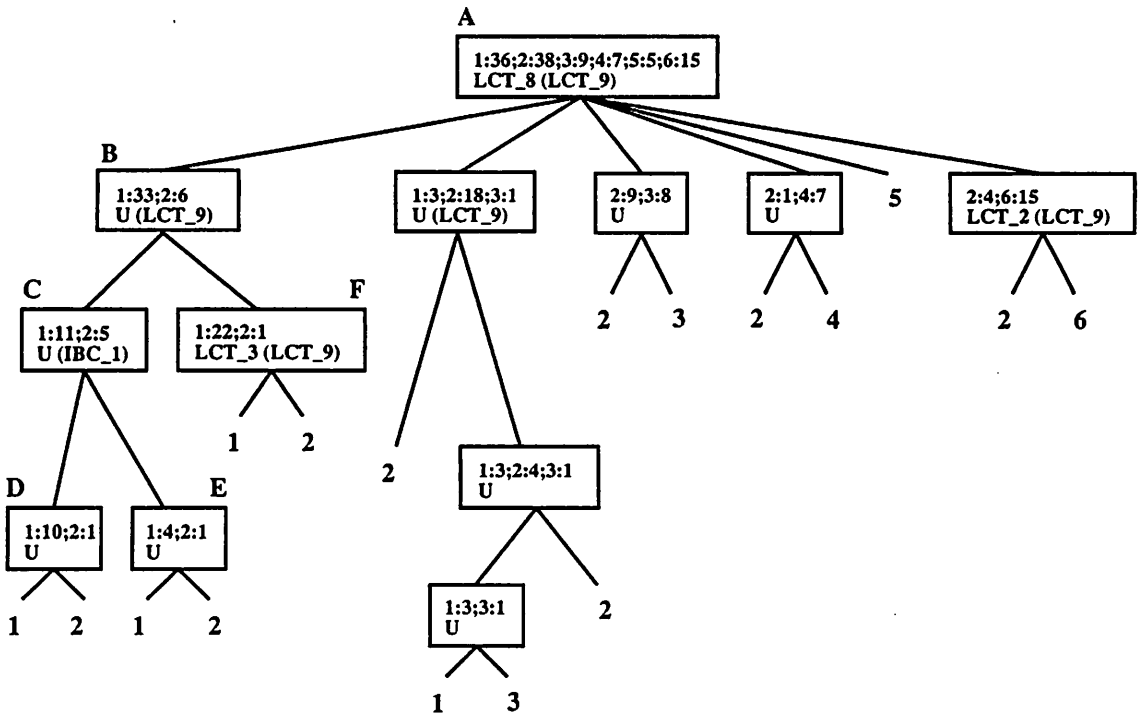


Figure 4.7 Hybrid constructed by MCS for the Glass data set.

4.4 An Example

To illustrate how MCS's search strategy creates a hybrid classifier, we present a detailed example of MCS in action. Our example is from an actual run of MCS on the Glass Recognition task. For this data set, the task is to identify a glass sample taken from the scene of an accident. The 213 examples were collected by the Home Office Forensic Science Service at Aldermaston, Reading, UK. There are six types of glass and each instance is described by nine numeric features. We have split the discussion of our example into two parts. Section 4.4.1 describes how the hybrid tree was constructed from the training instances and Section 4.4.2 describes how the hybrid tree is pruned back to avoid overfitting to the training data.

4.4.1 Tree Construction

In Figure 4.7 we show the hybrid tree constructed by MCS before pruning. In Table 4.11 we show a trace of MCS's search for part of the tree (nodes A – F). Each node in Figure 4.7 shows the class distribution of instances, the type of test selected by MCS, and the alternative test (if one was chosen). For example to construct Node A, 110 training instances were used: 36 of class 1, 38 of class 2, 9 of class 3, 7 of class 4, 5 of class 6, and 15 of class 6. The test selected by MCS was a linear combination

test based on eight features and the alternative was a linear combination test based on nine features. Node A has a branch for each linear discriminant function (recall that a linear machine has a linear discriminant function for each class).

In Table 4.11 we show the actions taken by MCS at each step in the search for Nodes A – F. At each node the candidate set starts out empty. Depending on the number of instances, MCS fits either a linear combination test based on all nine features or MCS finds a best univariate test. Note that for this data set, the hyperplane capacity is 18 (twice the number of features).

To create Node A, after MCS has fit an LCT based on all nine features, it examines the LCT and determines that it is not 100% accurate. The next step is to find a best univariate test. At this point the LCT based on all nine features is stored as an alternative test as it is the most accurate test observed thus far. MCS next decides to search for a better LCT, using SBE, because the Information-Gain Ratio of the LCT based on nine features is higher than that of the univariate test. (For an explanation of why the Information-Gain Ratios are negative, see Appendix B). The SBE algorithm returns an LCT based on eight features. At this point the LCT is selected from the candidate set because it has the highest information score and because it compresses the data.

The first three steps of the search for a test for Node B are the same as for Node A. At the point in the search when the candidate set contains the best univariate test and a linear combination test based on eight features[2] found by the SBE algorithm, MCS decides that a depth search is necessary because the univariate test and LCT test appear almost equally good. Consequently MCS creates a subtree of depth one for each of these tests and then compares the subtrees. MCS determines that the subtree whose root is a univariate test is the better of the two. However, because the univariate test does not compress the data, MCS investigates an instance-based classifier. MCS finally selects the univariate test because its subtree is 100% accurate whereas the IBC is not.

To create Node C, MCS first finds a best univariate test. Seeing that the univariate test does not compress the data, MCS next tries an IBC. Because the accuracy and the information score of the univariate test are better than those of the IBC, MCS selects the univariate test. However, MCS retains the IBC as an alternative, because the difference in its accuracy from the univariate test is less than 10%.

Nodes D and E have similar search histories. The first step in the search is to fit a univariate test, which is then selected because it is 100% accurate.

To create Node F, MCS first finds an LCT based on all nine features. Next MCS tries to find an LCT based on fewer features, because the LCT based on all nine features is 100% accurate. The result of this search is an LCT based on three features. This smaller LCT does not compress the data, leading MCS to find a best univariate test. Since the univariate test is not 100% accurate, MCS selects the LCT based on three features.

[2]It is a coincidence that SBE found LCT based on eight features for each of Node A and B.

Table 4.11 Partial trace of MCS's search strategy

Node	Candidate Set	Action	Reason
A	{}	fit $LCT_{n=9}$	110 > hyperplane capacity
	$\{LCT_{n=9}\}$	fit U	accuracy(LCT_n) < 100%
		select alt $LCT_{n=9}$	
	$\{U,LCT_{n=9}\}$	fit LCT_i	info(LCT_9=-.396) > info(U=-1.656)
	$\{U,LCT_{i=8}\}$	select $LCT_{i=8}$	info(LCT_8=-.304 > info(U=-1.656)
B	{}	fit $LCT_{n=9}$	39 > hyperplane capacity
	$\{LCT_{n=9}\}$	fit U	accuracy(LCT_n) < 100%
		select alt $LCT_{n=9}$	
	$\{U,LCT_{n=9}\}$	fit LCT_i	info(LCT_9=-.594) > info(U=-532), but accuracy(LCT_9=94.8)>accuracy(U=84.6)
	$\{U,LCT_{i=8}\}$	depth search	U and LCT_8 close
		fit $IBC_{k=1}$	info(S-U=0.0) > info(S-LCT_8=-0.167) but U does not compress
	$\{U,IBC_{k=1}\}$	select U	accuracy(S-U=100) > accuracy(IBC=89.7)
C	{}	fit U	16 < hyperplane capacity
	{U}	fit $IBC_{k=1}$	U does not compress data
	$\{U,IBC_{k=1}\}$	select U	accuracy(U=87.5) > accuracy(IBC=81.2) and info(U=-.589) > info(IBC=-.851)
		select alt $IBC_{k=1}$	acc(IBC) is within 10% of acc(U)
D	{}	fit U	11 < hyperplane capacity
	{U}	select U	accuracy(U) = 100%
E	{}	fit U	5 < hyperplane capacity
	{U}	select U	accuracy(U) = 100%
F	{}	fit LCT_n	23 > hyperplane capacity
	$\{LCT_{n=9}\}$	fit LCT_i	accuracy(LCT_n) = 100%
	$\{LCT_{i=3}\}$	fit U	$LCT_{i=3}$ does not compress the data
	$\{U,LCT_{i=3}\}$	select $LCT_{i=3}$	accuracy(LCT_i=100) >accuracy(U=95.7)

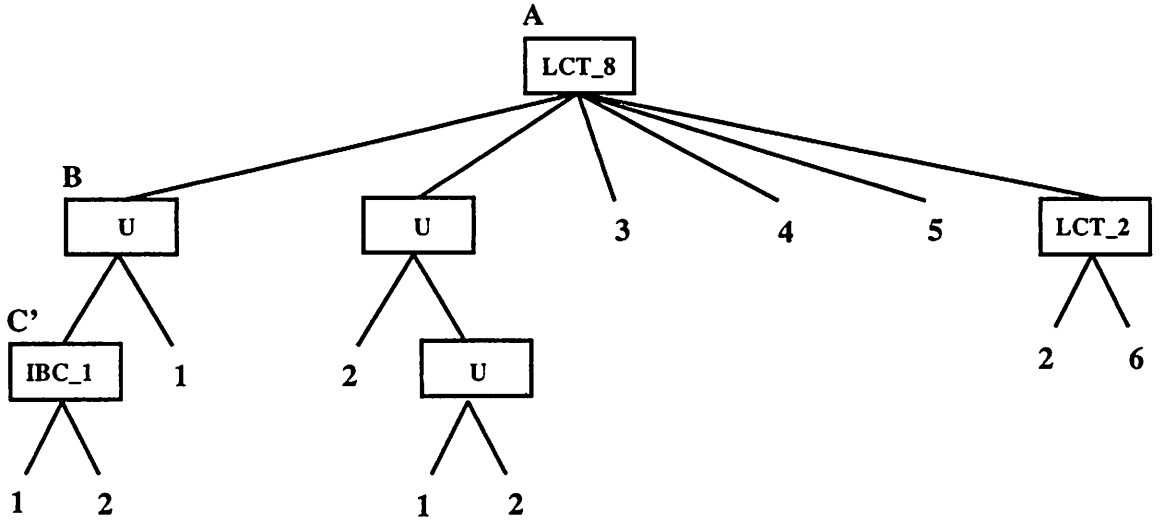


Figure 4.8 Hybrid produced by MCS after pruning.

4.4.2 Tree Pruning

In Figure 4.8 shows the tree for the Glass Recognition data set after pruning. In Table 4.12 we show the information considered by MCS to make its pruning decisions for Nodes C–F in the order in which the decisions were made. The first test node considered for pruning is Node D (see Figure 4.7). Of the seven pruning instances, the original test classified three correctly. There was no alternative test at Node D and MCS decided to replace the test with leaf labelled with class 1, because it did not reduce the accuracy of the node.

At Node E, no pruning instances were observed. MCS replaces such test nodes with a leaf labelled with the class observed most frequently in the training data, which in this case was class 1.

To decide whether to prune Node C, MCS compares the classification accuracy of the original univariate test, the alternative IBC test and with the maximum class. The highest count of correct classifications was for the alternative test and MCS replaces the original test at node C with its alternative, producing the node labelled C' in Figure 4.8.

At Node F, both the alternative test and replacing the original test with the maximum class would result in the same accuracy for the pruning data, which is higher than the original test's accuracy. Therefore, MCS replaces the test with class 1, which is the maximum class.

