

# Recursive Automatic Bias Selection for Classifier Construction

CARLA E. BRODLEY

brodley@ecn.purdue.edu

*School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907*

**Editor:** M. des Jardins and D. Gordon

**Abstract.** The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a selective superiority; each is best for some but not all tasks. Given a data set, it is often not clear beforehand which algorithm will yield the best performance. In this article we present an approach that uses characteristics of the given data set, in the form of feedback from the learning process, to guide a search for a tree-structured hybrid classifier. Heuristic knowledge about the characteristics that indicate one bias is better than another is encoded in the rule base of the Model Class Selection (MCS) system. The approach does not assume that the entire instance space is best learned using a single representation language; for some data sets, choosing to form a hybrid classifier is a better bias, and MCS has the ability to determine these cases. The results of an empirical evaluation illustrate that MCS achieves classification accuracies equal to or higher than the best of its primitive learning components for each data set, demonstrating that the heuristic rules effectively select an appropriate learning bias.

**Keywords:** Inductive bias, hybrid classifiers, automatic algorithm selection, decision trees, learning from examples.

## 1. The problem of bias in classifier construction

For any given dataset, examining the entire space of possible hypotheses is computationally intractable. Therefore to find an accurate hypothesis in a reasonable amount of time, learning algorithms employ a restricted hypothesis space bias and a preference ordering bias for hypotheses in that space (Dietterich, 1990). Empirical comparisons among algorithms illustrate that no single bias exists that is best for all learning tasks (Weiss & Kapouleas, 1989; Aha, Kibler & Albert, 1991; Shavlik, Mooney & Towell, 1991; Salzberg, 1991). The manifestation of this problem is the selective superiority that we observe for each learning algorithm; each algorithm is best for some, but not all tasks (Brodley, 1993). The problem of selecting an appropriate learning bias is complicated further because for some learning tasks different subtasks are learned best using different algorithms. In such cases, the ability to form a hybrid classifier that combines different concept representation languages will produce a more accurate classifier than employing a single representation language and search bias.

This article describes an approach to automatic selection of an appropriate bias for a given dataset. The approach uses a set of heuristic rules to perform a hill-climbing search for the best hypothesis space (model class) and search bias for a given dataset. The rules measure characteristics of the dataset using feedback from the learning process. In addition, the approach has the ability to select different biases for different subspaces of the learning task, thereby forming a hybrid classifier of the data.

Before describing an implementation of the approach, we discuss how feedback from the learning process can guide an automated search for the best learning bias, and we outline situations in which a hybrid bias leads to the best classifier of the data. We then describe an implementation of the approach, the Model Class Selection (MCS) system, which forms a tree-structured hybrid classifier that mixes three primitive representations: linear discriminant functions, decision trees and instance-based classifiers. Our choice of these three representation languages stems from the results of empirical comparisons, which illustrate that these three languages produce classifiers of differing accuracy for different datasets; each of the three was selectively superior for some datasets. The classifier formed by MCS can be either a hybrid or a homogeneous classifier, depending on what the characteristics computed during search indicate is the best bias. Specifically, in Section 2 we describe the representations and search strategies for each of MCS's primitive learning algorithms and discuss the differences among them. We give a detailed description of the rule set which describes the characteristics that each rule tests and explains why the presence of a particular characteristic leads MCS to prefer one bias over another.

In Section 3 we present the results of empirical experiments that illustrate that MCS achieves classification accuracies equal to or higher than the best of its primitive learning components for each of a variety of datasets. Our results demonstrate that the heuristic rules effectively select an appropriate learning bias, thereby solving the selective superiority problem for these algorithms. Moreover, the results of these comparisons illustrate that for some datasets a hybrid bias is better than a homogeneous bias, and that MCS's search strategy can find these cases.

In addition, we illustrate empirically that using knowledge increases accuracy and reduces time over methods that form hybrid classifiers by trying all of the available algorithms and selecting among them using a single source of feedback. Specifically, we compare MCS to three other hybrid classifier construction methods; each of the three methods applies a different criterion to select a best classifier at a node. Indeed, the results demonstrate that these "knowledge-poor" methods perform worse for some datasets than the best of the primitive algorithms. In contrast, our knowledge-based approach performs equal to or better than each of the primitive algorithms, and equal to or better than each of the other three hybrid methods.

