Dynamic Automatic Model Selection

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

COINS Technical Report 92-30
February 1992

Abstract

The problem of how to learn from examples has been studied throughout the history of machine learning, and many successful learning algorithms have been developed. A problem that has received less attention is how to select which algorithm to use for a given learning task. The ability of a chosen algorithm to induce a good generalization depends on how appropriate the *model class* underlying the algorithm is for the given task. We define an algorithm's model class to be the representation language it uses to express a generalization of the examples. Supervised learning algorithms differ in their underlying model class and in how they search for a good generalization. Given this characterization, it is not surprising that some algorithms find better generalizations for some, but not all tasks. Therefore, in order to find the best generalization for each task, an automated learning system must search for the appropriate model class in addition to searching for the best generalization within the chosen class.

This thesis proposal investigates the issues involved in automating the selection of the appropriate model class. The presented approach has two facets. Firstly, the approach combines different model classes in the form of a model combination decision tree, which allows the best representation to be found for each subconcept of the learning task. Secondly, which model class is the most appropriate is determined dynamically using a set of heuristic rules. Explicit in each rule are the conditions in which a particular model class is appropriate and if it is not, what should be done next. In addition to describing the approach, this proposal describes how the approach will be evaluated in order to demonstrate that it is both an efficient and effective method for automatic model selection.

Contents

1 Introduction

A fundamental issue in machine learning is how to learn from examples. Given a set of examples, each labeled with its correct class name, a machine learns by inducing a *generalization* of the examples. For a given learning task there may be many generalizations consistent with the examples. Within the set of feasible generalizations, one generalization is judged better than another if it can be used to classify previously unseen examples with a lower error rate. The quality of a generalization is tied directly to how appropriate the representation of the examples is for the learning algorithm used to induce the generalization. For example, Quinlan spent two months coming up with features that allowed a decision tree algorithm to induce a good generalization for the chess-end-game problem (Quinlan, 1983). Indeed, throughout the study of Artificial Intelligence it is a well know problem that a machine's ability to reason is tied directly to its representation of a problem.

Since the beginning of the study of Artificial Intelligence at the Dartmouth Conference in 1956, researchers have studied how a machine can learn, and many successful learning algorithms have been developed (see Dietterich, London, Clarkson and Dromey (1982) for a history of the field). A problem that has received less attention is how to select which learning algorithm to use for a given task. Typically, when given a learning task, a human selects an input representation for the examples and a learning algorithm. The ability of the chosen algorithm to induce a good generalization depends on how appropriate the *model class* underlying the learning algorithm is for the given task. In science, a model is defined as "a representation of some or all of the properties of a device, system or object" (Van Nostrand, 1989). In the context of machine learning, we define a model to be a generalization of the data. A model class is the representation language used by the learning algorithm to induce a generalization. For example, the underlying model class of a decision tree algorithm, such as ID3 (Quinlan, 1986a), is the Disjunctive Normal Form (DNF). The underlying model class of a perceptron learning algorithm (Minsky & Papert, 1972) is linear discriminant functions.

Given a model class, inductive learning can then be viewed as finding the model within the chosen class of models that best fits the data. For example, ID3 fits a DNF hypothesis to the data. The absolute error correction procedure (Nilsson, 1965) fits a linear threshold function to the data by learning its weights. We view *model fitting* as the process in which a learning algorithm searches for the best model within a model class to represent a generalization of the data. In contrast, we view *model selection* as the process of searching for the appropriate model class.

We can characterize learning algorithms by their underlying model class and their bias for searching for a generalization. Given a particular learning task, the *generalization space* an algorithm searches is defined by both the input representation for the task and the underlying model class of the algorithm. Given this characterization, it is not surprising that some algorithms find better generalizations for some, but not all tasks. Therefore, although there are benefits to creating algorithms that work well within a circumscribed task area, if we are to automate learning fully, then in addition to searching for the best generalization within a chosen model class, a learning algorithm must also search for the appropriate class of models.

Typically, humans perform model selection by selecting which learning algorithm to use. Continuing to require humans to perform this task goes against a central objective of machine

learning, which is to remove the human from the learning process; a machine should select the appropriate model class. We refer to this process as *automatic model selection.* One simple way to automate model selection would be to create a system that induces a set of n generalizations, one for each of n different learning algorithms. Such a system would then choose the best generalization from this set as its output. The drawback of such a system would be its inefficiency; for a given algorithm the number of possible generalizations it can find can be exponential in the number of input features. The problem of selecting the appropriate model is not restricted to the selection of which representation language to use, but extends to the identification of where in the specified generalization space the algorithm should focus its efforts. Ideally one should have some indication of the appropriate model before expending the effort required to induce a generalization.

We are not alone in our view that model selection should be automated. Systems that perform *constructive induction* attempt to find a better model by changing the input representation. This has the effect of changing the generalization space for the given task, because a generalization space is defined by both the initial features and an algorithm's representation language. Rendell and Cho show how to improve learning by transforming the instance space to decrease the number of distinct concept regions, which increases concept concentration (Rendell & Cho, 1990). Adaptive neural net algorithms seek the appropriate model for a given task by searching for the appropriate network architecture dynamically (Gallant, 1986; Ash, 1989). Hybrid algorithms seek to combine the strengths of different algorithms by selecting among a set of model classes. Indeed, we can trace this view back as far as 1968 when Amarel showed that by changing the representation of a problem one changes a system's ability to find a solution (Amarel, 1968). Amarel suggests that we should design problem solving procedures such that they can find a point of view of the problem that maximally simplifies the process of finding a solution.

Previous endeavors employ search strategies with a fixed bias; they impose an order on the search through the *model space.* An algorithm's model space is the set of all models considered by the algorithm. A model space is the union of one or more model classes. If automatic model selection is to be feasible, then the search for the appropriate model must be efficient. There are several reasons that a fixed search strategy is inefficient. Firstly, if the appropriate model class is the last to be tried, then the search strategy has degenerated into trying n model classes and selecting the best of the n generated generalizations. Secondly, it is unclear how to fix the order in which models are tried. The best order for an efficient search depends on the particular learning task. Thirdly, any fixed bias cannot be optimal for all learning tasks. The problem lies in the fact that evaluating the appropriateness of a model class is expensive. Typically the evaluation requires that the system fit a model from the candidate model class to the data and then examine the resulting classifier. Finally, it is unclear when searching through the model space how one knows when to stop searching; perhaps the best model is the next in the ordering.

The approach we will take to the problem of automatic model selection differs from previous approaches in one fundamental way: we believe that the search strategy should be *dynamic.* A dynamic search strategy is capable of varying the order in which models are considered, using measures of the learning process to guide the search for the appropriate model. Rather than fix the bias for searching the model space, a system would try one model, and if the model is inappropriate, it would use heuristic measures of the first attempt and of the data to choose the next candidate model. To realize the benefits of automatic model

selection, the search for the appropriate model must be efficient. Therefore, the hypothesis that this research seeks to validate is:

A dynamic search control strategy is an effective and efficient approach to automatic model selection.

We propose to investigate the issues involved in creating a dynamic search algorithm for automatic model selection by implementing a system that uses heuristic measurements of the data and the learning process to guide its search strategy. The model space searched by our system is the combined model classes of linear discriminant functions, decision trees and instance based classifiers. We choose these model classes because previous research has illustrated that each has different strengths (Schlimmer, 1987; Pagallo, 1990; Utgoff, 1989; Utgoff & Brodley, 1990; Aha, 1990; Brodley & Utgoff, 1992). Because the ease with which the different representation languages can describe a particular concept varies, combining them allows the system to learn good generalizations for a wider class of learning tasks (Utgoff, 1989). To validate our hypothesis we will evaluate the implemented system empirically to demonstrate that it adjusts its search bias effectively. The expected result of the evaluation is: the proposed system will be capable of determining the appropriate model for any concept, if it can be represented by a model in the proposed model space. This will demonstrate that a dynamic control strategy for automatic model selection is more effective than a static control strategy, and is therefore a better approach to the problem of automatic model selection.

The rest of the proposal is organized as follows. In Section 2 we describe previous approaches to automatic model selection and point out why these approaches have not satisfactorily solved the automatic model selection problem. In Section 3 we expand our approach to automatic model selection and detail a set of heuristic measures for the implementation of this strategy. Finally in Section 4 we outline the proposed research, the expected results of this research and their implications for automated learning.

2 Existing Methods for Model Selection

Throughout the history of science, researchers have examined empirical data to form a model of the phenomenon under study. They have used methods such as regression analysis and dimensional analysis to guide them in their search for a model that explains or predicts the data. Many of the techniques of these methods aim to help the researcher find the best model from a specified model class to determine if the model fits the data; the choice of which model class to examine has fallen on the shoulders of the researcher. In this section we focus on techniques that automate the model selection process itself. In Section 2.1 we present a brief introduction to statistical methods for model selection and in Section 2.2 we describe approaches in machine learning for automated model selection. Our reason for dividing the existing approaches into these two categories stems from the motivation of each discipline for creating methods for automated model selection. In the field of statistics, techniques for automated model selection were created to *aid* the human analyst select the appropriate model, whereas in the field of machine learning they were created to *replace* the human analyst in the model selection process.

2.1 Statistical Approaches

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 term, ε, is unobserved and represents the underlying disturbance process of the data. ε is the amount by which any individual observation, y_i, may fall off the regression line. 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 a 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 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 first 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 model selection procedures have been created. There are several methods for finding the appropriate model, and all require specification of the entire set of terms 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. In our view of model selection, specifying the terms to be considered is equivalent to specifying the model space to be searched. We present the three most commonly used procedures for finding the appropriate model. A more comprehensive list can be found in Draper and Smith (1981).