Table 4.12 Partial trace of MCS's pruning strategy

Node D: 7 instances
Original Test = U, CC = 3
Alternative = none
Max Class = 1, CC = 3
Decision: replace with leaf labelled with class 1

Node E: 0 instances
Original Test = U, CC = 0
Alternative = none
Max Class = none, CC =0
Decision: replace with leaf labelled with class 1

Node C: 7 instances
Original Test = U, CC = 3
Alternative = $IBC_{k=1}$, CC = 5
Max Class = 1, CC = 3
Decision: replace with alternative

Node F: 12 instances
Original Test = LCT_3, CC = 9
Alternative = LCT_9, CC = 10
Max Class = 1, CC = 10
Decision: replace with leaf labelled with class 1

CHAPTER 5

EVALUATION

This chapter provides an evaluation of MCS. The evaluation is split into two parts: Section 5.1 provides an empirical evaluation of MCS's performance across a variety of learning tasks and Section 5.2 discusses the representational strengths and limitations of MCS. In Section 5.3 we summarize the results of the evaluation.

The empirical evaluation has several objectives. Firstly, we want to determine whether our predictions of the best bias based on data-set characteristics, computed during search, works well in practice. Secondly, we want to know whether there exist real-world data sets whose underlying concept is best represented by a hybrid classifier. Thirdly, we evaluate various aspects of MCS's search strategy to understand the contribution each makes to MCS's performance.

Our analysis of the representational biases of MCS is aimed at gaining a clear understanding of the strengths and weaknesses of the primitive model classes in MCS and how representational ability expands when the model classes are mixed in a tree-structured hybrid classifier. The analysis examines the similarities and differences of instance-based, linear discriminant and univariate decision tree classifiers from a representational point of view. Building on this analysis we examine how the space of hypotheses that can be represented changes when these three model classes are combined in a hybrid.

5.1 MCS's Rule Based Search Strategy

In this section we present the results of several experiments that illustrate that the rules in MCS choose an appropriate learning bias using characteristics of the data set, computed from feedback from the learning process. Our experiments were designed to test the two hypotheses put forth in Chapter 1. The hypotheses were:

1. *Domain independent knowledge about characteristics in the form of feedback from learning can effectively guide an automatic algorithm selection search.*

2. *Our knowledge-based search strategy for finding a hybrid classifier will produce a classifier that is never worse than, and for some data sets is better than, any homogeneous classifier produced by its primitive components.*

We report the results of four comparisons. The first compares MCS to its primitive components. The results, reported in Section 5.1.3 illustrate that MCS is sometimes more accurate and is never less accurate than each of its primitive components across a variety of data sets, demonstrating that the heuristic rule-based search strategy solves the selective superiority problem for these algorithms.

In Section 5.1.4 we report the results of a comparison of MCS to three other hybrid classifier construction algorithms. Each of the three applies all of the primitive algorithms in the construction of each test node in the tree. The results of the comparison illustrate that merely increasing the search space by permitting hybrids will not increase accuracy. MCS's knowledge-based approach is demonstrated to be both more accurate and less time-consuming than the alternative three hybrid methods.

The last two experiments were designed to evaluate that contribution that considering alternatives during pruning and searching one ply deeper make to MCS's performance. In Section 5.1.5 we report the results of a comparison of MCS run with and without pruning alternatives. The results show that in some cases pruning alternatives increase MCS's performance. In Section 5.1.6 we report the results of a comparison of MCS run with and without the ability to search one ply deeper (when the rules indicate that it should). The results indicate that allowing MCS to search one ply deeper does not improve accuracy by a statistically significant amount. However, taken together these two aspects of MCS's search do improve performance significantly.

5.1.1 Data Sets

In this section we describe the data sets used in our empirical comparisons. We chose these data sets because they represented a cross-section of different domains and different types of characteristics. In addition, as we will see in Section 5.1.3 this test suite is not biased toward one of the primitive homogeneous algorithms; each of the primitive algorithms is selectively superior for at least 20% of the data sets. The Breast Cancer, Heart Disease, Iris Plants, and Pixel data sets were used for rule development. The results of an earlier version of MCS for the Hepatitis, LED-7 Digit Recognition, Road Segmentation, Congressional Votes and Vowel Recognition data sets were reported in Brodley (1993).

Breast Cancer: The breast cancer data consists of 699 instances, described by nine numeric features. The class of each instance indicates whether the cancer was benign or malignant [Mangasarian & Wolberg, 1990].

Congressional Votes: In this domain the task is to classify each of 435 members of Congress, in 1984, as Republican or Democrat using their votes on 16 key issues. There are 392 values missing.

Diabetes: The task is to decide whether a patient shows signs of diabetes according to World Health Organization criteria. Each of 768 instances is described by eight numeric features.

Glass Recognition: For the Glass data set, the task is to identify a glass sample taken from the scene of an accident as one of six types of glass using nine numeric features. The 213 examples were collected by B. German of the Home Office Forensic Science Service at Aldermaston, Reading, UK.

Heart Disease: The Heart data set consists of 303 patient diagnoses (presence or absence of heart disease) described by thirteen symbolic and numeric attributes [Detrano, Janosi, Steinbrunn, Pfisterer, Schmid, Sandhu, Guppy, Lee & Froelicher, 1989].

Hepatitis: The task for this domain is to predict from test results whether a patient will live or die from hepatitis. There are 155 instances, each described by 19 features (both numeric and symbolic features). There are 167 values missing in this data set.

Iris Plants: Fisher's classic data set [Fisher, 1936], contains three classes of 50 instances each. Each class is a type of iris plant. Each instance is described by four numeric attributes.

Landsat: The task for this domain is to predict the type of ground cover from satellite images. Each of 1000 instances is described by seven features (the channels) and labeled with one of four types of ground cover.

Lymphography: This data set consists of 148 instances, each described by nineteen attributes and labeled as one of four classes.

LED-7 Digit Recognition: The data for the LED-7 digit recognition problem consists of ten classes representing whether an LED display shows a 0-9. Each of seven Boolean attributes has a 10% probability of having its value inverted. There are 500 instances. Note that this is not the version of the data set reported in Breiman et al (1984), for which the Bayes optimal rate is known to be 74%.

LED-24 Digit Recognition: The data for the LED-24 digit recognition problem consists of ten classes representing whether an LED display shows a 0-9. Each of seven Boolean attributes has a 10% probability of having its value inverted. The remaining 17 attributes are irrelevant. There are 200 instances.

Liver Disorder: The task for this domain is to determine whether a patient has a propensity for a liver disorder based on the results of six blood tests. There are 353 instances.

Pixel: In the pixel segmentation domain the task is to learn to segment an image into one of seven classes. Each of the 3210 instances is the average of a 3x3 grid of pixels represented by nineteen low-level, real-valued image features.

Road Segmentation: The data come from four images of country roads in Massachusetts. Each instance represents a 3X3 grid of pixels described by three color and four texture features. The classes are road, road-line, dirt, gravel, foliage, trunk, sky, tree and grass. There are 2056 instances in this data set and 105 values are missing.

Vowel Recognition: The task is to recognize the eleven steady-state vowels of British English independent of the speaker. There are ten numeric features describing each vowel. Eight speakers were used to form a training set, and seven different speakers were used to form an independent test set. Each of the 15 speakers said each vowel six times creating 528 instances in the training set and 462 in the test set. For runs using this data set we retained the original training and test sets.

Waveform: This data set originates from Breiman et al (1984). In this version of the data set there are 300 instances each described by forty continuous-valued attributes.

5.1.2 Experimental Method

In this section we describe the experimental method used in each of the experiments reported in this chapter. In each experiment we compare two or more different learning methods across a variety of learning tasks. For each learning method, we performed ten runs on each data set.[1] For each run we split the original data randomly into 70% training data, 20% pruning data and 10% testing data. To ensure that the distribution of instances across the classes of a data set is the same in the training, pruning and test sets, we first sorted the data into their classes. We then dealt the instances out randomly to the training, pruning and test sets in the specified proportions (70, 20 and 10). Each method in a comparison was run using this partition.[2]

To estimate the accuracy of classifiers produced by each method, we average, for each method, the results of the ten runs. In the experiments we report both the sample average and standard deviation of each method's classification accuracy for the independent test sets. To determine the significance of the differences between

[1]For the Vowel data set, we used the original training and test sets and therefore did not perform multiple runs.

[2]Our goal in ensuring an even distribution of classes among the training, testing and pruning sets was to reduce variation in performance across different runs. We wanted both the data used for training and testing to mirror the distribution of the entire data set.

two learning methods we used paired t-tests. Because the same random splits of each data set were used for each method, the variances of the errors for any two methods are each due to effects that are point-by-point identical.

One learning algorithm, Thermal Training, has a random component: the resulting linear combination test's weights depend on the order in which the instances are observed. To eliminate this effect when comparing MCS to the primitive class of linear discriminant functions, we ran both of these algorithms with the same random seed. Therefore, if MCS determines that a single linear combination test (LCT) is the best classifier for the data, then the linear combination test's weights will be identical to those produced by running the primitive learning algorithm. We used the same random seed for each of the hybrid algorithms (described in Section 5.1.4) so that if each determines that a single LCT is the best classifier for the data, then the LCT's weights will be identical to those produced by MCS or the primitive learning algorithm for an LCT.

5.1.3 Comparison of MCS to its Primitive Components

Our first experiment is a baseline comparison of MCS to a univariate decision tree algorithm, a linear discriminant algorithm (which constructs a linear machine for multiclass tasks) that builds a classifier using all of the input features, a k-nearest neighbor algorithm ($k = 1$), and to a method that uses a ten-fold crossvalidation (CV) over the training data to select the best method. Our goal in this first comparison is to illustrate what the accuracy of a classifier produced by MCS would be *if* MCS had selected a single homogeneous representation. For each of the sixteen data sets, Table 5.1 shows the sample average and standard deviation of the classification accuracy for ten runs. In Table 5.2 we show the results of a paired t-test between MCS and each of the primitive learning methods.[3]

For three of the sixteen data sets (Heart Disease, Lymphography and Road Segmentation) MCS constructed a classifier that is statistically significantly (at the 0.05 level using a paired t-test) more accurate than each of its primitive components. For the Glass Recognition, Landsat, LED-24 Digit, Pixel, Vowel Recognition and Waveform data sets, the accuracy of the classifier produced by MCS is identical to the best of the other algorithms. For the remaining data sets, MCS produced classifiers that are not statistically different from the best homogeneous classifier for each data set.

The problem that no single model class will be best for all tasks is illustrated by the results for the individual model classes; for some data sets the difference between the accuracy of the best and the worst of the primitive algorithms is greater than 20. In contrast, the classifiers found by MCS for each data set were never significantly less accurate and were sometimes significantly more accurate than the

[3]For the Vowel data set, we used the original training and test sets and therefore ran each algorithm once.