### *1.1. Automatic algorithm selection*

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

There are many measurable characteristics of datasets. For example, one can characterize datasets by whether the instances are described by numeric or symbolic features, or by a measure of correlation among the features and the class labels. The challenge is in uncovering characteristics that will indicate the best bias, and further, whether such characteristics can be computed. Rendell and Cho (1990) point out that “data character” plays an extensive role in determining the behavior of learning algorithms. Their view of concepts as functions over the instance space led them to define geometric characteristics such as concept size (proportion of positive instances) and concentration (a characterization of the distribution of positive instances through the instance space), which they subsequently illustrate have a large affect on learning.

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

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

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

Indeed, a recent focus of research in machine learning is to understand the tasks for which a particular algorithm will perform better than some specified set of alternatives (Feng, Sutherland, King, Muggleton & Henry, 1993; Aha, 1992; Shavlik, Mooney & Towell, 1991). Systems that allow the user to specify an inductive policy require that the biases of the available learning algorithms be known and be represented explicitly for manipulation during search (Provost & Buchanan, 1992).

In the STATLOG project, sixteen algorithms were compared across twelve datasets (Feng, et al. 1993). One of the goals of the project was to discern what characteristics of datasets suit particular algorithms. Twelve statistical characteristics were uncovered that they think will be useful for predicting which of the algorithms will perform the best for a given task. However, the heuristics about these characteristics that were put forth have

not at this point been evaluated on any datasets not used in the original comparative study.

Shavlik, Mooney and Towell (1991) compared the ID3, Backprop and Perceptron learning algorithms. They analyzed empirically the effect that the amount of training data, imperfect training examples, distributed output encodings and the type of features used to describe the data (numeric or symbolic) have on the performance of each algorithm. Like the STATLOG project, the resulting heuristics put forth, e.g. Backprop works better than ID3 when the data is described by numeric features, were not evaluated on any new datasets.

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

Prior to the recent focus on knowledge-based approaches to automatic algorithm selection, the strategy was to try all candidate methods and choose one based on estimates of their accuracies. One well-known approach from statistics is to use cross-validation (Linhart & Zucchini, 1986). Recently, Schaffer (1993) applied this idea to selecting a classification algorithm. The results of an empirical comparison of a cross-validation method (CV) to each algorithm considered by CV, illustrated that on average, across the test-suite of domains, CV performed best. Of course these methods are also using a characteristic of the dataset to make a selection; the accuracy of a particular algorithm is a descriptive characteristic of a dataset. An important difference between the traditional approach and the approach presented here is that the traditional approach applies all candidate algorithms to the data before making a selection, whereas our approach may choose an algorithm without trying all of them.

In this article, we describe a set of heuristic rules used to guide an automatic algorithm selection search. We compare our approach to a variety of different search methods on datasets not used to develop the rule base. The results illustrate that general domain-independent knowledge does exist and that it is useful for guiding search.

### ***1.2. Homogeneous versus hybrid classifiers***

One can characterize algorithms as searching either a *homogeneous* or *hybrid* hypothesis space. A homogeneous hypothesis space is one that contains a single representation language. For example, univariate decision trees and linear discriminant functions are

each single representation languages. A hybrid hypothesis space is a space that combines different homogeneous hypothesis spaces; each hypothesis is expressed using one or more representation languages. For example, a representation language that mixes linear discriminant functions and univariate decision trees in a single hypothesis defines a hybrid hypothesis space.

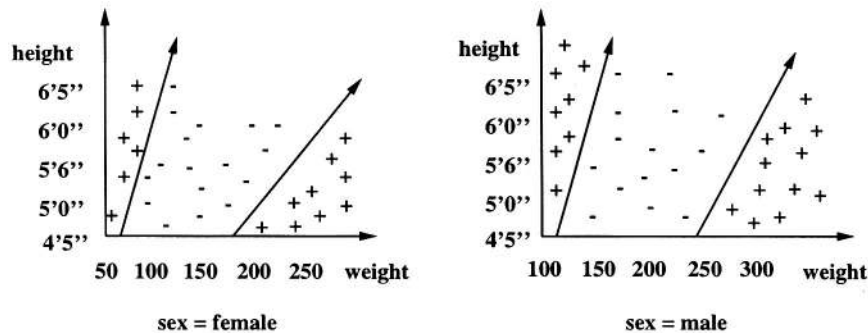


Figure 1. The concept heart attack; "+": positive instance, "-": negative instance.