The first procedure evaluates all possible equations (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 variables 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 variables in the model and adds one variable at a time until either all variables are in the model, or until a stopping criterion is satisfied. The first term to be included is the term which 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 which has the highest correlation with y, after y has been adjusted for the effect of the first variable, i.e., the variable 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 variable to be eliminated is the one that makes the smallest contribution to the reduction of 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 variables is retained in the model. After each variable is eliminated, the equation is refit 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 $y = (\beta_1/\beta_2)(e^{\beta_1 t} - e^{\beta_2 t}) + \varepsilon$ 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 model 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.

2.2 Machine Learning Approaches

In this section we describe learning systems that select an appropriate model class for a given data set automatically. Previous work in automatic model selection can be divided into four categories, based on a system's control strategy for searching the model space:

1. **Initial model selection:** Given a finite set of learning algorithms, each with a different underlying model class, a selection algorithm selects the learning algorithm that it judges most appropriate for the given learning task.

2. **Concept form:** These methods use the form of the learned classifier to refine the learning algorithm's underlying model class. The algorithm repeats the following cycle until it appears as if the best model has been obtained: induce a classifier and then evaluate the form of the classifier to change the model.

3. **Error rate:** These methods adjust the model using the performance of the current classifier to guide the search through the model space. A system starts with an initial model class and induces a classifier using this class. Then after examining the error rate of the classifier, the system either retains the current classifier or induces a new classifier using the next model class. Which class is tried next is determined by the system's *a priori* search bias.

4. **Relationships among the variables:** Search for the appropriate model is guided by measuring the relationships among the variables of the given data set. These systems search iteratively for relationships among the variables, propose a generalization based on these relationships and then test the proposed generalization to see if it fits the data. If it does not, then they resume searching for new relationships using the input variables and the relationships found in the previous iterations.

This characterization allows us to examine the differences in the control strategies for searching the model space. In the discussion of each system we will answer the following questions:

- What is the model space?
- How is the search ordered through the model space?
- How does the system determine when to stop searching?

2.2.1 Initial Selection

An algorithm that performs initial model selection attempts to determine which model class, from a finite set of possible model classes, best fits the given learning task. The decision procedure examines each learning algorithm and then chooses the one that appears most promising for the given data set; the choice is made before the actual learning begins. To date, we know of no such procedure. In this section we present an algorithm, which although not originally designed to perform initial model selection, can be modified to perform this task.

An algorithm, ACR, 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 model selection (Saxena, 1991a). Given one input representation, ACR would select which of two model classes, or 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. Although finding provably optimal codings is unsatisfiable, ACR could be used as a *heuristic* method for model selection by using the best known codings.

2.2.2 Using Concept Form

One approach for changing the underlying model class of a learning system is to evaluate the classifier induced from a set of training instances, described in an initial vocabulary, and then change the model class 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 model class of the learning system. 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 the 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 $feature_j$. As the feature set becomes more complex so does the underlying model class.

FRINGE constructs Boolean combinations of the initial set of features to overcome the *tree replication problem* (Pagallo & Haussler, 1990). Given positive and negative examples of a concept, each described by the same set of Boolean features, a decision tree can represent a DNF hypothesis of the examples. 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 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 (Pagallo, 1990). LINT learns (0,1)DLF functions (i.e., disjunctions of linear threshold functions such that each non-zero weight has a magnitude 1). 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. The weight for a feature at a node in which both children are leaves is 1. If one child is leaf and the other is a test-node, then the weight is 1 if the leaf is labeled negative and -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 as 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; 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, 1986a) to a set of instances described by only the nine initial features.

2.2.3 Using the Error Rate

In this section we describe learning systems in which a model is selected, or a new one proposed, in response to measures of the error rate of the current classifier. These systems monitor the learning process, typically using some measure of the current classifier's ability to classify examples of the concept, to change the underlying model class. Automatic model selection takes place during learning; previous learning is not discarded, but is incorporated

into the new model.

Adaptive Neural Net Algorithms:

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. Because 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 the 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.

Systems that start with a large net and remove units/connections bias the search through the model space from complex to simple architectures. A net's complexity is measured by how many units and connections it has. Such systems search a finite model space. In contrast, systems that start with a small net and add units/connections search a potentially infinite model space. However, this problem is not as severe as it appears because if the instance space is finite, then there is an upper bound on the number of units required to classify all of the training instances correctly.

Hybrid Representations:

In this section we describe systems that mix different formalisms to produce a *hybrid representation* (Utgoff, 1989). A hybrid representation permits the learning algorithm to select the appropriate formalism for each *subconcept*, drawing on the special strengths of each of the individual model classes that make up the hybrid. A subconcept is defined by the partially learned classifier. We focus our discussion on algorithms that choose among different formalisms. Examples of systems that combine various properties of different algorithms can be found in Clark and Niblett (1989), and Towel, Craven and Shalik (1991).

One hybrid representation, a *perceptron tree*, combines a 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 model classes, 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 (Brodley & Utgoff, 1992). PT2 induces decision trees in which each node in the tree is an LTU. At each node in the tree, the algorithm trains an LTU based on all n of the input variables. When a good set of weights has been found, as measured by Gallant's pocket algorithm (Gallant, 1986), the system then tries to eliminate variables irrelevant to classification. It trains n LTUs, each based on $n - 1$ variables (the result of eliminating a different variable each time). This greedy search procedure continues until further elimination will decrease the classification accuracy of the LTU. At that point, if necessary, the algorithm grows two subtrees, one for each branch, and the procedure repeats at each of the subtrees.

LMDT, a successor to PT2, places a linear machine (Nilsson, 1965) at each node in the tree. A linear machine is a set of k linear discriminant functions, which together can discriminate among k different classes. Like PT2, LMDT searches for a good multivariate split by first training a linear machine based on all n of the input variables. However, the method by which irrelevant variables are eliminated is different. LMDT eliminates variables

one by one, choosing the next candidate for elimination by using the *dispersion* of the weights for each variable. The dispersion measure for a variable is calculated by summing the squared magnitude of each weight corresponding to that variable in each linear discriminant function of the linear machine. The variable whose weights have the smallest dispersion makes the smallest contribution to discriminating the instances at that node, and is therefore the best candidate for elimination. If the accuracy drops below a certain level, then the search terminates and the most accurate linear machine observed thus far is placed as a test for that node.

The search control strategies of all three systems fix the order for considering which model class to use. The perceptron tree algorithm chooses between an LTU, based on all n of the input features, and a single attribute test. PT2 and LMDT have a graded search strategy. Like the perceptron tree algorithm, they both start searching from an LTU (or a linear machine) based on all n of the input features, but they can select an LTU based on m input features, $m \leq n$ Therefore, PT2 and LMDT search a larger model space than the perceptron tree algorithm. The method by which each algorithm terminates search for the appropriate model is related to some measure of the error of the current model. The perceptron tree algorithm chooses an LTU if the set of instances that it is trained from are linearly separable. PT2 and LMDT use measures of the error rate of the current LTU (linear machine) to determine when to stop searching.

Incremental Algorithms:

In this section we describe two incremental concept learning algorithms, STAGGER (Schlimmer, 1987) and IB3-CI (Aha, 1991), which adjust the concept formalism 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. In addition to learning incrementally, the two systems adjust the concept model or formalism during the learning process. The method by which both STAGGER and IB3-CI change the underlying model class is different than those used by adaptive neural net or hybrid algorithms. Rather than change the model by examining the classifier after some subset of the training instances has been observed, these systems can adjust the concept formalism in response to just one instance: a change in the formalism 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 if 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.

Fitting Equations to Data:

In this section we describe two systems that seek to find a mathematical equation, $y = f(x_1, x_2, ..., x_n)$, of the independent variables, $x_1, x_2, ..., x_n$, which predicts the value of the dependent variable, y. In these systems, a model is the functional form, $y = k_0 + k_1 Z_1 + ... + k_p Z_p$, where each of the Z_i is a function of the input variables. Fitting a model requires finding the values for the coeficients, k_i of the functional form. Both systems use the error rate of the classifier induced using the candidate model to guide the search through the model space.

Schaffer's E^* algorithm searches a finite space of candidate models. 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 which 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.

Sutton and Matheus present a method for learning higher order polynomial functions from examples using linear regression and feature construction (Sutton & Matheus, 1991). Their system begins by performing a regression on the training set to learn a regression equation for the model $y = k_0 + k_1 x_1 + ... + k_p x_p$, 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 out 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.

Optimization:

Tcheng et al. (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 v-fold crossvalidation (the user specifies the choice of the error 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. T_1, T_2 and T_3 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 performance measure direct search to points that have a low error rate. It is unclear how they determine which points in the bias space 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. It is unclear how large n is and how its value affects further search in the bias space. How long the system spends with one learning strategy is a user controlled parameter. The system halts when CRL's stopping criteria are met.

2.2.4 Measuring Relationships Among the 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 prespecified

degree of accuracy. Such systems have been called numerical discovery systems (Langley, Bradshaw & Simon, 1983). These systems search the model space using 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 node 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 prespecified 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 breadth-first 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 than 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.