Table 5.1 Accuracy of MCS and its primitive components

Data Set	k-NN	LCT	UTree	CV	MCS
Breast Cancer	96.2 ± 2.1	97.2 ± 2.5	95.7 ± 2.6	97.2 ± 2.5	96.2 ± 1.7
Congressional Votes	94.5 ± 3.4	96.2 ± 2.3	96.2 ± 2.6	95.7 ± 2.2	96.4 ± 2.3
Diabetes	71.6 ± 4.3	72.0 ± 5.4	72.5 ± 3.1	73.3 ± 3.3	73.7 ± 4.3
Glass Recognition	68.3 ± 8.3	56.1 ± 13.5	75.0 ± 9.5	68.9 ± 8.8	75.0 ± 9.5
Heart Disease	76.6 ± 8.1	79.7 ± 5.7	77.6 ± 4.9	79.0 ± 5.7	83.1 ± 4.4
Hepatitis	82.0 ± 8.9	80.0 ± 5.4	84.0 ± 7.2	81.3 ± 8.2	84.7 ± 10.0
Iris Plants	92.7 ± 6.6	92.0 ± 4.2	93.3 ± 7.0	92.7 ± 6.6	96.0 ± 4.7
Landsat	81.7 ± 2.5	69.5 ± 16.0	82.2 ± 3.3	80.3 ± 4.5	82.2 ± 4.4
LED-7 Digit	54.3 ± 9.8	72.3 ± 4.2	74.3 ± 4.0	74.2 ± 4.0	73.3 ± 4.4
LED-24 Digit	37.5 ± 12.8	57.5 ± 7.7	62.5 ± 12.1	63.1 ± 11.9	62.5 ± 13.8
Liver Disorder	61.5 ± 4.0	62.1 ± 5.8	66.5 ± 6.7	62.9 ± 7.9	67.4 ± 9.8
Lymphography	80.7 ± 5.9	73.6 ± 6.8	77.1 ± 8.1	80.7 ± 5.9	85.0 ± 7.1
Pixel	95.4 ± 2.1	89.7 ± 2.2	93.8 ± 1.8	95.4 ± 2.1	95.4 ± 2.1
Road Segmentation	78.9 ± 1.6	76.0 ± 7.0	81.3 ± 1.7	81.3 ± 1.7	83.4 ± 1.9
Vowel Recognition	50.2 ±	38.7 ±	40.5 ±	50.2 ±	50.2 ±
Waveform	66.4 ± 7.4	83.2 ± 6.1	68.6 ± 9.2	83.2 ± 6.1	83.2 ± 6.1

Table 5.2 Results of paired t-tests of MCS and primitives

Data Set	IBC	LCT	UTree	CV
Breast Cancer	1.000	0.163	0.390	0.163
Congressional Votes	0.410	0.996	0.336	0.373
Diabetes	0.173	0.115	0.207	0.713
Glass Recognition	0.005	0.011	1.000	0.018
Heart Disease	0.016	0.056	0.000	0.063
Hepatitis	0.382	0.163	0.815	0.323
Iris Plants	0.430	0.297	0.433	0.666
Landsat	0.656	0.042	1.000	0.129
LED-7 Digit	0.000	0.349	0.121	0.216
LED-24 Digit	0.000	0.285	1.000	0.590
Liver Disorder	0.133	0.102	0.842	0.214
Lymphography	0.054	0.006	0.033	0.054
Pixel	1.000	0.000	0.056	1.000
Road Segmentation	0.000	0.004	0.008	0.008
Waveform	0.000	1.000	0.003	1.000

Table 5.3 Model classes in the MCS classifiers

Data Set	Leaves	k-NN	LCT	UT	Hybrids	Prim. Alg
Breast Cancer	5.0	0.3	1.6	2.1	5	LCT
Congressional Votes	2.5	0.0	0.3	1.2	3	UTree
Diabetes	12.7	2.3	3.4	6.0	9	IBC
Glass Recognition	12.5	0.3	0.3	10.2	4	Utree
Heart Disease	2.9	0.9	0.5	0.5	3	IBC(6), LCT(1)
Hepatitis	2.5	1.2	0.2	0.1	4	IBC(5), LCT(1)
Iris Plants	3.5	0.5	1.0	0.0	3	IBC(1), LCT(6)
Landsat	16.6	3.5	1.8	4.4	9	IBC
LED-7 Digit	28.4	0.8	2.1	12.5	9	LCT
LED-24 Digit	14.9	0.3	0.1	11.8	4	Utree
Liver Disorder	8.2	1.5	2.8	2.9	9	LCT
Lymphography	4.4	0.9	0.4	0.0	4	IBC(5), LCT(1)
Pixel	7.0	1.0	0.0	0.0	0	IBC
Road Segmentation	54.1	2.4	4.7	25.7	10	
Vowel Recognition	11.0	1.0	0.0	0.0	0	IBC
Waveform	3.0	0.0	1.0	0.0	0	LCT

best of the primitive algorithms; indicating that a hybrid can provide a significant performance improvement for some data sets. This property of robustness is desirable in an automated algorithm selection system.

From these results we conclude that MCS's rules effectively choose a learning bias that produces an accurate classifier for each data set. The relationships that we have drawn, from data-set characteristics to bias allow MCS to find a classifier at least as accurate as the best of its primitive components and sometimes better. In contrast, the CV method is never statistically significantly more accurate and is sometimes *less* accurate than the best primitive algorithm for each data set.

Table 5.3 shows the average over the ten runs of the number of leaves and the number of test nodes, of each type of model class, in the MCS classifiers. The last two columns are not averages; the second to last column shows, out of the ten runs, the number of classifiers produced by MCS that were hybrids and the last column shows the primitive algorithm(s) that was chosen by MCS when it did not produce a hybrid. For example, MCS produced a hybrid in three of the ten runs for the Iris Plants data set. Of the remaining seven runs, MCS selected an LCT six times and an IBC once. For one data set, MCS produced a hybrid for each of the ten runs. For four data sets MCS produced a hybrid for nine of the ten runs. Except for the Road Segmentation data, this did not result in higher accuracy for MCS than the best of the primitive algorithms. For three cases, Pixel, Vowel Recognition and Waveform, MCS found a classifier that is identical to the best primitive algorithm. For the remainder of the data sets, the results are mixed.

In many cases MCS formed a hybrid classifier, but its accuracy was not significantly different from the best of the primitive algorithms. In previous work [Brodley, 1993] we reported results of an experiment that calculated the percentage overlap in the classification decisions made by MCS and the best primitive algorithm for each data set. Our conclusion in that experiment was: because all of the primitive algorithms are piecewise linear, for many data sets they will each form classifiers that define the same decision boundaries (Figure 4.3 in Section 4.2 illustrates one such case). In Section 5.2 we explore these representational similarities in greater detail.

5.1.4 Comparison of MCS to Other Hybrid Algorithms

In our second experiment we compared MCS to three hybrid algorithms, which have the same primitive components as MCS, but different search strategies. The three algorithms, RAcc, RInfo and RCV each search for the best classifier for each node by trying all possible algorithms. All three have the same general recursive procedure shown in Table 4.1. They differ from MCS in how the best algorithm is chosen for a data set. The classifiers compared at each node in the tree are a univariate test, a k-nearest neighbor (they search for the best value of k) and two linear combination tests, one constructed using SFS the other using SBE.

The three hybrid algorithms differ in the criterion used to choose among the candidates. RAcc uses the accuracy of the classifiers on the training data. RInfo uses the Information-Gain Ratio [Quinlan, 1986]. Finally, RCV uses the results of a four-fold crossvalidation to select the best learning algorithm, and then applies that algorithm to the set of data observed at the node to produce the classifier to place at that node in the hybrid tree. It is important to note that the comparison criteria used by RAcc, RInfo and RCV to choose a classifier are part of the feedback used by MCS to guide the rule-based search strategy.

The results of the experiment are shown in Table 5.4. In Table 5.5 we show the results of a paired t-test of MCS with each of the other hybrid methods. MCS is statistically significantly more accurate than RAcc for three data sets. MCS is significantly more accurate than RInfo for three data sets and more accurate than RCV for three data sets. None of the methods is significantly more accurate than MCS for any of these sixteen data sets.

A comparison of RAcc, RInfo and RCV to the primitive algorithms shows that each performs worse than the best of the primitive algorithms for some data sets. RAcc is statistically significantly worse than a univariate decision tree for the Congressional Votes, Hepatitis and LED-7 data sets, worse than an instance-based classifier for the Lymphography data, and worse than a linear combination test for the Waveform data set. RInfo is significantly worse than the best primitive algorithm for the Glass Recognition, Hepatitis, Lymphography and Waveform data sets. RCV is significantly worse than the best primitive algorithm for the Congressional Votes and Waveform data sets. In contrast, MCS never produced a classifier than was significantly less accurate than the best of the primitive algorithms.

Table 5.4 Accuracy of MCS and three other hybrid methods

Data Set	RAcc	RInfo	RCV	MCS
Breast Cancer	96.5 ± 2.3	96.7 ± 1.9	96.8 ± 3.0	96.2 ± 1.7
Congressional Votes	94.0 ± 3.2	95.5 ± 1.8	95.2 ± 2.3	96.4 ± 2.3
Diabetes	72.4 ± 4.5	74.5 ± 4.6	73.0 ± 3.4	73.7 ± 4.3
Glass Recognition	68.9 ± 6.0	62.2 ± 6.3	68.9 ± 6.0	75.0 ± 9.5
Heart Disease	82.4 ± 5.7	80.0 ± 6.0	82.4 ± 4.7	83.1 ± 4.4
Hepatitis	81.3 ± 8.8	81.3 ± 8.8	83.3 ± 1.1	84.7 ± 10.0
Iris Plants	96.0 ± 4.7	96.7 ± 4.7	94.7 ± 4.2	96.0 ± 4.7
Landsat Segmentation	83.7 ± 3.6	82.1 ± 2.9	82.6 ± 3.1	82.2 ± 4.4
LED-7 Digit Recognition	72.4 ± 3.9	73.8 ± 3.9	74.6 ± 4.4	73.3 ± 4.4
LED-24 Digit Recognition	60.0 ± 13.2	59.4 ± 12.9	60.6 ± 1.3	62.5 ± 13.8
Liver Disorder	62.6 ± 8.2	67.1 ± 8.7	62.4 ± 6.0	67.4 ± 9.8
Lymphography	74.3 ± 7.7	74.3 ± 7.7	77.9 ± 9.8	85.0 ± 7.1
Pixel	95.4 ± 2.1	95.4 ± 2.1	95.4 ± 2.1	95.4 ± 2.1
Road Segmentation	83.7 ± 1.4	83.2 ± 2.0	83.9 ± 1.7	83.4 ± 1.9
Vowel Recognition	50.2 ±	50.2 ±	50.2 ±	50.2 ±
Waveform	78.6 ± 8.6	78.6 ± 8.6	78.2 ± 7.2	83.2 ± 6.1

The results demonstrate that using feedback characteristics in a "knowledge-poor" way can lead to worse performance than the best of the primitive algorithms. These three algorithms attempt to choose a best algorithm with which to form each node in the hybrid classifier, but they do so using a single selection criterion that may not be appropriate for the given data set. For example, RAcc performs worse than a linear discriminant function for the Liver Disorder data set (the best primitive algorithm for this data set), whereas RInfo performs roughly the same as a linear discriminant function. This difference is due to the bias used to select the classifier for each node in the hybrid tree; the bias of the Information-Gain Ratio is preferable to the bias of accuracy for this data set. Indeed, the results show that performing automatic algorithm selection with an inappropriate bias can lead to worse performance than the best of the primitive algorithms.

In contrast, MCS is never worse than the best of the primitive algorithms. MCS's strategy for selecting an algorithm is itself a static bias; the rules do not change for a particular data set. However, MCS's strategy does not depend on only one source of feedback; the rules are designed to account for situations in which two or more measures yield conflicting results. For example, even if an LCT (based on i features) is more accurate than an IBC (k=1) for the training data, MCS continues to explore IBCs if more than half of the features remain in the LCT or the difference in accuracy is less than 10% (see Table 4.9).