Selecting a learning algorithm that employs a homogeneous representation bias assumes that a classifier for the dataset is best represented using a single representation language. For some datasets this may not be the case; different subspaces of the learning task may be learned best using different representation biases, indicating that forming a hybrid classifier will result in higher classification accuracy than a homogeneous classifier. For example, consider the following concept: heart attack caused by a weight problem. A heart attack can be caused by two extremes, obesity or anorexia.<sup>1</sup> Given a set of positive and negative training examples described by three attributes, sex, height and weight, a concept learner needs to learn the subconcepts "underweight" and "overweight". These two subconcepts are different depending on whether the patient is a woman or a man. A single linear threshold unit (LTU) is not appropriate for the entire instance space because the space is not linearly separable, as can be seen in Figure 1. Moreover, a symbolic decision tree algorithm would have difficulty learning a compact generalization for this concept because it would need to approximate the hyperplanes with a series of splits orthogonal to each of the axes.

A better solution (shown in Figure 2) is a hybrid formalism that combines LTUs and decision trees. This example illustrates that choosing initially between a decision tree and an LTU will not yield as succinct and accurate a classifier as combining both representation languages and employing a control strategy to choose between them, for each subspace of the instance space. For such datasets one would like to mix different representations and search biases to create a hybrid classifier.

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

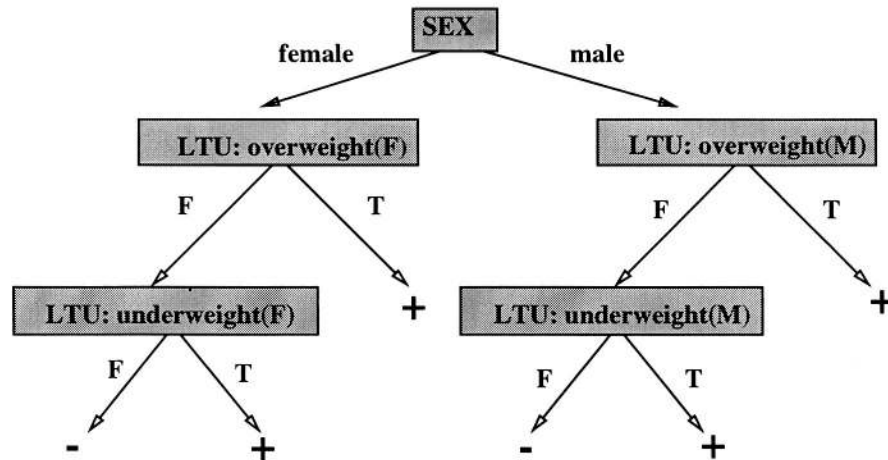


Figure 2. Classifier induced for the concept heart attack using a hybrid formalism.

The hypothesis space of hybrid classifiers is larger than the hypothesis space of the primitive components. By increasing the space of possible hypotheses we increase the probability that for a given dataset, a good generalization will exist in the space searched. Increasing the search space does not ensure that a good generalization will be found; it depends on the search algorithm. Therefore, a second hypothesis of this research is: *Our knowledge-based search strategy for finding a hybrid classifier will produce a classifier that is never worse than, and for some datasets is better than, any homogeneous classifier produced by its primitive components.* Our empirical results (reported in Section 3) illustrate that merely increasing the search space by permitting hybrid classifiers will not lead to an increase in performance; one needs a search strategy that will not be misled by the larger number of possibilities.

There are two basic approaches to constructing a hybrid classifier. The first is to apply each of several different learning algorithms to the entire set of data and then to combine their outputs (Wolpert, 1992; Breiman, 1992; Zhang, Mesirov & Waltz, 1992; LeBlanc & Tibshirani, 1993). The combination scheme can be simply a weighted average, or alternatively a learning algorithm can be applied to determine how the outputs of the primitive classifiers should be combined. Such approaches have been called *Stacked Generalization* (Wolpert, 1992; Breiman, 1992). These approaches can result in a hybrid classifier that is less accurate than one of its primitive components if a bad combination scheme is used (Wolpert, 1992; Breiman, 1992). Indeed, it is an open problem how to form a procedure for choosing or combining the outputs of the primitive learning components.