2.3 Limitations of the Existing Work

The approaches to automatic model selection described in Section 2.2 were categorized by examining each system's control strategy for searching the model space. The purpose of this section is to discuss the limitations of these systems as solutions to the automatic model selection problem. Specifically, each of the surveyed systems suffers from one or more of the following limitations:

- Methods that choose initially among a finite set of model classes cannot take advantage of information gained during the learning process.
- Systems that change the model space by forming Boolean combinations of the initial input features are not sufficient for problems in which the features are related numerically.
- Search methods that optimize accuracy do not have the ability to determine if the best model has been found unless they try all possible models.
- A control strategy that searches the model space in a prespecified order is inefficient because in the worst case all possible model classes will be considered; any static bias

for searching the model space will be inappropriate for some tasks.

- Methods that employ an exhaustive search strategy for searching the model space are intractable for problems with large initial feature sets.

Methods that perform initial selection choose the model class that appears to be best for the entire instance space. Assuming a divide-and-conquer approach to learning, a model class that is appropriate for one *subspace* of the concept to be learned may be inappropriate for another one. Subspaces of the concept to be learned are formed by the partition of the instance space defined by a partially learned classifier. For example, consider the following concept: heart attack caused by a weight problem. A heart attack can be caused by a severe overweight problem, obesity, or a severe underweight problem, anorexia (anorexic patients typically have a potassium deficiency due to lack of food, which in turn causes heart failure). 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. An LTU is not appropriate for the entire concept space because the space is not linearly separable, as can be seen in Figure 1. Moreover, a symbolic decision tree algorithm would have difficulty learning a compact generalization for this concept as it would need to approximate the hyperplanes with a series of splits orthogonal to each of the axes.

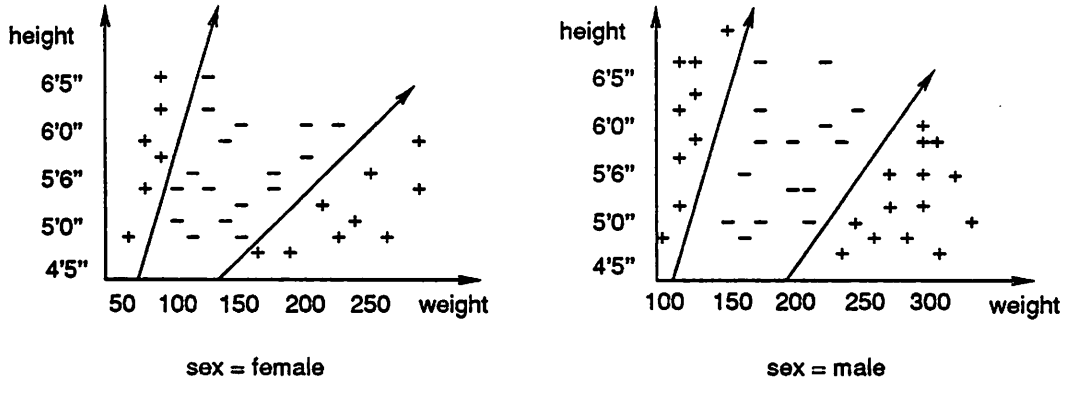

Figure 1. The concept heart attack; "+": positive instance, "-": negative instance.

A better solution (shown in Figure 2) is a hybrid formalism that combines LTUs with decision trees. This example illustrates that choosing initially between a decision tree and an LTU will not yield as succinct and accurate a classification rule as combining both model classes and employing a control strategy to choose between them, for each subspace of the instance space. One solution to this problem would be to select initially among decision trees, LTU's and an algorithm based on the combination of decision trees and LTU's. However, this is unnecessary if the combination algorithm is capable of making the correct choice at each subspace. Moreover, this example illustrates utility of an algorithm that can take advantage of information gained during the learning process. Because different models may be appropriate for different subspaces, the appropriate model for each subspace can be learned separately.

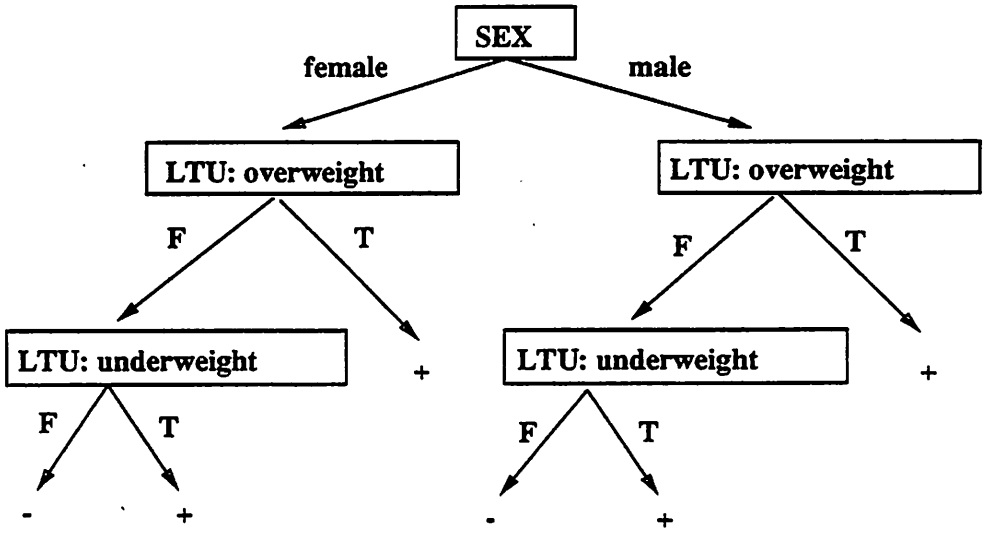

Figure 2. Classifier induced for the concept heart attack using a hybrid formalism.

One limitation of methods that use the concept form to create Boolean combinations of the input features is that they search the model space using a fixed bias. A fixed bias will be inappropriate for some tasks and therefore is inefficient. Another drawback is that the feature combination space is exponential in the number of initial attributes. One would like to restrict search of this space to situations for which there is evidence that Boolean combinations are required or useful. Finally, the simple heart attack example given above illustrates a need for feature formation techniques that use numeric construction operators in addition to Boolean operators for some tasks.

The systems described in Section 2.2.3 come closest to a solution to the automated model selection problem. However, they all suffer from one limitation; they have a predetermined bias for searching the specified model space. Adaptive neural net algorithms either start with a small net and add units/connections or they start with a large net and delete units/connections. One problem with these approaches is that it is unclear, for a given task, how to determine which of the two approaches will produce the better classifier. We propose that a better solution, albeit difficult to implement, would examine the current net and, rather than add or delete *one* unit/connection, figure out how many units/connections to add or delete. In addition, the ability to add *and* delete units/connections would allow one to search in both directions and correct previous choices of how many units/connections to add or delete. Our principle reason for focusing our research on symbolic learning is that empirical results have illustrated that neural nets, trained using the BACKPROP algorithm, require significantly more training instances than symbolic algorithms to converge to a satisfactory generalization. Studies have shown that BACKPROP takes much longer than ID3 on the same tasks (Fisher & McKusick, 1989; Mooney, Shavlik, Towell & Gove, 1989). For these test cases the error rates of ID3 and BACKPROP were not significantly different. When noise was added to the test cases, BACKPROP outperformed ID3. However, it is

unclear how much of this difference should be attributed to the pruning methods used in the particular implementations of ID3. In general, neural net algorithms search a much larger space than most symbolic algorithms as they can learn *arbitrary* boundaries in the feature space. Many tasks do not require this power, and restricting the search space permits one to find a solution in less time.

Systems that use a hybrid representation permit different subspaces to be represented with different formalisms. Combining decision trees with linear threshold functions allows one to draw on the strengths of both formalisms (Utgoff, 1989). However, the control strategies of perceptron trees, PT2 and LMDT implement biases that may be inappropriate for some learning tasks. In the original 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. PT2 and LMDT sought to overcome this second limitation. However, both PT2 and LMDT bias the search for a multivariate split at a node toward a linear combination based on all of the input variables. Although variables are eliminated, the procedures suffer a problem inherent in all hill-climbing procedures: they can get stuck on local maxima. If the best solution is a univariate test, the variable elimination procedures of both PT2 and LMDT are not guaranteed to find that test. Moreover, starting from a test based on all n input variables and then eliminating variables one by one is wasteful if the best test is indeed univariate. In summary, the problem with these three systems is that they implement a fixed bias in which they consider possible models. In the worst case this is equivalent to trying all model classes and is therefore an inefficient solution to automatic model selection. A better solution would try to determine, before too much effort has been expended, which model class is best and would then direct search to the appropriate place in the generalization space.

Tcheng's system (Tcheng, Lambert, C-Y Lu & Rendell, 1989), which searches for the best model by optimizing novelty and performance, has several limitations. Firstly, if the model with the best accuracy is to be found, then it must evaluate all models. Secondly, the system does not use information about why a particular model fails, and therefore must search randomly. Finally, 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.

The quantitative discovery systems described in Section 2.2.4 search for the equation that best represents the relationships among the variables of the given data set, based on numerical relationships in the current feature set. All three systems, BACON, ABACUS and COPER suffer from one severe limitation: they are potentially exhaustive. Therefore, for problems that have a large initial feature set they are intractable.

In summary, the systems surveyed in this section attempt to select the appropriate formalism automatically. However, none of the surveyed systems presents a full solution to the automatic model selection problem. In the next section we discuss the desired properties such a solution should have.