From the results of this experiment we conclude that MCS is more robust than each of the other methods. In addition, MCS is, on average, less time consuming

Table 5.5 Results of paired t-tests of MCS and other hybrid methods

Data Set	RAcc	RInfo	RCV
Breast Cancer	0.619	0.428	0.458
Congressional Votes	0.021	0.579	0.144
Diabetes	0.457	0.579	0.772
Glass Recognition	0.072	0.005	0.072
Heart Disease	0.726	0.160	0.642
Hepatitis	0.230	0.230	0.327
Iris Plants	1.000	0.678	0.990
Landsat Segmentation	0.092	0.930	0.713
LED-7 Digit Recognition	0.294	0.519	0.090
LED-24 Digit Recognition	0.747	0.680	0.697
Liver Disorder	0.163	0.919	0.023
Lymphography	0.007	0.007	0.044
Pixel	1.000	1.000	1.000
Road Segmentation	0.579	0.778	0.285
Waveform	0.010	0.010	0.053

than each of the other hybrid classifier construction algorithms. In Table 5.6 we show the average across the ten runs of the number of seconds that each method used to form a hybrid classifier.[4] MCS requires less time than RAcc in twelve cases, RInfo in thirteen cases and RCV in all sixteen cases. For cases in which RAcc and RInfo took less time than MCS, the difference can be attributed to RAcc and RInfo selecting tests early on that preclude further search. For example, when an instance-based classifier is chosen for which k is equal to 1, no subtree will be grown below such a node. If for many nodes in the tree an IBC is selected over a linear combination or a univariate test, and if none of the three is 100% accurate for the training data, search terminates more quickly then if the linear combination or univariate test were chosen. For example, RAcc produced hybrid trees for the Diabetes data set that on average have 4.6 leaves (the trees produced by MCS have on average 12.7 leaves) and for which the majority of the test nodes are instance-based classifiers.

The results illustrate that MCS's rules not only produce more accurate classifiers than the other methods' more exhaustive approach to searching for a classifier at a node, but MCS requires less computation time, on average. The reduction in time is due to the fact that MCS does not necessarily need to try each algorithm at each node, unlike the other three methods.

[4]All algorithms were performed on a DEC 3000 running under OSF/1.

Table 5.6 Training time of hybrid algorithms (seconds)

Data Set	RAcc	RInfo	RCV	MCS
Breast Cancer	91.7	87.6	354.1	67.2
Congressional Votes	144.1	271.4	391.1	128.2
Diabetes	165.0	214.4	765.6	172.5
Glass Recognition	37.9	63.6	167.9	71.2
Heart Disease	133.1	150.7	702.5	80.7
Hepatitis	128.4	128.1	864.6	70.9
Iris Plants	3.0	3.2	11.3	2.6
Landsat Segmentation	255.7	195.7	777.6	181.3
LED-7 Digit Recognition	203.6	144.9	524.3	164.7
LED-24 Digit Recognition	718.6	716.3	2722.7	803.2
Liver Disorder	27.1	51.1	161.5	38.7
Lymphography	122.6	122.6	615.9	75.1
Pixel	1310.4	1809.3	4782.4	1076.3
Road Segmentation	1292.1	624.2	2863.6	528.0
Vowel Recognition	149.0	149.0	459.0	83.0
Waveform	1530.0	1530.2	2556.1	834.2

5.1.5 Benefits of Pruning Alternatives

In this experiment our goal is to examine the benefits of saving an alternative test at a node for consideration during pruning. A pruning alternative is saved if it is difficult to distinguish between two tests for the set of data observed at a node. During pruning, the alternative replaces the chosen test only if it increases the accuracy on the pruning data. In the first two columns of Table 5.7 we report the average accuracy of the classifier found by MCS for each data set with and without using pruning alternatives. In the last two columns we report the average time required when MCS was run with and and without pruning alternatives. From the results we see that pruning alternatives increased the classification accuracy in four cases (Heart Disease, Iris, Liver Disorder and Waveform). For the Waveform data set this difference was significant. Pruning alternatives decreased the accuracy (by more than 0.1) for only the Road Segmentation data set, but the difference was not statistically significant. The difference in time is negligible; only a few extra seconds were required to consider alternatives.

Although this aspect of MCS's search helped in only four cases, we believe that this is a promising direction for future research and we would like to explore this idea in conjunction with other pruning schemes. Indeed, one possible extension would be to store a test that is simpler than the chosen test. This would address situations in which the chosen test overfits the training data, but pruning it from the tree causes

Table 5.7 Contribution of pruning alternatives

Data Set	Accuracy		Training Time	
	PA	no PA	PA	no PA
Breast Cancer	96.2	96.5	67.2	65.9
Congressional Votes	96.4	96.2	128.2	76.8
Diabetes	73.7	73.9	172.5	171.1
Glass Recognition	75.0	75.6	71.2	68.4
Heart Disease	83.1	82.4	80.7	77.4
Hepatitis	84.7	84.7	70.9	72.4
Iris Plants	96.0	92.7	2.6	2.5
Landsat	82.2	82.1	181.3	177.6
LED-7 Digit	73.3	73.4	164.7	162.1
LED-24 Digit	62.5	62.5	803.2	798.5
Liver Disorder	67.4	63.8	38.7	40.0
Lymphography	85.0	83.6	83.2	82.2
Pixel	95.4	95.5	1076.3	1061.7
Road Segmentation	83.4	84.3	528.0	492.3
Waveform	83.2	77.5	834.2	835.8

more errors. In such cases, replacing the test with a simpler test might be the best choice.

5.1.6 Benefits of One-ply Deeper Search

In this experiment our goal is to examine the benefits of allowing one-ply deeper search. Recall that MCS searches one ply deeper if it cannot distinguish which of two tests appears more promising. In the first two columns of Table 5.8 we report the average accuracy of the classifiers found by MCS with and without using one-ply search. The last two columns report the average time MCS spent forming classifier when one-ply search was and was not used. Searching one ply deeper, when MCS can't distinguish between two candidate tests, increased accuracy in six cases (Congressional Votes, Glass Recognition, Landsat, LED-7 Digit, LED-24 Digit and Liver Disorder). However, for none of these cases is the difference statistically significant at the 0.05 level. It is important to note that although the increases in accuracy of one-ply deeper search over no extra search were not significant, these increases coupled with the increases due to using pruning alternatives are significant. Allowing MCS search one ply deeper never decreased the accuracy, but it did increase the amount of time required to form a classifier. From these results we conclude that more investigation of when to apply deeper search is necessary to bring down computation time; because in most cases allowing extra search does not increase accuracy, it could be applied less frequently.

Table 5.8 Contribution of depth search

Data Set	Accuracy		Training Time	
	SD	no SD	SD	no SD
Breast Cancer	96.2	96.2	67.2	38.0
Congressional Votes	96.4	94.5	78.8	96.2
Diabetes	73.7	73.6	172.5	125.7
Glass Recognition	75.0	68.9	71.2	38.3
Heart Disease	83.1	83.1	80.7	71.9
Hepatitis	84.7	84.7	70.9	71.1
Iris Plants	96.0	96.0	2.6	2.4
Landsat	82.2	82.1	181.3	152.1
LED-7 Digit	73.3	72.8	164.7	132.0
LED-24 Digit	62.5	58.1	803.2	342.9
Liver Disorder	67.4	65.9	38.7	29.2
Lymphography	85.0	85.0	83.2	82.3
Pixel	95.4	95.5	1076.3	1057.6
Road Segmentation	83.4	83.4	528.0	530.4
Vowel Recognition	50.2	50.2	149.0	149.0
Waveform	83.2	83.2	834.2	834.0

5.2 Representational Biases of MCS

In this section we leave aside the issue of whether a search algorithm can find a good classifier for a set of data and focus entirely on whether a model class can represent the true concept underlying a data set. We first explore the representational strengths and limitations of each of the individual model classes contained in MCS. Building on this analysis we then examine the representational capabilities and limitations of the tree-structured hybrid classifiers produced by MCS.

5.2.1 Individual Model Classes

Classifiers constructed from any one of MCS's three model classes each form a piecewise-linear partition of the given instance space. They differ in how (where) the boundaries may be placed in the space.

A univariate test of feature F_i, represents a decision boundary that is orthogonal to feature F_i's axis (and parallel to all the other feature axes). A univariate decision tree defines a set of orthogonal decision boundaries that partition the instance space into a set of hyper-rectangular regions each labeled with a class name. Univariate decision trees are biased toward concepts that can be expressed as Boolean combinations of the input features. The set of available branches at a test node in the decision tree represents a disjunction, whereas each path in the tree (from root to leaf) represents

a conjunction. When there are non-Boolean relationships among the features, a univariate decision tree can fail to capture the concept definition correctly.

A strength of univariate decision trees is that they need not evaluate all of the input features, which is desirable for representing concepts that are described by a subset of the input features. Indeed for many tasks, the set of relevant features may be unknown, and applying a univariate decision tree algorithm to such tasks can generate feedback as to which features are relevant to the task. For example, Cardie (1993) uses univariate decision trees to find a useful subset of the available features for a natural language processing task. The subset is then used in an instance-based classifier.

Univariate decision trees are the best representation for concepts defined by hyper-rectangular regions that are orthogonal to the feature axes. However, they can be used to approximate other types of boundaries. For example, a univariate decision tree can approximate a hyperplane boundary (see Figure 4.5). The quality of the approximation depends on the concentration of instances in the data set that are near the hyperplane boundary. The more dense the concentration the more accurate the univariate decision tree approximation will be. However, this will increase the size of the decision tree, and result in a description of the concept that is less precise than using a model class that is designed to represent hyperplanes, such as linear discriminant functions.

A set of R linear discriminant functions defines a set of R regions in the instance space, separated by hyperplanes, each labeled with a different class name. Linear discriminant functions are ideal for concepts for which the R classes are linearly separable. If a concept is not-linear then a linear discriminant function cannot represent it accurately. For example, a linear discriminant function would do poorly for each of the concepts shown in Figures 5.2 and 5.6.

An instance-based classifier defines a piecewise-linear partition of the instance space; the number of blocks is determined by the number and distribution of the instances, and the choice of k, which is the number of nearest neighbors to examine when classifying an unlabeled instance. The result is a set of regions, each labeled by a different class name, separated by piecewise-linear boundaries. Instance-based classifiers do best when all features are equally important because each feature contributes equally in the calculation of the distance between two instances. There are heuristic methods for weighting the contribution of each feature, but for some data sets, these methods decrease the accuracy of the classifier [Aha, 1990]. If a subset of the features describing the data are not relevant to the concept definition then an instance-based classifier that does not weight the features will have trouble representing such a concept.

Even given their different representational biases, for some concept definitions, the best hypothesis defined by each of the three model classes will be the same, given the goal of perfectly partitioning the instances into their classes. Figure 4.3 illustrates an example of such a case. For other data sets, all three will define different hypotheses. In Figure 5.1 we show a Venn Diagram of the concepts that can be represented by

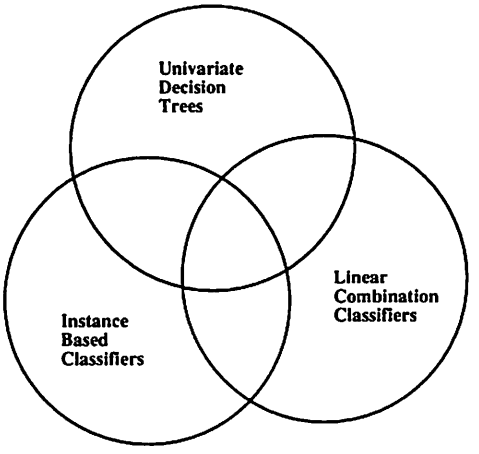


Figure 5.1 Hypothesis space of MCS's primitive representation languages.