A second approach, the one presented in this article, is to assign explicitly a different classifier for each mutually exclusive subset of the data (Tcheng, Lambert, C-Y Lu & Rendell, 1989; Utgoff, 1989). Given an instance, a decision procedure is used to decide

to which subspace it belongs and the classifier for that subspace is used to classify the instance. This approach constructs a tree-structured hybrid classifier; the internal test node(s) partition the instance space into a set of regions, each of which has a separate classifier.

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

In the AIMS system (Yerramareddy, Tcheng, Lu & Assanis, 1992), the model formation component, CRL (Tcheng, Lambert, C-Y Lu & Rendell, 1989), forms a recursive hybrid structure. CRL partitions the instance space recursively by selecting among the user-specified decomposition strategies (univariate tests and arbitrary hyperplanes). After CRL determines that further decomposition (partitioning) is not desirable, it searches for the best of a user-specified subset of a univariate test, a neural network, a  $k$ -nearest-neighbor classifier or a regression model (linear, quadratic, logarithmic or exponential). CRL restricts the types of tests permitted at internal nodes of the tree to decomposition strategies. In contrast, our approach can construct a test for any node in the tree using any of the available representation languages and search biases. Another difference between CRL and our approach is the search strategy used to select a test for a node. CRL tries all the user-specified methods. If CRL is coupled with the ISO system, an optimization search is performed.

## 2. The MCS system

We have implemented our recursive automatic algorithm selection approach, producing the Model Class Selection (MCS) system. Given a set of data, MCS builds a classifier using a set of heuristic rules to guide a hill-climbing search for a best representation language and search bias from which to form a test for each node in a hybrid classifier. Figure 3 shows an example of a hybrid classifier that MCS might construct. The root of the tree is a linear combination test, the left subtree is an instance-based classifier and the right subtree is a two-node univariate decision tree. Each leaf node is labeled with one of three classes (A, B, or C). In the next two sections we describe MCS's model classes, search strategy, and the dataset characteristics, computed during search, that lead MCS to prefer one bias over another.

### 2.1. Model classes

MCS combines three primitive representation languages that have been used extensively in both machine learning and statistics algorithms: linear discriminant functions, decision trees and instance-based classifiers. Classifiers constructed from any one of these three

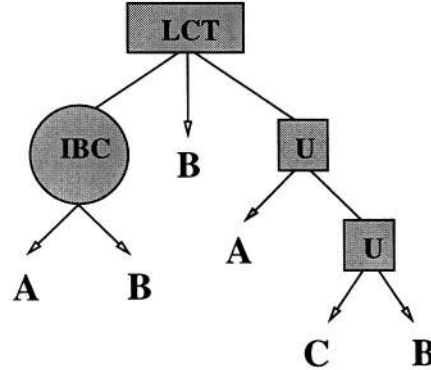


Figure 3. Example of a hybrid tree-structured classifier.

languages each form a piecewise-linear partition of the given instance space. They differ in how (where) the boundaries may be placed in the space.

A univariate test of feature  $F_i$  represents a decision boundary that is orthogonal to  $F_i$ 's axis. A univariate decision tree defines a set of orthogonal decision boundaries that partition the instance space into a set of hyper-rectangular regions each labeled with a class name. A set of  $R$  linear discriminant functions defines a set of  $R$  regions in the instance space, separated by hyperplanes, each labeled with a different class name. An instance-based classifier defines a piecewise-linear partition of the instance space; the number of blocks is determined by the number and distribution of the instances, and by the choice of  $k$ , which is the number of nearest neighbors to examine when classifying an unlabeled instance. The result is a set of regions, each labeled by a different class name, separated by piecewise-linear boundaries.