3 A New Approach to Automatic Model Selection

In the previous section we surveyed learning systems that attempt to select the appropriate formalism automatically. However, none of the surveyed systems presents a satisfactory solution to the automatic model selection problem. In this section we present a new approach to this problem that differs from previous approaches in one fundamental way: the control strategy for searching the model space is *dynamic* rather than static. A dynamic control strategy does not consider candidate models using a predetermined static bias, rather it guides the search in response to heuristic measures of the learning process and of the data. In Section 3.1 we describe the desired characteristics of a system that selects the appropriate model automatically and show how these characteristics necessitate a dynamic control strategy. In Section 3.2 we propose a new system that will have these characteristics.

3.1 Desired Characteristics of a Model Selection System

Creating a system that selects an appropriate model for a given data set requires specifying the model space, a control strategy for determining which model to select in that space and methods for fitting models from each model class to the data. Given these requirements there are three desired characteristics of a system that performs automatic model selection:

1. The model space should be as large as possible.
2. The search control strategy must be effective and therefore dynamic.
3. The system must be capable of recognizing when the best model has been found without examining all candidate models.

One would like the model space to be as large as possible, thereby increasing the system's ability to find the best model for a given task; if the best model is not in the specified model space, then clearly, the system has no ability to select this model. However, increasing the number of candidate models can, in turn, increase the computational effort required to select the best model. Therefore, the control strategy for searching the model space should be *effective*.

An effective control strategy has two desirable properties. Firstly, it must be capable of identifying the appropriate model class for a given task. Secondly, the search for this class must be efficient: it should examine as few candidate models as possible. We can measure the efficiency of a control strategy by examining the percentage of the model space it considers before finding an appropriate model. One that considers model classes in an *a priori* fixed order is not efficient, because in the worst case it degenerates into an exhaustive search. Therefore, to search a model space effectively, the order should not be fixed.

A dynamic control strategy fits a model, from one of the model classes to the data and then examines how well the model fits the data. The system either determines that the model is appropriate or decides which model class to try next. Clearly, the search must begin somewhere and, in the absence of any *a priori* knowledge about the data, the choice of which model class to try first must be fixed. However, this choice should not be arbitrary; the first model class should be chosen such that, if it is inappropriate, the system can determine *why* it is inappropriate and from this can make a good choice for which class to try next.

There are many forms of information that the system can use to determine which model class to try next, depending on which models the system has already tried. The simplest

form is the accuracy of the classifier resulting from fitting a model from the class to the data. A low accuracy indicates that the model class is inappropriate, but not necessarily why. Understanding why a particular model class is inappropriate requires examining the form of the classifier. What the system examines depends on the particular model class, and in Section 3.2.2 we describe several different heuristic methods for determining which model to try next.

Recognizing when the best model has been found should not be based solely on the corresponding classifier's predictive accuracy. The reason stems from the fact that a good generalization is both accurate and concise. One generalization is more concise than another if it can be represented in a smaller number of bits (Rissanen, 1989). Blumer, et al. (1987) have shown that these two characteristics are related; given two classifiers that both classify the training instances with the same degree of accuracy, the smaller of the two leads to better predictive accuracy for previously unseen instances. Therefore, a system that searches for an appropriate model should try to find the model class that yields the simplest classifier with the highest degree of accuracy. One cannot have complete confidence that a system has obtained the best model in the model space unless it tries each model class. However, examination of a classifier can yield indications that the underlying model class is appropriate. The system can examine both the predictive accuracy and the form of the classifier induced using the chosen model class.

3.2 A Proposed System

In order to demonstrate that a system can perform automatic model selection in a practical manner using a dynamic control strategy, we propose to build such a system. Specifically, in Section 3.2.1 we propose a particular model space, and detail our reasons for this choice. In Section 3.2.2 we propose a set of methods for searching the proposed model space effectively.

3.2.1 Model Space

The proposed system will search a *combined* model space of linear discriminant functions, decision trees and instance based classifiers. Rather than choose one model class to use for the entire instance space. the system will have the ability to combine the different classes, thereby forming a hybrid representation. A hybrid representation is the result of mixing one or more individual formalisms or statements within those formalisms (Utgoff, 1989). In choosing which model classes to combine one desires that the individual model classes' strengths are complementary. Concepts or subconcepts that are difficult to represent well in one formalism may be easy to represent in another. A system that can combine different formalisms can represent succinctly and accurately a larger class of concepts. In this section we first describe each of the individual model classes and their corresponding algorithm for fitting them to the data. We then outline how these individual model classes can be combined to form a hybrid model.

The individual models:

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, 1986a). One approach to constructing decision trees is to use information theory to select the 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).

A univariate decision tree is biased toward concepts that can be expressed as Boolean combinations of the initial input features that describe each instance. The tests in a univariate decision tree split the instance space with cuts orthogonal to each of the axes, forming a set of hyper-rectangular regions, each labeled with a class name. However, if the concept to be learned is not represented easily by a set of hyper-rectangular regions, then a univariate decision tree algorithm will produce a classifier that is a poor generalization of the concept.

A *linear machine* (Nilsson, 1965) is a set of k linear discriminant functions that are used together to assign an instance to one of the k classes. Let $\mathbf{Y}$ be an instance description, also known as a pattern vector, consisting of a constant threshold value 1 and the numerically encoded features by which the instance is described. Each discriminant function $g_i(\mathbf{Y})$ has the form $\mathbf{W}_i^T\mathbf{Y}$, where $\mathbf{W}$ is a vector of adjustable coefficients, also known as weights. A linear machine infers instance $\mathbf{Y}$ to belong to class i if and only if $(\forall j, i \neq j)\ g_i(\mathbf{Y}) > g_j(\mathbf{Y})$.

In general, to train a linear machine, one adjusts the weight vectors $\mathbf{W}$ of the discriminant functions g in response to any instance that the linear machine would misclassify. This is done by increasing the weights of $\mathbf{W}_i$, where i is the class to which the instance belongs, and decreasing the weights of $\mathbf{W}_j$, where j is the class to which the linear machine erroneously classifies the instance. Adjusting the weights of a linear machine changes the location of the hyper-plane, representing the boundary between class i and class j, in Euclidean *n-space*, where n is the number of vector components. If the instances are linearly separable by a linear machine, then the above training procedure, with a suitable error correction rule as described below, will find a solution machine in a finite number of steps (Duda & Hart, 1973).

A linear machine is biased toward concepts that are linearly separable. If, however, the space is not linearly separable then a linear machine, 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 linear machine even when the space is not linearly separable. The LMS training rule seeks to minimize the squared length of the error vector; it tries to minimize the overall error rather than just focusing on misclassified instances (Duda & Hart, 1973). Gallant's pocket algorithm searches for the set of weights that produce the most consecutive correct classifications when trained using instances drawn randomly from the training set (Gallant, 1986). The thermal training rule 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 will use their method to train linear machines in the proposed system.

A special case of a linear discriminant function is a Boolean combination feature; it is a linear discriminant in which all the weights are of magnitude 1 and the threshold is zero. Such features are formed by combining some subset of the current set of features using the Boolean AND operator. The current set of features is the union of the set of initial features

and any Boolean combination features that have been discovered previously.

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. Typically, to compute how near one instance is to another, the Euclidean distance between the two instances is used. To handle symbolic data we will encode symbolic variables as propositional variables. In order that all variables contribute equally to classification decisions we will normalize variables using standard normal form, i.e., zero mean and unit standard deviation.

The Hybrid Model:

The three model classes can be combined to form a hybrid model, which we call here a *model combination decision tree (MCDT)*. This hybrid is the union of the model spaces searched by k-NN, FRINGE, CITRE, perceptron trees, PT2 and LMDT (Duda & Hart, 1973; Pagallo, 1990; Matheus, 1990; Utgoff, 1989; Utgoff & Brodley, 1990; Brodley & Utgoff, 1992). We define an MCDT as either a leaf node containing a classification or a test with for each *outcome* of the test, a branch to an MCDT. A test can be a univariate symbolic test, a Boolean combination of the initial features, a linear machine or an instance based classifier. A univariate symbolic test has a branch for each observed value of the attribute. A Boolean feature can take on two values, TRUE and FALSE, and has a branch for each value. A linear machine has k outcomes, one for each of the k observed classes. A linear machine can be based on i variables, $i \in \{1, ..., n\}$, where n is the number of input features. Finally, like a linear machine, an IBC has k outcomes, one for each of the k observed classes.

Combining all three individual model classes overcomes the representation problems of each of the individual classes. A concept need not be linearly separable as an MCDT 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. An MCDT classifier can represent both Boolean and non-Boolean concepts by placing linear machines as tests at the nodes, allowing the system to find more compact representations by solving the tree-replication problem (Pagallo & Haussler, 1990). Instance based classifiers can represent non-hyperplane boundaries and therefore can represent instance spaces that are not linearly separable. Finally, instance based classifiers are well suited to instance spaces in which there are few observed instances, which can happen near the leaves of a decision tree. In such cases, a multivariate test will underfit the instances (Duda & Hart, 1973) and the information metric used to select a univariate test will perform poorly (Aha, 1990).

3.2.2 Control Strategy

In this section we propose a control strategy for searching the model space described in the previous section. The control strategy is a rule-based system in which the rules use heuristic measurements to direct the search of the model space. Explicit in the rules are the situations in which a particular model class is either appropriate or inappropriate. To determine the appropriateness of a model class, the rules measure the accuracy, simplicity and form of a generalization resulting from fitting a model from the class to the examples. If a particular model is inappropriate, then the system chooses the next candidate model by examining why the previous choices were inappropriate.