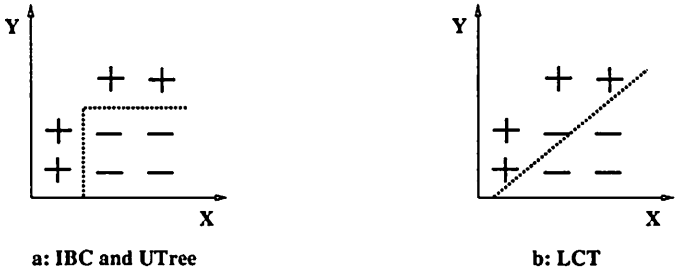
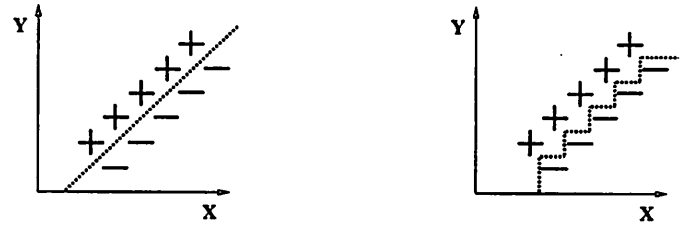


Figure 5.2 An example in the intersection of instance-based and univariate decision tree classifiers.

the primitive model classes contained in MCS. The diagram depicts the fact that the three model classes intersect in hypothesis space. To illustrate that the intersection of any two model classes is non-empty we present an example from each. (In Figure 4.3 we saw an example of the intersection of all three model classes.)

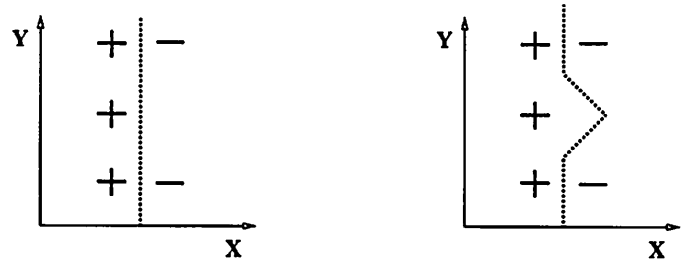
Figure 5.2 shows an example of a data set for which an instance-based (IBC) and a univariate decision tree (Utree) classifier would define the same hypothesis, but a linear discriminant classifier (LCT) would define a different hypothesis. Given that the true concept definition is the one found by the instance-based classifier and the univariate decision tree, this is an example from the intersection IBC $\cap$ Utree.

Figure 5.3 shows an example of a data set for which an instance-based and a linear discriminant classifier would define the same hypothesis, but a univariate decision tree would define a different hypothesis. Given that the true concept definition is the one found by the instance-based classifier and the linear discriminant classifier, this is an example from the intersection IBC $\cap$ LCT.



a: IBC and LCT b: UTree

Figure 5.3 An example in the intersection of instance-based and linear combination classifiers.



a: LCT and UTree b: IBC

Figure 5.4 An example in the intersection of linear combination and univariate decision tree classifiers.

Figure 5.4 shows an example of a data set for which a linear discriminant and a univariate decision tree classifier would define the same hypothesis, but an instance-based classifier would define a different hypothesis. Given that the true concept definition is the one found by the linear discriminant classifier and the univariate decision tree classifier, this is an example from the intersection LCT $\cap$ UTree. Figure 5.4 illustrates a situation in which an IBC is a poor choice because not all of the input features are relevant to the concept definition. In this example, only the value of X is relevant to the concept definition. If the feature Y were removed from the IBC, then it too could represent the correct hypothesis. Indeed, there are algorithms designed especially to rid IBCs of the problem of irrelevant features [Aha, 1990; Cardie, 1993].

One question of interest is how large is each space that represents the overlap among two or more of the three model classes? Although we know of no way to answer this question analytically, we can provide some empirical analysis. In the above discussion we have held the search algorithm separate from a model class's representational ability. In the following experiment we cannot enforce this separation.

To examine the overlap in the model classes we would like to examine the differences among the decision boundaries each classifier forms. To this end, we

Table 5.9 Classification overlap (percentage)

Data Set	IBC-LCT	IBC-Utree	LCT-Utree	All Three
Breast Cancer	**97.0**	95.9	96.1	94.5
Congressional Votes	94.0	93.1	**96.7**	91.9
Diabetes	74.1	73.6	**80.0**	63.8
Glass Recognition	55.0	**64.4**	49.4	37.2
Heart Disease	78.3	76.9	**82.1**	68.6
Hepatitis	**91.3**	83.3	82.7	78.7
Iris Plants	**95.3**	92.7	94.7	91.3
Landsat Segmentation	70.3	**81.8**	74.9	64.7
LED-7 Digit Recognition	61.8	61.7	**85.0**	56.0
LED-24 Digit Recognition	41.3	36.9	**58.1**	23.1
Liver Disorder	56.5	**63.8**	56.8	38.5
Lymphography	76.4	**81.4**	78.6	68.6
Pixel	88.5	**91.8**	87.3	84.4
Road Segmentation	75.0	**81.2**	78.7	69.3
Waveform	66.8	60.4	**68.6**	48.6

compute a heuristic measure of the similarity between two classifiers' boundaries: we compute the percentage overlap in the classification decisions that each makes. Although this does not tell us the exact difference in the decision boundaries, it gives a rough measure of how different these boundaries are for classification of the types of instances observed in the domain of interest.

After each of MCS's primitives learning algorithms was run using the same random split of a data set into training and test sets[5], we computed the overlap between the classification decisions made by each pair of learning algorithms on the test set; we counted the number of the instances in the test set that were classified the same way by each of two classifiers constructed using different algorithms. In Table 5.9 we report the average for each data set, over ten runs, of the classification overlap percentage between each pair of classifiers and among all three classifiers. The higher the percentage, the closer the decision boundaries are for classification of instances likely to be observed in the domain. For example, the largest pair-wise overlap for the Diabetes data set is between LCT and Utree. This indicates that the LCT defines decision boundaries that are more similar to Utree's boundaries than IBC's. We show the largest pair-wise overlap for each data set in boldface.

The first observation to make from the results of this experiment is that no pair is more similar than any other pair *across the entire set of tasks.* Obviously, for an

[5]The experimental method is described in Section 5.1.2.

individual task, one pair will have a higher overlap than the others, but there is no trend.

Combining the results reported in Tables 5.1 and 5.9, we observe that in twelve out of fifteen cases the highest overlap is between the two most accurate model classes for the data set. This makes sense because classifiers from these two model classes are closer to defining the true concept definition and therefore their boundaries are more similar. In six cases the largest overlap for a data set is higher than the best accuracy for that data set. When the overlap is higher than the best accuracy this means that the two classifiers are misclassifying some of the same instances the same way. In five cases the overlap is lower than the best accuracy and in the remaining four cases they are about the same. When the overlap is lower than or equal to the accuracy it means that they are either not classifying the same instances correctly or when they misclassify an instance they do so as different classes. Note that in the case of data sets that have only two classes, an overlap that is lower than the minimum accuracy of the two methods indicates that they are not classifying the same subset of the instances correctly.

In conclusion we should restate that in the overlap experiment we cannot separate the affects of the model class and the search bias for that model class. Indeed we find it highly likely that by using a different search bias these results would change. What this experiment does provide is some insight into trying to understand the differences in these model classes.

5.2.2 Hybrid Combinations

In this section we explore the representational strengths and weaknesses of the model class of tree-structured hybrids that combines univariate decision trees, linear combination tests and instance-based classifiers. Combining all three individual model classes overcomes many of the representation problems of each of the individual classes. For example, the problem that a linear discriminant function can only represent linearly separable concepts is eliminated because the hybrid classifier can represent a series of hyperplanes; the children of a linear machine test, in the tree, can correct the classification errors of the parent. None of the homogeneous model classes can represent a curved boundary and this remains true of the hybrid model class as well, but the hybrid model class can approximate a curved boundary.

In Figure 5.5 we show a Venn Diagram of the hypothesis space of all piecewise-linear classifiers. Clearly the hypothesis space of MCS is at least a large as the union of the three primitive model classes; by selecting among the three homogeneous model classes MCS can choose a hypothesis in the union of the three model classes. The question is how much larger is the space when hybrids are permitted? A linear combination test can represent any arbitrary hyperplane in the space. Permitting a tree of linear combination tests will allow the representation of any piecewise-linear concept definition. Therefore MCS's hypothesis space is equal to the entire class of piecewise-linear concept boundaries. Note that we are in no way stating that all

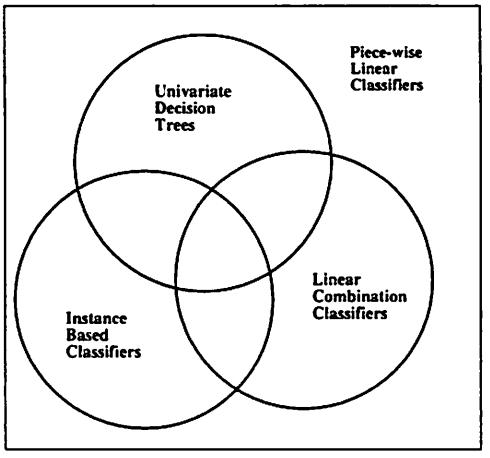


Figure 5.5 Hypothesis space of MCS's hybrid representation.

of these can be found by the search strategy of MCS, but merely that the hybrid representation can represent them.

Our reason for including all three model classes, is that just because we can represent all piecewise classifiers with a tree of linear discriminant functions does not mean that the existing search algorithms for linear discriminant functions can find the best piecewise-linear classifier. We include univariate decision trees and instance-based classifiers because they have different search biases (preference biases) that may lead to a more accurate concept representation than the biases of using linear discriminant function algorithms only. Indeed in the overlap experiment we saw strong evidence that the different search biases find different hypotheses for many of the data sets.

Although a concept definition that defines a curved boundary in the instance space cannot be represented precisely by a piecewise-linear classifier it can be approximated. For a finite population (eg. all two-class concepts described by five Boolean features) it is not possible for a concept to be defined by curved boundaries. For infinite populations (any task for which one or more of the features is continuous) there can be concept descriptions that are not piecewise linear. However, to produce a classifier in a finite amount of time, a finite set of data is used to form the classifier, and therefore the curved boundary can be approximated by a finite number of linear segments. (If there is an infinite amount of data, then there is no piecewise-linear approximation of a curved boundary that contains a finite number of pieces.) As the number of instances observed near the boundary is increased so too will the number of segments in the piecewise-linear approximation.

In Figure 5.6 we depict a data set from a concept description that has a curved boundary in the instance space. The figure illustrates how the number of pieces in a piecewise-linear approximation increases as the concentration of instances around the concept boundary increases. Although a piecewise-linear representation can

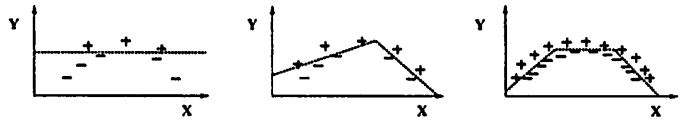


Figure 5.6 Piecewise approximations of a curved boundary from the given data

approximate a curved concept description boundary it does not provide as precise a definition as using a representation that naturally represents curves, such as Taylor polynomials.

5.3 Summary

In this chapter we evaluated the MCS's performance and representational ability. The results of an empirical comparison of MCS to its primitive components illustrate that MCS produces classifiers that are sometimes more accurate and never less accurate than each of the three homogeneous algorithms. This result indicates that the heuristic rule-based search strategy implemented in MCS solves the selective superiority problem for these algorithms. In some cases, MCS produced classifiers that were statistically significantly more accurate than the best primitive algorithm, illustrating that for some data sets a hybrid bias is best.

The results of an empirical comparison of MCS to three other hybrid classifier construction algorithms illustrate that merely increasing the search space by permitting hybrids will not increase accuracy. Each of the three methods is statistically significantly worse than MCS for several of the data sets. In contrast, MCS produced classifiers that were always equal to or better than each of the other hybrid methods.