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

In Figure 4 we show an instance space for which classifiers from each of the three model classes would define an identical partition of the instance space, given the goal of partitioning the instances into regions, each containing instances from a single class. For many datasets, classifiers from each of the three model classes will define different partitions. Figures 5a, 5b, and 5c illustrate the type of partition each model class might define for a simple instance space consisting of five negative and three positive examples of the concept to be learned. Which of these concept representations is best depends on where in the instance space the true concept boundary lies. MCS's rule-based search

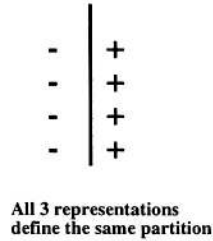


Figure 4. Representation similarities

strategy addresses this problem by using feedback from the learning process to determine which of the three representation biases is best for the given instance space.

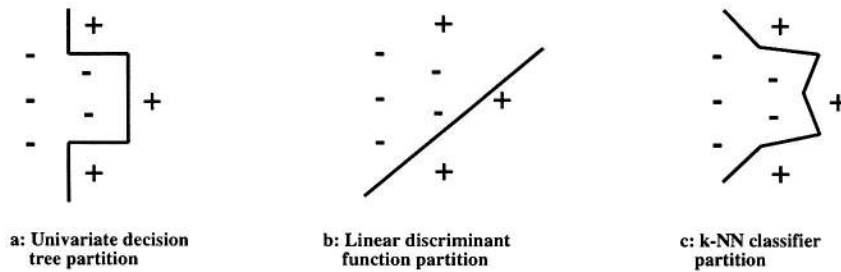


Figure 5. Representation differences

For each model class there are many different biases for searching for the classifier that best fits the data. We have chosen a commonly used method for each. The search biases for each model class that MCS uses are:

**Univariate Decision Trees:** To build a univariate decision tree, MCS uses the Information-Gain Ratio metric (Quinlan, 1986) to select each test node of the tree.

**Linear Discriminant Functions:** For two-class tasks the system uses a linear threshold unit, and for multiclass tasks it uses a linear machine (Nilsson, 1965). To find the weights of the threshold unit (linear machine) MCS uses the Thermal Training rule (Frean, 1990; Utgoff & Brodley, 1991). Because the weights found by this rule depend on the order in which the instances are presented, we train ten times and select the set of weights that maximizes the Information-Gain Ratio metric. To select the terms to use with a linear discriminant function, one of three search procedures is applied: Sequential Backward Elimination (SBE) (Kittler, 1986), a variation of SBE, Dispersion Sequential Backward Elimination (DSBE), which uses the form of the function to determine which terms to eliminate (Brodley & Utgoff, 1995), and Sequential Forward Selection (SFS) (Kittler, 1986). The choice of which of these

search biases to apply is determined dynamically during learning, depending on the hypotheses that have already been formed.

**Instance-Based Classifiers:** MCS uses the  $k$ -nearest-neighbor algorithm (Duda & Hart, 1973), which stores each instance of the training data. To find  $k$ , the system estimates the classifier's accuracy with the following measure: for each instance in the training data, classify that instance using the remaining instances. The system selects the value of  $k$  that produces the highest number of correct classifications for the training data.

## 2.2. Search strategy

MCS searches for a hybrid classifier from the available model classes using a hill-climbing search to select a test for each node in the tree-structured classifier. A set of heuristic rules is used to decide which model classes to try, which model classes should be avoided, and to determine when a best test at each node has been found. The instance space is partitioned according to the chosen test, and the search is applied recursively to each resulting subset that contains instances from two or more classes. The general recursive procedure of MCS is shown in Table 1. After MCS has constructed a hybrid classifier that perfectly partitions the training instances into regions each labeled with a single class name, it applies a pruning algorithm to reduce the estimated error of the classifier as computed for an independent set of instances.

One problem that can occur during MCS's search is when two tests (classifiers) appear equally good for the set of instances observed at a node in the tree. The current version of MCS differs from our original version (Brodley, 1993) in order to handle these situations better. If two tests appear equally good at a node, and do not perfectly partition the set of instances at that node, then MCS examines whether one defines a better *partition* of the data than the other. In a decision tree, test nodes have one of two functions, depending on their position in the tree. Test nodes whose children are each leaves serve as classifiers of the subspace defined by the tree above them. Internal test nodes (nodes for which at least one child is not a leaf node) partition the instance space into subspaces. For example, in Figure 3 the linear combination test (LCT) partitions the space into two subspaces and the IBC node serves as a classifier for one of the subspaces. We define the quality of a partition test as the accuracy of the subtree whose root is that test. One *partition* test is judged better than another if its subtree's accuracy is higher than the other test's subtree accuracy.