Given a set of instances the system performs a best-first search for the appropriate model

class, guided by the heuristic rules. Nodes in the system's *search tree* are generalizations induced using a particular model. To decide which nodes to expand, which nodes to prune and to determine when the best model has been found the system uses the heuristic rules. The condition part of each rule is compared to the contents of the model memory and to measures the contents computed by a set of measurement algorithms. The action part either invokes a fitting algorithm or halts the search because the best model has been found. The rules will be constructed such that at any given time only one rule's condition part will match the state of the model memory. A preliminary set of rules is described in Section 3.2.3.

In Figure 3 we show the architecture of the proposed system. The five components of the system are shown in the boxes. The flow of information and control are shown by directional line segments. The components are:

- **Model Memory:** Initially the memory is empty. During the search for an appropriate model the memory holds all of the candidate models induced by the fitting algorithms. After the system finds the most appropriate model the other candidate models are deleted, and the appropriate model becomes a node in the MCDT classifier. During the search some candidate models will be more promising than others, and they are stored in a special part of the memory, which we call the *current set.*

- **Fitting Algorithms:** For each model class in the system's model space there is an associated fitting algorithm.

- **Measurement Algorithms:** The condition part of each rule measures aspects of the *current set* , which are computed by a set of measurement algorithms.

- **The Rule Base:** The set of heuristic rules used by the system to guide the search for the appropriate model is stored in the rule base. The rule base includes a set of initial rules to start the search, rules to determine if a model is appropriate and rules to focus the search if a model is inappropriate.

- **The Matcher:** Search for the appropriate model is determined by which rule matches the current state of the model memory. The matcher examines the contents of the model memory using the measurement algorithms to find the matching rule from the rule base. Once the matching rule has been found the system fires the action part of the rule. The action part of the rule either invokes a fitting algorithm or halts the search because the best model has been found.

The system searches at each node for the best model class , and then partitions the data using the generalization that results from fitting the model to the data. The system then recurses with each resulting partition. The system's control strategy at a node is shown in Table 1. To begin the search the system matches characteristics of the instances to the condition parts of a special set of rules, called the *initial rule set.* The action of the matching rule invokes a fitting algorithm. When the fitting algorithm has finished control is passed back to the rule matcher. The system then repeats the following three steps (steps 4-6) until the appropriate model has been found: match the condition part of each rule in the rule base to the current state of the model memory, invoke the action part of the matching rule and check if this action determines which is the best model, by checking whether the system variable *model* has been set. Once the system has found the best model it uses the selected

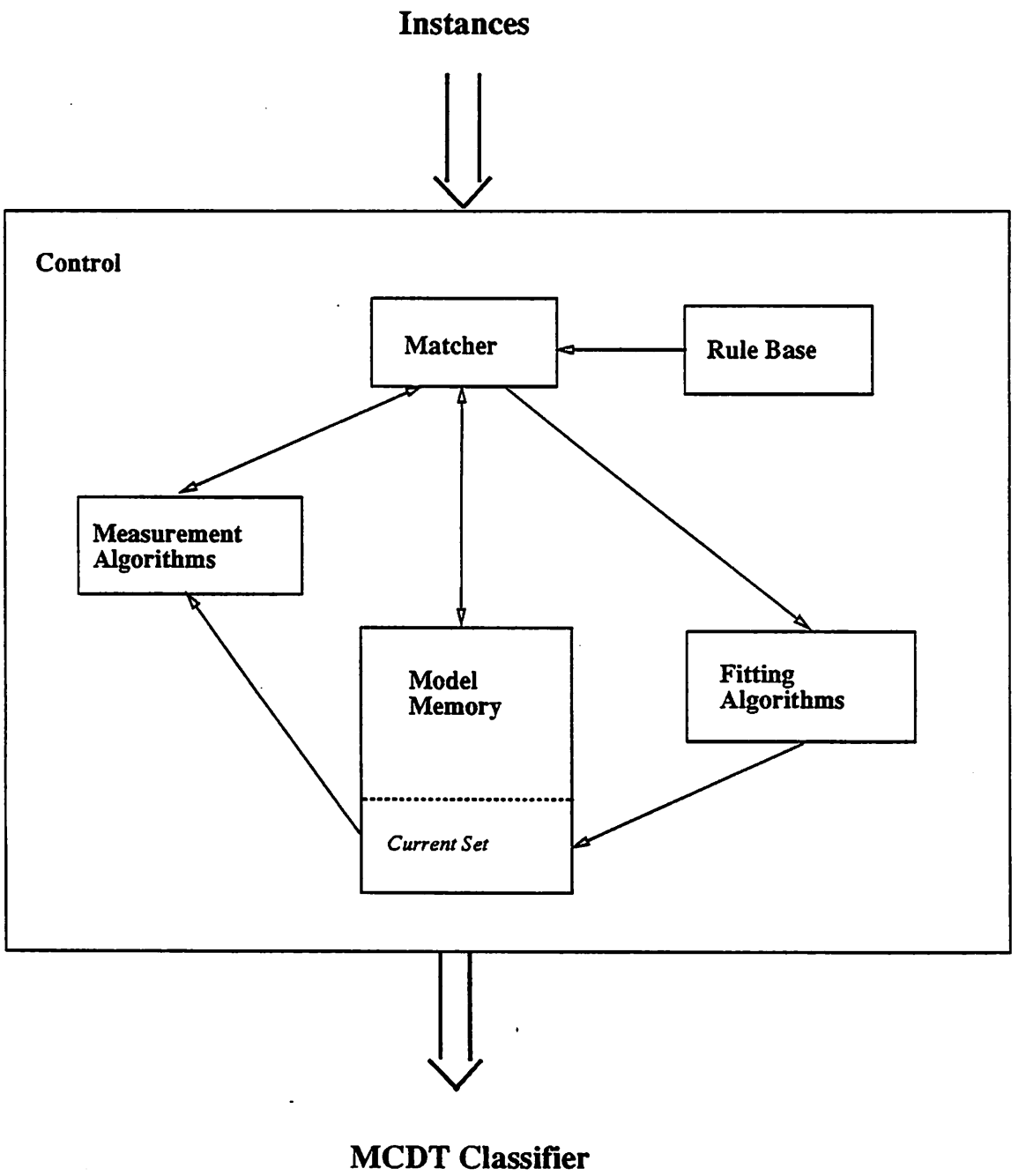

Figure 3. The system architecture

Table 1. Control Strategy

1. If all the instances are from one class, then set *model* to be a leaf node and label it with the class name, return.
2. Otherwise, set *model* to NIL
3. Use the *initial rule set* to determine which model to fit first, and invoke the action part of the matching rule.
4. Find the rule whose condition part is satisfied.
5. Invoke the action part of the matching rule.
6. If *model* is NIL, then goto step 4.
7. Partition the set of instances using the selected model and recurse with each partition.

model to partition the instance space and then recurses with each partition. The system halts and outputs the MCDT classifier when each leaf node of the MCDT contains instances from only one class.

3.2.3 Rule Base

In this section we describe a preliminary set of heuristic rules that the proposed system will use to guide the search through the model space. To determine whether the condition part of a rule matches the current state of the model memory the system must compute various measures of the models in the *current set*. Which models are in this set is determined by previous invocations of the action parts of rules in the rule base. Before describing the set of rules we first describe the measures used by the system to determine the matching rule.

- **Information Theory Measure:** The gain ratio information metric measures the gain in information if test T_i is used to partition the instances into v_i subsets, where v_i is the number of observed values for test T_i (Quinlan, 1986a). We choose this particular metric because Mingers has shown that using the gain-ratio metric produces the smallest trees (Mingers, 1989a). This is desirable because a smaller tree will be typically more accurate than a larger tree (Blumer, Ehrenfeucht, Haussler & Warmuth, 1987; Rissanen, 1989), although Mingers' experimental results did not validate this relationship. Traditionally, such measures have been used to evaluate a set of candidate symbolic univariate tests. However we do not need to limit the use of the information theoretic measure to univariate tests; a linear machine has an information score, and details of how this can be measured are given in Appendix A.
- **Codelength Measure:** Our use of this measure 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 data when coded using the hypothesis to predict the data (Rissanen, 1989). The codelength of a classifier (the hypothesis) is the number of bits required to code the classifier plus the number of bits needed to code the error vector resulting from using the classifier to partition the instances. Details of how to compute the codelength for each model are

given in Appendix B.

- **Form Measures:** Pagallo and Matheus both demonstrate the utility of examining the form of a classifier to change the model space (Pagallo, 1990; Matheus, 1990). During the learning process we can examine the partially formed tree to determine whether a set of models used to construct a subtree was appropriate. There are two form measurement algorithms. The first examines a subtree to determine if a set of univariate tests is repeatedly tested in the tree. What action is taken depends on the form of the subtree. The second algorithm examines a subtree to determine whether a set of multivariate tests is repeatedly tested in the tree.
- **Accuracy Measure:** One measure of the appropriateness of a particular model is the accuracy of the model for the training set. For example, if a linear machine classifies the set of instances with 100% accuracy then the space is linearly separable and therefore, a linear machine is an appropriate model.
- **Instance Set Measures:** We use two different measures of the set of training instances: the number of instances and the number of features.