The other three hybrid methods exhaustively try each of the primitive algorithms at a node to generate a set of candidates. Each method chooses a test from this set using a different criterion. In contrast, MCS's rules use the criteria of these methods, but rather than selecting a classifier for each data set based on a single source of feedback, MCS's rules use more than one source of feedback to make a selection. Moreover, when the various measures yield conflicting results, MCS applies more search before making a decision. For example, when choosing between a univariate test and a linear combination test, and the measures of accuracy and the Information-Gain Ratio give conflicting results, MCS applies one-ply deeper search rather than rely on only one of these measures. The results demonstrate that MCS's knowledge-based approach is both more accurate and less time-consuming than the alternative three methods.

We evaluated the contribution to MCS's performance of pruning alternatives and search depth. Our results indicated that although both increased performance in some cases, more work in both of these areas needs to be done. Specifically the rules

for applying deeper search need to be more stringent to cut back on computation time, and other criteria for retaining alternative tests needs to be explored.

In our evaluation of the representational biases of MCS, we first examined the strengths and limitations of the homogeneous model classes. We illustrated that there is some overlap among the three model classes in hypothesis space. We then extended our evaluation to include the hybrid tree-structured classifiers produced by MCS. We concluded that the hybrids can represent any piecewise-linear concept definition, but can only approximate concept definitions that split the instance space with curved boundaries.

CHAPTER 6

CONCLUSIONS

The objective of this research was to study the problem of how to select the best learning bias for a data set. The research was motivated by the selective superiority results of empirical comparisons among learning algorithms, which illustrate that no single bias exists that is best for all learning tasks [Brodley, 1993]. The problem of selecting an appropriate learning bias is complicated further because for some learning tasks, different subtasks are learned best using different biases. This chapter summarizes the results obtained in this dissertation and their implications toward solving the selective superiority problem in machine learning. We conclude with a discussion of the issues for future investigation uncovered by this research.

6.1 Summary of the Dissertation

This dissertation presented a new approach to automatic selection of an appropriate bias for a given data set. The approach uses a set of heuristic rules to perform a hill-climbing search for the best hypothesis space (model class) and search bias for a given data set. The rules measure characteristics of the data set by using feedback from the learning process. In addition, the approach has the ability to select different biases for different subspaces of the learning task, thereby forming a hybrid classifier of the data. In the dissertation we tested the following two hypotheses about the approach:

1. *Domain independent knowledge about characteristics in the form of feedback from learning can effectively guide an automatic algorithm selection search.*

2. *Our knowledge-based search strategy for finding a hybrid classifier will produce a classifier that is never worse than, and for some data sets is better than, any homogeneous classifier produced by its primitive components.*

In this section we summarize the research conducted to validate these two claims. We begin by describing the design and implementation of the Model Class Selection System (MCS), a recursive automatic algorithm selection system. We then discuss the

results obtained from applying MCS to a variety of learning tasks. We conclude this section with a discussion of the implications of these results for solving the selective superiority problem.

6.1.1 MCS: The Design and Implementation

Our approach to automatic algorithm selection applies knowledge about the biases of machine learning algorithms to guide the search for the best algorithm to apply to a given set of data. The approach computes characteristics of a data set using feedback from a search through the space of available representation and search biases. The approach iteratively fits a classifier to the data using the representation language and search bias currently considered best for the data set. Next it computes measures of how well the resulting classifier fits the data. The measures are used by a heuristic rule-based search strategy to decide whether a best classifier has been found or whether to search further, and if so which bias to try next.

Our approach has the ability to select different learning biases for different subspaces of the learning task. We included this ability because choosing among a set of learning algorithms that each employ a homogeneous representation bias assumes that a classifier for the data set is represented best using a single representation language. For some data sets this is not the case; different subspaces of the learning task may be best learned employing different representation biases, indicating that forming a hybrid classifier will result in higher classification accuracy than a homogeneous classifier.

We implemented our recursive automatic algorithm selection approach producing the Model Class Selection system (MCS). Given a set of data, MCS builds a classifier using a set of heuristic rules to guide a search for the best representation language, from which to form a test, for each node in a hybrid classifier. MCS combines three primitive representation languages that have been used extensively in both machine learning and statistics algorithms: linear discriminant functions, decision trees and instance-based classifiers. Our choice of these representation languages stemmed from the results of empirical comparisons, which illustrate that these three languages produce classifiers of differing accuracy for different data sets; each of the three was selectively superior for some data sets. Classifiers constructed from any one of these three languages each form a piecewise-linear partition of the given instance space. They differ in how (where) the decision boundaries may be placed in the space.

MCS searches for a hybrid classifier from the available model classes using a hill-climbing search to select a test for each node in the tree-structured classifier. A set of heuristic rules is used to decide which model classes to try, which model classes should be avoided, and to determine when a best test at each node has been found. The instance space is partitioned according to the chosen test, and the search is applied recursively to each resulting subset that contains instances from two or more classes. After MCS has constructed a hybrid classifier that perfectly partitions the training instances into regions each labeled with a single class name, it applies a

pruning algorithm to reduce the estimated error of the classifier as computed for an independent set of instances.

6.1.2 Results Obtained by Using MCS to Form Classifiers

Our experimental results demonstrated that MCS performed as well as or better than each of its primitive learning algorithms across a variety of learning tasks, illustrating that MCS is a robust method for choosing among the biases of decision trees, linear discriminant functions and instance-based classifiers. In MCS we have solved the selective superiority problem for these algorithms by providing a robust automatic algorithm selection system.

The classifiers found by MCS for each data set were never significantly less accurate and were sometimes significantly more accurate than the *best* of the primitive algorithms, indicating that a hybrid can provide a significant performance improvement for some data sets. In many cases MCS formed a hybrid classifier, but its accuracy was not significantly different from the best of the primitive algorithms. We attribute this phenomenon to the fact that all of MCS's primitive model classes and the hybrids produced by MCS are from the set of piecewise-linear concept descriptions and therefore, will define similar hypotheses for many data sets. However, both the empirical results and the analysis of representational bias indicate that using MCS, strictly increases the number of data sets for which a good generalization can be found over its primitive components.

The results of an empirical comparison of MCS to three hybrid algorithms that each apply all candidate algorithms to construct each node of a hybrid tree illustrated that MCS's knowledge-based search strategy is less time consuming and produces more accurate classifiers. Each of the three other hybrid algorithms, RAcc, RInfo and RCV, performed worse than MCS for several data sets, and for some data sets, worse than the best of the primitive learning algorithms. In contrast, MCS never produced a classifier than was less accurate than the best of the primitive algorithms, illustrating that using the feedback characteristics in a "knowledge-poor" way can lead to worse performance than the best of the primitive algorithms. Unlike the other three hybrid methods, MCS's rule-based search strategy does not get misled by the larger search space that permitting hybrids creates. We conclude that MCS is more robust and less time consuming than each of the other methods.

6.1.3 Implications of the Results

Our results support the claim that domain-independent knowledge about characteristics in the form of feedback from learning can effectively guide an automatic algorithm selection search. First, recall that twelve of the data sets used in the empirical evaluation were not used to develop the rule base. Second, MCS was never less accurate than each of its primitive components. Taken together this provides

strong support for the claim that domain independent knowledge about the biases of machine learning algorithms exists and is useful for automatic algorithm selection.

In addition, our results support the claim that our knowledge-based search strategy for finding a hybrid classifier produces classifiers that are sometimes better, but never worse than a homogeneous classifier. Although we have no analytical proof of this claim, our results across sixteen data sets in our empirical evaluation of MCS provide strong support for such a claim. MCS created classifiers that were significantly more accurate than each of the homogeneous classifiers for three of the sixteen data sets. For the remaining thirteen, the accuracy of the hybrid classifiers produced by MCS was equal to the *best* of the homogeneous learning algorithms. Indeed our comparison to "knowledge-poor" hybrid classifier construction algorithms indicated that a knowledge-based strategy both increases accuracy and decreases computation time over algorithms that exhaustively try all candidate learning algorithm in the construction of each node in a tree-structured hybrid classifier.

6.2 Contributions

For the past few years researchers have become increasingly aware that selecting an appropriate learning bias plays a pivotal role in the success of forming an accurate classifier for a set of data. However, prior to this research no one had codified this process as a set of heuristic rules that use feedback from learning to guide an automatic algorithm selection search. This research illustrates that general domain independent knowledge about the biases of machine learning algorithms can be used to guide the search for the best learning bias.

The results obtained in the dissertation demonstrate that there are real-world data sets for which different subspaces require different learning biases. Our results comparing MCS to three "knowledge-poor" approaches illustrate that care must be taken in forming hybrid classifiers; an ineffective search algorithm for forming a hybrid classifier can lead to worse performance than selecting the best of a set of homogeneous classifiers. In contrast MCS is a robust recursive automated algorithm selection system.

The automation of classification learning is an ongoing concern in the machine learning, pattern recognition and statistics communities. As the amount of data and knowledge about the world continues to increase, so does the need for data analysis techniques that compress and help to make sense of the data. Inductive generalization provides a technique for both data compression and understanding. The creation of systems that search multiple representation spaces increases the autonomy of learning machines. Ultimately, a human will no longer be required to select a learning algorithm, making machine learning techniques accessible to a wider range of scientists. They will no longer need to understand the representational biases

of the available learning algorithms, because this step in the analysis of data will now be automated.

6.3 Issues for Future Research

This research has uncovered several directions for future investigation in automatic bias selection for classifier formation. The evaluation of the representational strengths and limitations of MCS illustrated that MCS can only approximate non piecewise-linear concept definitions. One direction in which to extend MCS is to include a non-linear model class, such as neural nets or higher-order polynomial functions. A question of interest is whether the performance of MCS would increase substantially with such an addition.

Indeed, one problem with evaluating MCS's performance is that it is unclear for many data sets whether better performance can be achieved. Unless one can exhaustively enumerate all possible concept definitions for a data set and know which is the best, it is impossible to know whether MCS has achieved the maximum obtainable accuracy for these data sets. It may be the case that because of noise in the data set, squeezing out more accuracy is impossible. For example, for the LED-7 data set, we know that univariate decision trees and MCS each form a classifier whose accuracy is near the maximum obtainable accuracy. If we can find data sets for which we know the maximum obtainable accuracy, then we will be able to evaluate more precisely the performance of automatic algorithm selection systems.

MCS contains several different measures of how well a classifier fits the data, but there are many more. Indeed, at the beginning of the search at a node, MCS determines where to begin the search based on a simple test relating the number of features to the number of instances. A direction in which to extend MCS is to incorporate the results of efforts such as the STATLOG project (Feng, et al. 1993) in order to reduce the amount of search required. One of the results of the STATLOG project is a study that relates statistical measures of a data set to the performance of different algorithms. Measures such as homogeneity of covariances and skewness were used to explain differences in the performances of a set of machine learning, statistical and neural net algorithms on a test-suite of classification tasks. The knowledge resulting from this type of study could be used to begin a heuristic search. By starting the search from a promising part of the space, it stands to reason that the time required to find a best classifier would be greatly reduced.

In our comparison to the other hybrid methods, we observed that for some data sets it was a better to use the Information-Gain Ratio and for others it was better to use accuracy to select a test at a node. This is an example of how using the same model class (in this case tree-structured hybrids), but using different search biases can lead to significantly different performances. In our implementation of MCS, we chose a single search bias for each homogeneous model class (except for the selection of terms

for inclusion in an LCT). An extension would be the inclusion of other search biases for each of the homogeneous model classes. For example, univariate decision trees in MCS were built using the Information-Gain Ratio. Perhaps, for some data sets it is better to use accuracy, and we would like a way to determine this automatically. For instance-based classifiers, MCS selects a value for k using accuracy.[1] For some data sets it may be better to use the Information-Gain Ratio or some other criterion. We conjecture that the performance of MCS could be increased with the addition of other search biases for the homogeneous model classes.