In cases for which two tests appear equally good, MCS builds a subtree, of depth one, for each of the two candidate tests. MCS builds a subtree by constructing a classifier for each subset of the instances defined by the partition test. Which representation languages and search biases are used to construct classifiers for each subspace depends on the context of the comparison. In our description of MCS's rules we detail how these choices are made. The two subtrees are then compared using the heuristic rules to determine which of the two partition tests to place at that node. In cases for which the

Table 1. General recursive algorithm

---

```

Form Classifier(instances)

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

```

---

two tests each partition the set of instances perfectly, MCS selects the simpler of the two tests.

Even with the addition of the one-ply deeper search, MCS may still have difficulty selecting among tests that appear equally good. In particular, this can happen when the measures of the set of candidate tests yield conflicting results. For example, suppose that the information score of test  $T_1$  is greater than that of test  $T_2$ , but  $T_2$ 's accuracy is higher than  $T_1$ 's. In such cases it is unclear which test to choose. To address these situations we have added a *global* model-class bias to MCS, which is determined automatically before MCS begins to construct a classifier. We describe how this global bias is selected in Section 2.2.1.

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

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

### 2.2.1. Choosing a global model-class bias

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

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

### 2.2.2. *Heuristic rule base*

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

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

Our judgement of whether a test compresses the data is based on the Minimum Description Length Principle, which states that the best "hypothesis" to induce from a dataset is the one that minimizes the length of the hypothesis plus the length of the exceptions (Rissanen, 1989). The *codelength* of a classifier (the hypothesis) is the number of bits required to encode the classifier plus the number of bits needed to encode the error vector resulting from using the classifier to classify the instances. We say that a test *compresses* the data if the number of bits required to represent the test and its corresponding error vector is *less* than the number of bits required to represent the error vector of the instances. In the rules, we examine only univariate and linear combination tests for compression.<sup>2</sup> The details of how to compute the codelength of these two types of tests can be found in Brodley (1994).

Before describing the rules in detail we outline how the rule set was developed. Our method was a cyclical process of trial and error. We began with general ideas about the situations in which one model class would be preferred to another. These ideas were culled from the literature and from our own experience with these three model classes. For example, it is fairly well-known that instance-based classifiers perform well for datasets for which all the features are relevant. However, if many features are irrelevant

Table 2. Rule for when candidate set = {}:

---

IF (number of instances < hyperplane capacity) OR (global-bias = U) THEN **fit**(U)  
 ELSE **fit**(LCT<sub>n</sub>)

---

then they are known to perform poorly (Aha, 1990). Whenever applicable we reference the original source of a rule in the description of the rules presented in Section 2.2.3.

We encoded our general heuristics into a set of rules such that no conflict resolution is required. We then used four datasets, Iris, Breast, Pixel and Heart Disease, which are described in Section 3.1, to debug the rules. (Our empirical experiments use these four and twelve additional datasets.) MCS generates a detailed rule trace, which prints out the decision made at each step in the search at a node, all of the available measures of the set of data observed at the node, and measures of the classifiers that have already been tried. After a run, we examined this detailed rule trace to decide whether MCS had made the correct decisions in the search. In particular, we looked for cases for which MCS produced a classifier that was less accurate than the best of its primitive components. We pinpointed where in the rule trace MCS made a decision that led it away from the best model class. We then altered the rules to ensure that the better decision was made and re-ran MCS on that dataset. Once we had corrected the problem for that dataset, we ran MCS with the new rules across all four development datasets. If performance did not decrease for the other datasets then the rule(s) remained, otherwise we examined why the new rule(s) hindered performance and readjusted the rule set. This cyclical process involved many months of experimentation. In the following description of the rules we describe the general intuition behind each rule and in some cases give illustrative examples.

### 2.2.3. Description of the rules

The description of the rules is organized by the model classes the search strategy has investigated, which we designate as the *candidate set*. In the description of the rules we use four symbols: **fit**(T) adds a new test, T, to the candidate set; **select**(T) terminates search and selects T from the candidate set to use as a test at that node; **delete**(T) removes T from the candidate set, because MCS has determined that it is not as good as the other candidates; and **examine-alt**(T) compares the accuracy of T to the alternative test (if it exists) and the chosen test, and retains the alternative with the highest accuracy. The rules are shown in Tables 2 through 10, and we spend the remainder of this section describing and motivating each rule.