We now describe each proposed rule, using the definitions shown in Table 2, which defines the symbols used to represent the different models, the fitting algorithms, the measurement algorithms and the three procedures for changing which models are currently being considered. The condition part of each rule has two parts. The first part defines the *context* of the rule. A rule's context is the set of model classes to which the rule applies. If the contents of the *current set* match a rule's context, the matching algorithm then tries to match the second part of the rule's condition, using the measurement algorithms. The action part of each rule specifies one or more of the following types of actions: invoking a fitting algorithm, selecting a model, and adding or removing a model to or from the *current set.*

In the following discussion we have grouped the rules by their context to demonstrate how the search proceeds from each state of the model memory. For each context we first describe the applicable rules in pseudo code, using the definitions in Table 2. We then explain how each rule works and our motivation for including the rule in the rule base. Following this discussion we illustrate how the rules take the model memory from one state to another by showing what a partial search tree would look like.

currentset $= \emptyset$:

RULE 1: IF *currentset* $= \emptyset$ AND
|instance set| $\geq 2 *$ |feature set|
THEN
add (LM[n] = fit-linear-machine)

RULE 2: IF *currentset* $= \emptyset$ AND
|instance set| $< 2 *$ |feature set|
THEN
add (T = fit-univariate-test)

Table 2. Definitions of symbols used to describe the rules.

- **Models:**

  T: A univariate test.

  B[i]: A Boolean combination test of i features.

  LM[i]: A linear machine test of i features.

  IBC: An instance based classifier test.

- **Fitting Algorithms:**

  fit-univariate-test: returns the feature that has the maximum information score.

  form-Boolean: creates a Boolean conjunction of two input features.

  fit-linear-machine: trains a linear machine using the thermal training procedure.

  greedy-discard: discards a feature from an LM using LMDT's elimination method.

  discard: eliminates a feature from an LM using sequential backward elimination.

  fit-IBC: creates a k-NN classifier.

- **Measurement Algorithms:**

  info-score: calculates the information score of a model.

  codelength: calculates the codelength of models and error vectors.

  repeated-univariate-test: searches a tree for a set of repeated univariate tests.

  repeated-multivariate-test: searches a tree for a set of repeated multivariate tests.

  accuracy: calculates the accuracy of a model for the observed instances.

- **Memory Management Procedures:**

  add: Procedure used to add a candidate model to the *current set.*

  remove: Procedure used to remove a candidate model from the *current set.*

  select: Procedure to select a model, which sets the system variable *model*

Initially the model memory is empty. To begin the search we examine characteristics of the training set. We have evidence that some classifiers perform poorly if the number of training instances used to form them is small. For example, if the number of training instances is less than twice the number of features used to describe the instances (the capacity of a hyperplane), then a linear threshold unit will underfit the instances (Duda & Hart, 1973). In this case, we find the best univariate test as measured by its information content. However, if the number of instances is greater than the capacity of a hyperplane, then we induce a linear machine based on all n input features. Clearly, for some data sets the initial rules will fit the wrong model. However, these rules do not *select* either model; they merely begin the search. Other rules overcome a bad initial selection.

currentset $= \{T\}$:

RULE 3: IF *current set* = { T } AND
| instance set |< 2∗ | feature set | AND
codelength(T) $\geq$ codelength(instances)
THEN
select (IBC = fit-IBC)

RULE 4: IF *current set* = { T } AND
| instance set |< 2∗ | feature set | AND
codelength(T) $\leq$ codelength(instances)
THEN
select (T)

This rule checks to see if the initial choice of a univariate test was appropriate. The information theoretic measure does not provide reliable results if the number of training instances is too small (Quinlan, 1987b; Aha, 1990). In this 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 the number of bits required to represent the univariate test and its corresponding error vector is *more* than the number of bits required to represent the error vector of the instances, then we select an instance based classifier. A univariate test partitions the instances in v groups, where v is the number of distinct values of the test observed in the set of instances. If each partition contains instances from more than one class and each class is represented equally in each partition, then using the test to partition the instances may not yield any compression of the data. If a classifier does not compress the data, then by the MDLP it is preferable to encode the data without the classifier. If on the other hand, the test does produce a compression of the data, then we select this test.

currentset = {LM[i]} :

RULE 5: IF *currentset* = {LM[i]} AND
accuracy(LM[i]) = 100%
THEN
add (LM[i-1] = greedy-discard(LM[i]))

RULE 6: IF *currentset* = {LM[i]} AND
accuracy(LM[i]) < 100%
THEN
add (T = fit-univariate-test)

If a linear machine based on i input features is able to classify all of the training instances correctly, then the space of instances is linearly separable, indicating that a linear machine is the appropriate model. However, because we would like to find the simplest classifier with the highest predictive accuracy we continue searching for a linear machine based on fewer features. Therefore, we add a linear machine based on one fewer variables to the *current set.* If the space is not linearly separable, then we add a univariate test to the *current set.* At this point we do not have enough information to know which is the more appropriate model, and several rules, which we describe later, are aimed at providing this information.

currentset = {LM[i], LM[i-1]}:

RULE 7: IF *currentset* = {LM[i], LM[i-1]} AND
info-score(LM[i-1]) $\geq$ info-score(LM[i])
THEN
add (LM[i-2] = greedy-discard(LM[i-1])) AND
remove (LM[i])

RULE 8: IF *currentset* = {LM[i], LM[i-1]} AND
info-score(LM[i]) > info-score(LM[i-1])
THEN
select (LM[i])

If the current set contains two linear machines and if the information score of the linear machine based on fewer variables is either higher or not statistically significantly lower than the other linear machine, then we want to continue searching for a linear machine based on fewer variables. To this end, we discard a variable from the smaller linear machine and we remove the larger linear machine from the *current set.* If the information score of the linear machine based on more variables is higher, then we select that linear machine and halt the search.

currentset = {LM[i],T}:

RULE 9: IF *currentset* = {LM[i],T} AND
info-score(T) > info-score(LM[i]) AND
codelength(T) < codelength(LM[i]) AND
codelength(T) > Codelength(instances)
THEN
select (T)

If the current set contains both a linear machine and a univariate test, then we select the univariate test if it has a higher information score than the linear machine and if it compresses the data.

RULE 10: IF *currentset* = {LM[i],T} AND
codelength(LM[i]) > codelength(instances) AND
codelength(T) > codelength(instances)
THEN
select (IBC = fit-IBC)

If both a linear machine test and a univariate test do not compress the data, then we fit an instance based classifier to the data. In this case neither test produces a compression of the data, and therefore each would have a low predictive accuracy for previously unseen instances.

RULE 11: IF *currentset* = {LM[i],T} AND
info-score(LM[i]) - info-score(T) < ϵ AND
codelength(LM[i]) > codelength(T)
THEN
add (LM[i-1] = greedy-discard(LM[i])) AND
remove (LM[i])

If there is only a small difference, ϵ, between the information scores of a univariate test and a linear machine then we examine the number of bits required to code each test. If the codelength of the linear machine is greater than the codelength of the univariate test, then we continue to search for a linear machine based on fewer variables. The parameter ϵ will be determined empirically when we implement the system. This rule is motivated by the results that a simple classifier is preferable to a complex one (Blumer, Ehrenfeucht, Haussler & Warmuth, 1987; Rissanen, 1989). Because a multivariate test is more complex than a univariate test and one of our goals is simplicity, we need to evaluate whether the gain in the information score merits the gain in complexity. If the complexity is not justified, then we can continue to discard variables.

RULE 12: IF *currentset* = {LM[i],T} AND
info-score(LM[i]) - info-score(T) > ϵ OR
(info-score(LM[i]) - info-score(T) < ϵ AND
codelength(LM[i]) < codelength(T)
THEN
add (LM[i-1] = discard(LM[i]))

If the information score of a linear machine is greater than the information score of a univariate test and its codelength is smaller, then we then we would want to direct the search to the space of linear machines based on more of the variables rather than fewer. We would desire a less greedy search procedure than that used by LMDT (Brodley & Utgoff, 1992). We will employ a search procedure similar to that used by PT2 and CART (Utgoff & Brodley, 1990; Breiman, Friedman, Olshen & Stone, 1984), which perform sequential backward elimination search procedures. The discard algorithm compares n linear machines, each with a different variable eliminated, and picks the linear machine that either realizes the largest increase in the information score or does not realize a statistically significant decrease.

RULE 13: IF *currentset* = {LM[1],T} AND
info-score(T) $\geq$ info-score(LM[1])
THEN
select (T)

RULE 14: IF *currentset* = {LM[1],T} AND
info-score(T) < info-score(LM[1])
THEN
select (LM[1])

If the we have eliminated all but one variable from a linear machine, then we select the test in the *current set* that has the highest information score.

currentset = **{LM[i],LM[i-1],T}:**

RULE 15: IF *currentset* = {LM[i],LM[i-1],T} AND
info-score(LM[i]) > info-score(LM[i-1]) AND
info-score(LM[i]) > info-score(T) AND
codelength(LM[i]) $-$ codelength(T) $\leq \epsilon$
THEN
select (LM[i])

RULE 16: IF *currentset* = {LM[i],LM[i-1],T} AND
info-score(LM[i-1]) > info-score(T) AND
info-score(LM[i-1]) > info-score(LM[i])
THEN
add (LM[i-2] = discard(LM[i-1]) AND
remove (LM[i])

If the information score of a linear machine is greater than the information scores of both a linear machine based on fewer variables and the best univariate test, then we select the linear machine if the greater complexity of the linear machine over the univariate test is justified because of the linear machine's higher information score. This tradeoff will be determined empirically during the implementation of the system. If the *current set* contains two linear machines and a univariate test, then we will never select the univariate test. It is retained to measure the tradeoff between a high information score and a high complexity. If the linear machine based on fewer variables has a higher information score than both the other linear machine and the univariate test, then we continue searching for a linear machine based on fewer variables.