Searching one ply deeper, for cases in which the measures could not distinguish between two classifiers, was found to contribute little to MCS's performance, but at a large cost in terms of the time required to form a classifier. Perhaps a better use of this time would be to employ a more restrictive test for when to apply further search and then to search deeper at those points. One criterion we used to determine when to search deeper examines whether the measures of two tests are within 10% of one another. A closer examination on a per-run basis could indicate if this is too loose a criterion. Another experiment of interest is to study the effects of a deeper search on MCS's performance. Indeed, it would be useful to to characterize those situations in which a deeper search depth helps in order to apply one-ply search more judiciously.

Saving alternative tests during search for consideration during pruning helped increase MCS's performance, at an insignificant time cost. We feel that this is a promising direction for future research for any tree-structured classifier – hybrid or not. One possible extension is to retain alternative tests only if they are simpler than the chosen test. This would alleviate the problem that during pruning a complex test at a node overfits the training data, but removal increases the error estimate more than retaining the complex test. Another future direction is to investigate the utility of selecting an alternative at a node using a different criterion than that used to select the actual test. This would be beneficial for data sets for which the bias of the best criterion to use changes as the algorithm proceeds deeper in the tree. In most cases the assumptions that can be made about the distribution of the instances is no longer true when constructing a test node for a subset of the instances (unless the subset is drawn randomly).

In conclusion, this research has uncovered many promising areas of future research in automatic learning bias selection. The results have indicated that a recursive knowledge-based automatic algorithm selection system can significantly increase the predictive accuracy of classifiers formed for sets of data over its primitive learning components.

[1] Recall that k is the number of nearest neighbors considered when classifying an instance.

APPENDIX A

CODING TESTS AND ERROR VECTORS

To determine the number of bits required to code a consistent hypothesis we need to code both the hypothesis induced by the learning algorithm and the errors that the hypothesis makes. To code the error-list we use Cover's (1973) enumerative encoding scheme. To code a linear machine or a univariate symbolic test we assume a fixed ordering for the features and that the receiver of the code knows this order. We code both the test and the error vectors of each partition resulting from the test. For the following discussion let k equal the number of classes, let n equal the number of attributes and let $V(a_i)$ equal the number of distinct values for attribute a_i.

A.1 Coding a Univariate Test

We need $log_2(n)$ bits to specify which attribute is being tested. If the test is discrete, then for each branch we encode the value of the branch, which takes $V(a_i)log_2(V(a_i))$ bits. If the attribute is continuous, then we need to code the value of the range and we use the method specified below for coding real numbers. In addition we need need $V(a_i)log_2(k)$ bits to code the class label for each branch of the tree.

A.2 Coding a Linear Machine

We need k bits to specify which classes have a linear discriminant in the linear machine. We code each linear discriminant as a vector of numbers using the method described below for coding real numbers. To represent variables that have been eliminated from the linear machine we assign their weights the value zero.

A.3 Coding Real Numbers

We first try to reduce the precision necessary to retain the same accuracy. For example given the number 23.67000 it is only necessary to code 23.67. There are two

ways we can encode a real-valued number. The first method multiplies the number by 10^x to achieve a precision to the x^{th} decimal place and then codes the result as an integer. The second method encodes the integer part of the number separately from the fractional part. The fractional part is then treated as an integer. We choose the second method as in our experiments the code for 23 plus the code for 67 was generally less than the code for 2367. We use Elias' (1975) asymptotically optimal prefix code for integers. An additional bit is needed to code the sign.

APPENDIX B

COMPUTING THE INFORMATION-GAIN RATIO OF A TEST

In this appendix we explain how the gain-ratio information theoretic measure is calculated for univariate, linear combination or instance-based classifier tests. The gain-ratio information theoretic measure for a test, T_i, in a decision tree measures the gain in information if the test is used to partition the instances. We use same formula as the decision tree algorithm ID3 uses, and for a detailed explanation of the gain-ratio metric and its underlying assumptions see Quinlan (1986).

B.1 Univariate Tests

For the two class case, let p and n be the number of instances labeled positive and negative in the training set, respectively. Let v be the number of distinct values observed for attribute A, and let p_i and n_i be the number of positive and negative examples for which attribute A takes on value i. Then the gain-ratio for attribute A is given by the formula: $gain(A)/IV(A)$, where

- $gain(A) = I(p,n) - E(A)$, measures the gain in information if attribute A is used as a test at the node.
- $I(p,n) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$, is the information required for classification.
- $E(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(p_i,n_i)$, is the expected information required for a tree with root A.
- $IV(A) = -\sum_{i=1}^{v} \frac{p_i+n_i}{p+n} log_2 \frac{p_i+n_i}{p+n}$, measures the information of the values of A.

For the multiple class case: let k be the number of distinct classes observed in the training instances; let C equal the total number of observed instances; let v be the number of values observed for attribute A; let $c_j, j = 1,..,k$ be the number of observed instances labeled class j; and let $c_{j,i}, j = 1,...,k, i = 1,..,v$ be the number of

observed instances labeled class j with value i for attribute A. Then for the multiple class case, the gain-ratio for attribute A is given by the formula: $gain(A)/IV(A)$, where

- $gain(A) = I(c_1, ..., c_k) - E(A)$
- $I(c_1, .., c_k) = -\sum_{j=1}^{k} \frac{c_j}{C} log_2 \frac{c_j}{C}$
- $E(A) = \sum_{i=1}^{v} \frac{\sum_{j=1}^{k} c_{j,i}}{C} I(c_{1,i}, ..., c_{k,i})$
- $IV(A) = -\sum_{i=1}^{v} \frac{\sum_{j=1}^{k} c_{j,i}}{C} log_2 \frac{\sum_{j=1}^{k} c_{j,i}}{C}$

B.2 Linear Combination and Instance-Based Classifier Tests

To calculate the information gain for a linear machine or an instance based classifier we use the formula for the multiple class case. The number of values (branches) for a test is equal to the number of observed classes, k. The gain-ratio for test T, is given by the formula: $gain(T)/IV(T)$, where

- $gain(T) = I(c_1, ..., c_k) - E(T)$
- $I(c_1, .., c_k) = -\sum_{j=1}^{k} \frac{c_j}{C} log_2 \frac{c_j}{C}$
- $E(T) = \sum_{i=1}^{k} \frac{\sum_{j=1}^{k} c_{j,i}}{C} I(c_{1,i}, ..., c_{k,i})$
- $IV(T) = -\sum_{i=1}^{k} \frac{\sum_{j=1}^{k} c_{j,i}}{C} log_2 \frac{\sum_{j=1}^{k} c_{j,i}}{C}$

APPENDIX C

A SAMPLE OF THE CLASSIFIERS FOUND BY MCS

In this appendix we show a sample MCS tree for each of the sixteen data sets used to evaluate the MCS system. The purpose is to show the hybrid structure of the trees found by MCS. In our experimental results we report the average of ten different partitions of each data set into training and test sets. The tree depicted here for each data set is the tree found for the first partition.

C.1 Breast Cancer

$LCT_{i=8}$
  LEAF: benign
  U
    U
    | U
    | | $IBC_{k=3}$
    | | | U
    | | | | LEAF: malignant
    | | | | LEAF: benign
    | | | LEAF: malignant
    | | LEAF: malignant
    | LEAF: malignant
    LEAF: malignant

C.2 Congressional Votes

U
  LEAF: democrat
  LEAF: republican

C.3 Diabetes

```
LCT_{i=6}
  LCT_{i=5}
  | U
  | | LCT_{i=7}
  | | | LEAF: 1
  | | | LCT_{i=8}
  | | |   LEAF: 0
  | | |   LEAF: 1
  | | LCT_{i=5}
  | |   LEAF: 1
  | |   LEAF: 0
  | U
  |   LCT_{i=3}
  |   | LCT_{i=7}
  |   | | LEAF: 1
  |   | | LEAF: 0
  |   | IBC_{k=3}
  |   |   LEAF: 1
  |   |   LCT_{i=7}
  |   |     LEAF: 1
  |   |     LEAF: 0
  |   LCT_{i=7}
  |     LCT_{i=8}
  |     | LEAF: 1
  |     | LEAF: 0
  |     LCT_{i=8}
  |       LEAF: 1
  |       U
  |         LEAF: 0
  |         IBC_{k=5}
  |           U
  |           | LEAF: 1
  |           | LEAF: 0
  |           LEAF: 0
  U
    LEAF: 0
    IBC_{k=3}
      LEAF: 1
      U
        LEAF: 0
        U
          LCT_{i=7}
          | LEAF: 1
```

| LEAF: 0
LEAF: 0

C.4 Glass Recognition

$IBC_{k=3}$
LEAF: 1
LEAF: 2
LEAF: 3
LEAF: 5
LEAF: 6
LEAF: 7

C.5 Heart Disease

$IBC_{k=3}$
LEAF: absence
LEAF: presence

C.6 Hepatitis

$IBC_{k=5}$
LEAF: 2
LEAF: 1

C.7 Iris Plants

$LCT_{i=1}$
LEAF: Iris-setosa
$IBC_{k=3}$
| $LCT_{i=2}$
| | LEAF: Iris-versicolor
| | LEAF: Iris-viginica
| LEAF: Iris-viginica
LEAF: Iris-viginica

C.8 Landsat

```
LCT_{i=6}
  U
  | LEAF: 2
  | U
  |   LEAF: 2
  |   U
  |     LEAF: 2
  |     LCT_{i=6}
  |       LEAF: 2
  |       U
  |       | LEAF: 2
  |       | LEAF: 4
  |       LEAF: 1
  IBC_{k=3}
  | LEAF: 2
  | IBC_{k=4}
  | | LEAF: 4
  | | U
  | | | IBC_{k=1}
  | | | | LEAF: 2
  | | | | LEAF: 4
  | | | LEAF: 4
  | | LEAF: 1
  | LCT_{i=3}
  |   LEAF: 4
  |   LEAF: 1
  LEAF: 1
  LEAF: 0
  LCT_{i=2}
    LEAF: 3
    LEAF: 1
    LEAF: 2
```

C.9 LED-7 Digit

$LCT_{i=7}$
  LEAF: 0
  U
  | U
  | | LEAF: 3
  | | LEAF: 0
  | LEAF: 8
  $IBC_{k=4}$
  | LEAF: 0
  | LEAF: 6
  | LEAF: 2
  $IBC_{k=1}$
  | LEAF: 1
  | LEAF: 2
  | LEAF: 3
  | LEAF: 5
  LEAF: 9
  $IBC_{k=3}$
  | LEAF: 1
  | LEAF: 4
  | LEAF: 3
  U
  | LEAF: 5
  | $IBC_{k=6}$
  | LEAF: 6
  | LEAF: 5
  | LEAF: 2
  | LEAF: 2
  LEAF: 3
  $IBC_{k=4}$
  | LEAF: 8
  | LEAF: 2
  | LEAF: 7
  | LEAF: 6
  | LEAF: 3
  | LEAF: 9
  | LEAF: 0
  $IBC_{k=3}$
  LEAF: 3
  LEAF: 1

LEAF: 7
LEAF: 1

C.10 LED-24 Digit

$IBC_{k=7}$
LEAF: 5
LEAF: 1
LEAF: 3
LEAF: 6
LEAF: 9
LEAF: 2
LEAF: 7
LEAF: 0
LEAF: 4
LEAF: 8

C.11 Liver Disorder

U
$LCT_{i=4}$
| LEAF: 1
| $LCT_{i=5}$
| U
| | LEAF: 2
| | LEAF: 1
| $LCT_{i=3}$
| LEAF: 2
| LEAF: 1
$LCT_{i=5}$
U
| U
| | LEAF: 2
| | LEAF: 1
| $LCT_{i=4}$
| LEAF: 1
| LEAF: 2
LEAF: 2

C.12 Lymphography

$LCT_{i=12}$
LEAF: 2
LEAF: 1
LEAF: 3
LEAF: 0

C.13 Pixel

$IBC_{k=1}$
LEAF: BRICKFACE
LEAF: SKY
LEAF: FOLIAGE
LEAF: CEMENT
LEAF: WINDOW
LEAF: PATH
LEAF: GRASS

C.14 Road Segmentation

$IBC_{k=4}$
LEAF: ROAD
$IBC_{k=4}$
| LEAF: ROADLINE
| U
| | U
| | | U
| | | | LEAF: FOLIAGE
| | | | $IBC_{k=1}$
| | | | | LEAF: FOLIAGE
| | | | | LEAF: TRUNK
| | | | | LEAF: ROAD
| | | LEAF: FOLIAGE
| | LEAF: FOLIAGE
| LEAF: ROADLINE
| LEAF: TRUNK
LEAF: ROADLINE
LEAF: SKY-TREE
$IBC_{k=1}$

| LEAF: SKY-TREE
| LEAF: TRUNK
| LEAF: ROAD
| LEAF: FOLIAGE
| LEAF: GRASS
LEAF: GRAVEL
LEAF: SKY
$LCT_{i=7}$
| LEAF: FOLIAGE
| LEAF: ROADLINE
| LEAF: GRASS
| LEAF: TRUNK
LEAF: DIRT

C.15 Vowel Recognition

$IBC_{k=1}$
LEAF: 0
LEAF: 1
LEAF: 2
LEAF: 3
LEAF: 4
LEAF: 5
LEAF: 6
LEAF: 7
LEAF: 8
LEAF: 9
LEAF: 10

C.16 Waveform

$IBC_{k=6}$
LEAF: 0
LEAF: 2
LEAF: 1

BIBLIOGRAPHY

Aha, D. W., & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 794-799). Detroit, Michigan: Morgan Kaufmann.

Aha, David W. (1990). *A study of instance-based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations.* Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine, CA.

Aha, D. W. (1991a). Incremental constructive induction: An instance-based approach. *Machine Learning: Proceedings of the Eighth International Workshop* (pp. 117-121). Evanston, IL: Morgan Kaufmann.

Aha, D. W., Kibler, D., & Albert, M. (1991b). Instance-based learning algorithms. *Machine Learning, 6*, 37-66.

Aha, D. W. (1992). Generalizing from case studies: A case study. *Machine Learning: Proceedings of the Ninth International Conference* (pp. 1-10). San Mateo, CA: Morgan Kaufmann.

Ash, T. (1989). *Dynamic node creation in backpropagation networks*, (ICS Report 8901), San Diego, CA: University of California, Institute for Cognitive Science.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

Breiman, L. (1992). *Stacked regressions*, (Technical Report No. 367), University of California, Berkeley.

Brodley, C. E., & Utgoff, P. E. (1992). *Multivariate versus univariate decision trees*, (Coins Technical Report 92-8), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Brodley, C. E., & Utgoff, P. E. (in press). Multivariate decision trees. *Machine Learning.*

Brodley, C. E. (1993). Addressing the selective superiority problem: Automatic algorithm/model class selection. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 17-24). Amherst, MA: Morgan Kaufmann.

Cardie, C. (1993). Using decision trees to improve case-based learning. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 25-32). Amherst, MA: Morgan Kaufmann.

Chatterjee, S., & Price, B. (1977). *Regression analysis by example.* New York: John Wiley and Sons.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*, 261-283.

Cover, T. M. (1973). Enumerative source coding. *IEEE Transactions on Information Theory, IT-19*, 73-77.

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disese. *American Journal of Cardiology, 64*, 304-310.

Dietterich, T. G. (1990). Machine learning. *Annual Review of Computer Science, 4*.

Draper, N. R., & Smith, H. (1981). *Applied regression analysis.* New York: John Wiley and Sons.

Drucker, H., Schapire, R., & Simard, P. (1993). Improving the performance in neural networks using a boosting algorithm. *Advances in Neural Information Processing Systems, 5*, 42-49.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: Wiley & Sons.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory, IT-21*, 194-203.

Fahlman, S. E., & Lebiere, C. (1990). The cascade correlation architecture. *Advances in Neural Information Processing Systems, 2*, 524-532.

Falkenhainer, B. C., & Michalski, R. S. (1986). Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning, 1*, 367-401.

Fayyad, U. M., & Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning, 8*, 87-102.

Feng, C., Sutherland, A., King, R., Muggleton, S., & Henry, R. (1993). Comparison of machine learning classifiers to statistics and neural networks. *Preliminary Papers of the Fourth International Workshop on Artificial Intelligence and Statistics* (pp. 41-52).

Fisher, R. A. (1936). Multiple measures in taxonomic problems. *Annals of Eugenics, 7*, 179-188.

Frean, M. (1990a). *Small nets and short paths: Optimising neural computation.* Doctoral dissertation, Center for Cognitive Science, University of Edinburgh.

Frean, M. (1990b). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation, 2*, 198-209.

Gallant, S. I. (1986). Optimal linear discriminants. *Proceedings of the International Conference on Pattern Recognition* (pp. 849-852). IEEE Computer Society Press.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics, 53*, 203-217.

Hanson, S. J., & Pratt, L. Y. (1989). Comparing biases for minimal network construction with backpropagation. *Advances in Neural Information Processing Systems, 1*, 177-185.

Hocking, R. R. (1986). The analysis and selection of variables in linear regression. *Biometrics, 32*, 1-49.

Honavar, V., & Uhr, L. (1988). A network of neuron-like units that learn to perceive by generation as well as reweighting of its links. *Proc. of the 1988 Connectionist Summer School.*

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation, 3*, 79-87.

Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. on Neural Networks, 1*, 239-242.

Kittler, J. (1986). Feature selection and extraction. In Young & Fu (Eds.), *Handbook of pattern recognition and image processing.* New York: Academic Press.

Kokar, M. M. (1986). Determining arguments of invariant functional descriptions. *Machine Learning, 1*, 403-422.

Langley, P. (1986). Machine learning and discovery. *Machine Learning, 1*, 363-366.

Le Cun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. *Advances in Neural Information Processing Systems, 2*, 598-605.

Lesser, V., Nawab, S. H., & Klassner, F. (to appear March 1995). IPUS: An architecture for the integrated processing and understanding of signals. *Artificial Intelligence.*

Linhart, H., & Zucchini, W. (1986). *Model selection.* NY: Wiley.

Mangasarian, O. L. , & Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News, 23* , 1-18.

Matheus, C. J. (1990). *Feature construction: An analytic framework and an application to decision trees.* Doctoral dissertation, Department of Computer Science, University of Illinois, Urbana-Champaign, IL.

Michalski, R. S., & Chilausky, R. L. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Policy Analysis and Information Systems, 4*, 125-160.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Moret, B. M. E. (1982). Decision trees and diagrams. *Computing Surveys, 14*, 593-623.

Mozer, M. C., & Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Connection Science, 1*, 3-26.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 71-99.

Pagallo, G. M. (1990). *Adaptive decision tree algorithms for learning from examples.* Doctoral dissertation, University of California at Santa Cruz.

Provost, F. J., & Buchanan, B. G. (1994). *Inductive policy: The pragmatics of bias selection* , (Technical Report ISL-94-4), University of Pittsburgh, Intelligent Systems Laboratory, Department of Computer Science.

Provost, F. J., & Buchanan, B. G. (1992). Inductive policy. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 255-261). San Jose, CA: MIT Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies, 27*, 221-234.

Quinlan, J. R. (1989). Unknown attribute values in induction. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 164-168). Ithaca, NY: Morgan Kaufmann.

Quinlan, J. R. (1993). Combining instance-based and model-based learning. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 236-243). Amherst, MA: Morgan Kaufmann.

Rendell, L., Seshu, R., & Tcheng, D. (1987). Layered concept learning and dynamically variable bias management. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 308-314). Milan, Italy: Morgan Kaufmann.

Rendell, L., & Cho, H. (1990). Empirical learning as a function of concept character. *Machine Learning, 5*, 267-298.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry.* New Jersey: World Scientific.

Salzberg, S. (1991). A nearest hyperrectangular learning method. *Machine Learning, 6*, 251-276.

Saxena, S. (1991a). *Predicting the effect of instance representations on inductive learning.* Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

Saxena, S. (1991b). *An algorithm to evaluate instance representations*, (TR-91-21), Amherst, MA: University of Massachusetts, Computer and Information Science Department.

Schaffer, C. (1990). A proven domain-independent scientific function-finding algorithm. *Proceedings of Eighth National Conference on Artificial Intelligence* (pp. 828-833). MIT Press.

Schaffer, C. (1993). Selecting a classification method by cross-validation. *Preliminary Papers of the Fourth International Workshop on Artificial Intelligence and Statistics* (pp. 15-25).

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*, 197-227.

Schlimmer, J. C., & Granger, R. H., Jr. (1986). Incremental learning from noisy data. *Machine Learning, 1*, 317-354.

Schlimmer, J. C. (1987). *Concept acquisition through representational adjustment.* Doctoral dissertation, University of California, Irvine.

Shavlik, J. W., Mooney, R. J., & Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning, 6*, 111-144.

Sutton, R. S. (1988). *NADALINE: A normalized adaptive linear element that learns efficiently*, (GTE TR88-509.4), GTE Laboratories Incorporated.

Sutton, R. S., & Matheus, C. J. (1991). Learning polynomial functions by feature construction. *Machine Learning: Proceedings of the Eighth International Workshop* (pp. 208-212). Evanston, IL: Morgan Kaufmann.

Tcheng, D., Lambert, B., C-Y Lu, S., & Rendell, L (1989). Building robust learning systems by computing induction and optimization. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 806-812). Detroit, Michigan: Morgan Kaufmann.

Towell, G., Craven, M., & Shavlik, J. (1991). Constructive induction in knowledge-based neural networks. *Machine Learning: Proceedings of the Eighth International Workshop* (pp. 213-217). Evanston, IL: Morgan Kaufmann.

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

Utgoff, P. E., & Brodley, C. E. (1990). An incremental method for finding multivariate splits for decision trees. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 58-65). Austin, TX: Morgan Kaufmann.

Utgoff, P. E., & Brodley, C. E. (1991). *Linear machine decision trees*, (COINS Technical Report 91-10), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Weiss, S. M., & Kulikowski, C. S. (1991). *Computer systems that learn.* Palo Alto: Morgan Kaufmann.

Weiss, S. M., & Kapouleas, I. (1989). An empirical comparision of pattern recognition, neural nets, and machine learning classification methods. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 781-787). Detroit, Michigan: Morgan Kaufmann.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*, 241-259.

Yang, D. S., Rendell, L., & Blix, G. (1991). Fringe-like feature construction a comparitive study and a unifying scheme. *Machine Learning: Proceedings of the Eighth International Workshop.* Evanston, IL: Morgan Kaufmann.

Yerramareddy, S., Tcheng, D. K., Lu, S., & Assanis, D. N. (1992). Creating and using models for engineering design. *IEEE Expert, 3*, 52-59.

Zhang, X., Mesirov, J. P., & Waltz, D. L. (1992). Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology, 225*, 1049-1063.