Global context:

RULE 17: IF *model* = T AND
the parent of this node is a univariate test
THEN
B[i] = fit-boolean(T,parent(T)) AND
IF info-score(B[i]) > info-score(parent(T))
THEN
backtrack AND
select (B[i])

Pagallo and Matheus both demonstrate the utility of forming new features that are Boolean combinations of the initial features (Pagallo, 1990; Matheus, 1990). The method that we will use to form Boolean combinations of the features is more general than that used by CITRE. Rather than wait until the entire tree has been formed, we can, in addition, form conjunctions of the initial features at intermediate stages. During the learning process we can examine the partially formed tree to construct new features. For each pair of features on the path to a (temporary) positive leaf, we form a conjunction. We can begin this process as soon as the tree has at least two nodes on the path to a positive leaf. (Note that the space of instances at the leaf may contain negative instances because the tree is only partially formed.) We can then calculate the information score of the new conjunctive feature. If this score is higher than the score of the attribute tested at the parent node, then we replace that attribute with the new feature. For the concept $ab \vee cde$ the tree produced by such a procedure is identical to the tree produced by FRINGE. We conjecture that at least for $\mu - concepts$ (concepts where each literal is contained in at most one term) such a procedure

will produce the same results as FRINGE or CITRE.

RULE 18: IF repeated-univariate-test AND
NOT relationship-is-periodic
THEN
backup AND
select (MAX of accuracy(subtree), accuracy(LM(tests)))

We examine the tree to see if a set of univariate numeric attributes are repeatedly tested in the tree, which gives evidence that *range* tests are being used to approximate a split that is not easily represented by a series of orthogonal splits. A range test is a Boolean test of the form $x > a$, where x is a feature and a is a value within the observed range of x. Consider the concept $ax + by > c$. A decision tree in which each test is a univariate symbolic test will repeatedly test both x and y, within the same subtree. This suggests that there is a relationship between x and y, and further that using a multivariate linear test will produce a better generalization. If we have evidence as to which variables together will form a good partition of the instances, then we do not need to search the entire space of multivariate tests. To this end, we fit a linear machine based on the set of repeated tests to the instances observed at the root of the subtree in which tests are repeated. We only replace the subtree with the linear machine if its accuracy is higher than the subtree's accuracy.

RULE 19: IF repeated-univariate-test AND
relationship-is-periodic
THEN
backup AND
select (MAX of accuracy(subtree), accuracy(Periodic-fn))

Repeated testing of single variables in the tree does not necessarily indicate a linear relationship among the variables. Consider the concept $y = sin(x)$. In a decision tree with univariate tests, both x and y would be repeatedly tested, but the ranges for x would vary within the set of Reals and the range for y would vary within the fixed range [-a,a]. If we were to observe such periodic behavior, then we could search for a periodic function of x and y, rather than linear function. We conjecture that other such patterns can be used to propose the appropriate search bias for tests in a decision tree.

RULE 20: IF repeated-multivariate-tests
THEN
backup AND
select (IBC = fit-IBC(tests))

If a set of variables is repeatedly tested in the tree, then we have evidence that a set of multivariate tests is being used to approximate a split that is not easily represented by a

series of hyperplanes. This suggests that there is a non-linear relationship in the data. In this case an IBC would be a more appropriate model. Moreover, if this set of variables is a subset of the set of input variables, then we have evidence that an IBC based on only these variables would be an accurate classifier. We test this hypothesis by comparing the accuracy of such a classifier to the accuracy of the series of multivariate splits, which it would replace.

Illustration of the Heuristic Search Tree

To illustrate how the rules together form a best-first search through the model space we show a partial search tree in Figure 4. The nodes in the tree show the state of the model memory; selected models are shown in bold-face type and the *current set* is shown in set notation. The arcs of the tree are labeled with the number of the rule that takes the model memory from one state to another. For example, if the *current set* is empty, then either rule 1 or rule 2 will match. If rule 1 matches, then its action part is invoked, which adds a univariate test, T, to the *current set.* If rule 2 matches, then the system adds a linear machine based on all n input features, LM[n], to the *current set.* Each dotted arc represents a series of rule applications. For example, on the left-most path in the tree the dotted arc, which takes the *current set* from {LM[n-1],LM[n-2]} to {LM[i],LM[i-1]}, represents the repeated application of rule 7. In an actual run of the proposed system only one path in this search tree would ever be expanded.

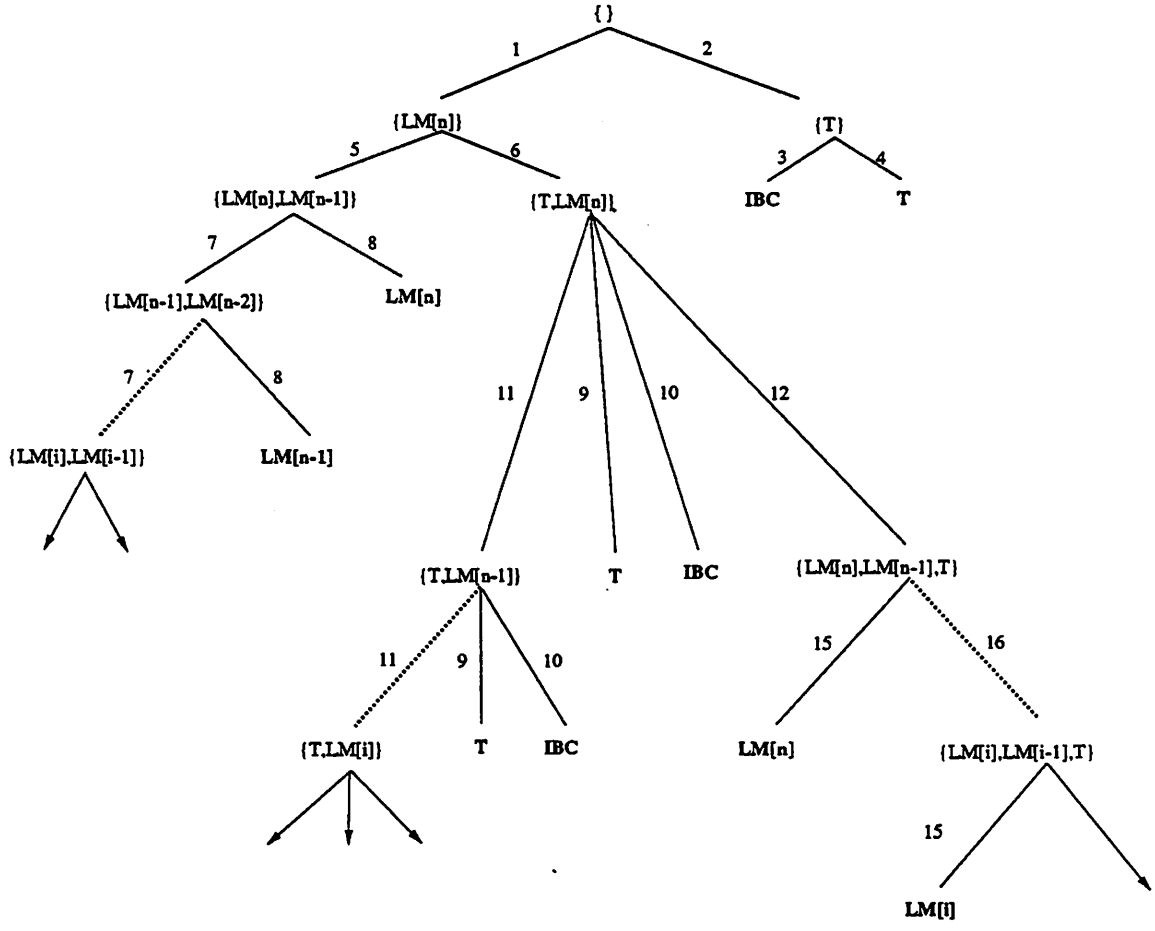

Figure 4. A Partial Search Tree

4 Proposed Work

The primary goal of this research is to show that inductive learning algorithms can adjust their search bias effectively, during learning, by using a dynamic control strategy to select the appropriate model. A bias, in this sense, is a restriction on how to search the model space. We propose to build such a learning system to demonstrate this. In Section 4.1 we describe our strategy for implementing the rule-based system we proposed in Section 3.2; Section 4.2 addresses the issue of overfitting in the proposed model space; Section 4.3 describes how we will evaluate the implemented system to determine its effectiveness; and finally in Section 4.4 we discuss the implications of obtaining this goal.

4.1 Implementation

The implementation of the proposed system will proceed in two stages. During the first stage we evaluate the efficacy of each of the proposed heuristic rules in isolation. In the second stage we will use the results of the first stage to combine the rules, which together will form an efficient control strategy.

- **Stage 1:** We will implement the fitting algorithms, the general architecture of the system and each of the proposed heuristic rules described in Section 3.2. To ensure that the heuristics are effective we will first evaluate each individually, using tasks for which we know which is the best model. If the heuristics choose the best model for these tasks then we will have preliminary evidence that they find the appropriate search bias.

- **Stage 2:** We will use the results of the analysis in Stage 1 to combine the individual heuristics to form an effective control strategy. We will evaluate the control strategy on the same set of artificial examples to check for interaction effects, ensuring that each heuristic still works when combined with the others. This stage may call for refining the proposed set of rules and adding additional rules.

4.2 Overfitting

In addition to implementing an effective control strategy for searching the model space, we must address the problem of *overfitting*. When learning from examples one wishes to avoid overfitting the classifier to the training examples. This problem surfaces in domains that contain *noisy* instances, i.e., instances for which either the class label is incorrect, some number of the attribute values are incorrect or a combination of both. Noise can be caused by factors such as faulty measurements, ill-defined thresholds and subjective interpretation, among others (Quinlan, 1986b). Overfitting occurs when the training data contain noisy instances and the learning algorithm induces a classifier that classifies all instances in the training set correctly. Such a classifier will perform poorly for previously unseen instances. A common approach to correcting for overfitting in a decision tree model is to prune back the tree to the appropriate level. A full tree is grown, which classifies all the training instances correctly and then nodes at the leaves of the tree are pruned back to reduce future classification errors (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1987b). A tree that overfits the training instances is overspecialized and pruning back the tree generalizes the classifier. Quinlan (1987b) and Mingers (1989b) provide comparative studies of commonly used methods.

We cannot apply these correction methods directly to our system. Consider a multivariate test at a node with two children, both of which are leaves. It may be the case that although the node overfits the training instances, pruning the entire node may result in even more classification errors. The issue here is the *granularity* of the nodes in the MCDT classifier: the granularity of a test at a node can vary from a univariate test at one end of the spectrum to a multivariate test based on all n of the initial features at the other end. Rather than prune the tree by eliminating the entire node, a less drastic measure is to attempt first to prune back the test at that node by further variable elimination. Eliminating variables from a multivariate test is another method for generalization; a multivariate test based on $n-1$ features is more general than one based on n features. Therefore to prune back an MCDT classifier we use two forms of pruning. If a node is a univariate test, then we determine if the node should be pruned. If the node is a multivariate test, then we eliminate variables. If only one feature remains after elimination, then we treat it as a univariate test, and decide if the node should be removed.

Alternatively, we could also use an instance based classifier instead of a multivariate or univariate test at the node. A common situation in which overfitting occurs is when there are too few training instances to make a good choice for which variable(s) to use as a test at a node. If the space is impoverished at a node, then an instance based classifier is a good model (Aha, 1990). We can choose between pruning back a node to a leaf (and therefore a single class) or we can retain the training instances and use them to form an IBC. Which choice the system makes will depend on the particular learning task.

4.3 Evaluation

To demonstrate that the implemented system adjusts the search bias effectively, we will evaluate the generalizations induced by the system on a variety of domains. A hybrid formalism is only useful if it subsumes the individual models from which it is constructed. Therefore we will compare our system to a univariate decision tree algorithm, such as C4.5 (Quinlan, 1987a), a k-nearest neighbor algorithm and a linear machine. To illustrate that a control strategy that changes the learning bias is more effective than a control strategy with a fixed bias, both in time spent learning and ability to find a good generalization, we will compare our system to perceptron trees, LMDT and FRINGE. In addition, we will compare our system to an adaptive neural net algorithm, such as DNC (Ash, 1989). The experiments will compare the quality of the generalization induced using our hybrid formalism to the generalizations induced by each of the other algorithms. The algorithms will be compared across the dimensions of accuracy for previously unseen examples, understandability, size of the generalization, time spent learning and time needed to classify instances. To evaluate the system we will use both benchmark machine learning domains and a set of classification tasks from the domain of computer vision. The tasks in the vision domain will consist of region and pixel segmentation problems.

The expected result of our experiments is: the proposed system is capable of determining the appropriate model for any concept, which can be represented by a model in the proposed model space. Clearly if a given data set is best represented by a model not in this space, then the system cannot find it. For some domains the best model may be from one of the individual model classes and we expect that the proposed system will be able to recognize this. In these situations, we expect that the system will produce the same generalization as the system that induces a generalization using the individual model class. Excluding these

situations, we expect that the proposed system will induce generalizations that are more concise and have a higher degree of predictive accuracy than each of the other algorithms listed above.

In addition to this empirical evaluation we will also evaluate the proposed system analytically. The evaluation will be aimed at understanding how the combination of the three model classes increases the set of tasks for which good generalizations can be learned. This will involve an analysis of the strengths and weaknesses of each model class. Building on this analysis, we will then analyze the representational strengths and weaknesses of the hybrid formalism. The motivation for this analysis is two-fold. Firstly, we will demonstrate how combining the models enlarges the set of tasks for which good generalizations can be found. Secondly, we will uncover the characteristics of tasks for which the hybrid formalism cannot find the appropriate representational bias.

4.4 Conclusion

A central objective of machine learning research is to remove humans from the learning process of a machine. One of the primary bottlenecks in building any artificially intelligent system is the knowledge acquisition process. Enabling machines to acquire knowledge without human intervention would do much to alleviate this bottleneck. We have many successful systems that learn from examples, but a human must choose which algorithm to use for a given learning task. The goal of this research is to demonstrate that we can automate this process effectively and efficiently. Obtaining this goal will further automate the ability of machines to learn independently.

There are two important aspects of the proposed system that will enable it to perform automatic model selection effectively. Firstly, using a hybrid formalism enables the learning algorithm to choose the appropriate model class for each subspace or subconcept of the given data set. The hybrid formalism enables the system to take a divide-and-conquer approach to learning the concept. A partially learned classifier, in this formalism, partitions the instance space into a set regions which we call subspaces. This permits the system to select different model classes for different subspaces of the data set, thereby increasing the likelihood of finding a good generalization. Secondly, the proposed system alters the search bias in response to information gained during learning. By employing a dynamic search strategy the space of models can be searched effectively.

The successful demonstration of the proposed system will affect the learning community in several ways. Firstly, it will show that by permitting automatic model selection, a machine can learn effectively a larger class of problems than by using only one model class. A human being is not required to select the appropriate model class. This is desirable because the human being could be wrong, and more importantly, because the goal of machine learning is to automate the entire learning process, and this includes model selection.

Secondly, it will show that initial automatic model selection is not sufficient; a learning algorithm should be capable of selecting different model classes for different subspaces of the data set. If the hybrid model enables one to induce a better generalization than each of the individual model classes for a given classification task, then we will have shown that different model classes are appropriate for different subspaces of a learning task.

Thirdly, the ability to search dynamically for the appropriate model permits an algorithm to focus search in the correct part of the hypothesis space. Using the proposed heuristics to combine the three model classes is more time efficient than trying each class at every stage.

Fourthly, it is a well know problem that the ability of the learning algorithm to induce a good generalization depends on the quality of the input representation. Automatic model selection permits the machine to find the best representation in the model space for the data. If we can create efficient methods for selecting the appropriate model class automatically, then we may loosen our requirements of a set of features to be: the set of features must discriminate the examples. The features do not need to be highly engineered, thereby freeing a human from spending days or even months searching for the appropriate input representation. This is particularly important because selecting a learning algorithm defines the context in which to measure the degree to which the input representation is appropriate.

Finally, our approach is not restricted to the proposed system's model space. Addition of another model class to the system's model space requires only that one add its corresponding fitting algorithm and a set of heuristics for determining its appropriateness. Therefore, the approach offers a general solution to the problem of automatic model selection.

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NCC 2-658, and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764.

I wish to thank my advisor, Paul Utgoff, for his comments on the contents and organization of this proposal. Many of the ideas in this proposal grew out of our joint work on the LMDT algorithm.

References

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

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie (Ed.), *Machine Intelligence.* Edinburgh University Press.

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

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). *Learnability and the Vapnik-Chervonenkis dimension*, (UCSC-CRL-87-20), Santa Cruz, CA: University of California.

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

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.

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.

Dietterich, T. G., London, B., Clarkson, K., & Dromey, G. (1982). Learning and inductive inference. In Cohen & Feigenbaum (Eds.), *The Handbook of Artificial Intelligence: Volume III.* San Mateo, CA: Morgan Kaufmann.

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

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.

Fisher, D. H., & McKusick, K.B. (1989). An empirical comparison of ID3 and back-propagation. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 788-793). Detroit, Michigan: Morgan Kaufmann.

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.

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.*

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

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

Langley, P., Bradshaw, G., & Simon, H. (1983). Rediscovering Chemistry with the BACON system. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

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.

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.

Mingers, J. (1989a). An empirical comparison of selection measures for decision tree induction. *Machine Learning, 3*, 319-342.

Mingers, J. (1989b). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4*, 227-243.

Minsky, M., & Papert, S. (1972). *Perceptrons: An introduction to computational geometry (expanded edition).* Cambridge, MA: MIT Press.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 775-780). Detroit, Michigan: 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.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

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

Quinlan, J. R. (1986b). The effect of noise on concept learning. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1987a). Decision trees as probabilistic classifiers. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 31-37). Irvine, CA: Morgan Kaufmann.

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

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.

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.

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

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., & Shalik, 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.

Van Nostrand (1989). *Van nostrand's scientific encyclopedia.* Van Nostrand Reinhold.

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.

Appendix A: The Information Score for a Linear Machine

In this appendix we explain how the gain-ratio information theoretic measure can be calculated for a linear machine. 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 (1986a).

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

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

To calculate the information gain for a linear machine we use the formula for the multiple class case. The number of values (branches) for a linear machine test is equal to the number of observed classes, k. For a linear machine the gain-ratio for linear machine 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 B: 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 of the partitions 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.

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. Note that for discrete attributes tested in the subtree rooted at this node, we need only use $log_2(n-1)$ bits because a discrete attribute is never re-tested in the subtree. 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.

Coding a Linear Machine:

We need k bits to specify which classes have linear discriminants 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.

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.