Initially the candidate set is empty and MCS must decide where in the model space to begin the search. To this end, MCS examines the number of instances relative to the number of features that describe each instance. When the ratio of instances to features is small, either a univariate test (U) or an instance-based test (IBC) is preferred over a linear combination test (LCT). The rule shown in Table 2 determines whether to start the search

Table 3. Rules for when candidate set = {U}:

---

```

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

```

---

with a univariate test or an LCT. (The rules shown in Table 3 address the situation where an IBC would be preferred to a univariate test in the case of a small number of instances.) Specifically, if the number of training instances is fewer than twice the number of features used to describe each instance (the capacity of a hyperplane) (Duda & Hart, 1973) then MCS starts with a univariate test; otherwise the search begins with an LCT based on all  $n$  features. This rule is motivated by the observation that when there are too few instances relative to the number of features, then there are many possible orientations of the hyperplane that are consistent with the data, and not enough information to choose among the possibilities. This rule is overridden when the global model-class bias is toward univariate decision trees; in this case the first test formed is always a univariate test. This does not preclude the investigation of an LCT; it may, however, reduce search effort.

The rules shown in Table 3 determine whether the initial choice of a univariate test was appropriate or whether further exploration of other model classes is required. Recall that at this point the candidate set = {U}. If the univariate test perfectly partitions the data, then there is no need to search for a more complex test, and search halts. Otherwise MCS tries to determine if a univariate test representation is inappropriate by checking for the presence of one of two different data characteristics. The first characteristic is whether no single feature is superior; several features partition the data equally well. In this case a linear combination test may provide a better partition of the data. An example of such a situation is shown in Figure 6, which shows a two-dimensional instance space and the corresponding univariate decision tree, which approximates the hyperplane boundary,  $x + y \leq 8$ , with a series of orthogonal splits. In the figure, the dotted line represents the hyperplane boundary and the solid line represents the boundary defined by the univariate decision tree. This example illustrates the well-known problem that a univariate test using feature  $F_i$  can only split a space with a boundary that is orthogonal to  $F_i$ 's axis (Breiman, Friedman, Olshen & Stone, 1984).

In an earlier version of the rule set we explicitly tested for such situations, but we found that by examining the relative difference in the information score of the best feature to the other features we could catch such situations and save a substantial amount of computation time. If the average difference of the best U to each of the others is less than a threshold,  $\epsilon$ , then MCS explores whether a linear combination test will do better. We have set  $\epsilon$  to 0.20. To this end, MCS starts with a best feature (the one

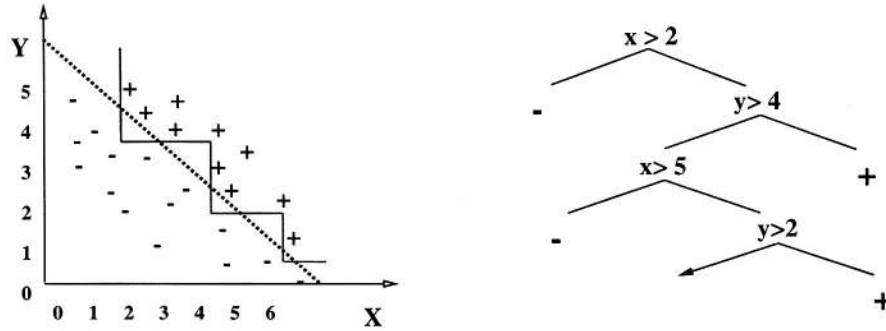


Figure 6. An example instance space; "+": positive instance, "-": negative instance and the corresponding univariate decision tree.

Table 4. Rule for when candidate set =  $\{LCT_n\}$ :

---

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

---

chosen for the univariate test) and performs an SFS search. The second characteristic is whether a univariate test compresses the data. It is known that the information-theoretic measure does not provide reliable results if the number of training instances is too small (Quinlan, 1987; Aha, 1990). In these cases, an instance-based classifier is more likely to be appropriate because there is no minimum number of instances required to form an IBC. Note that if the univariate test was initially chosen based on the global bias (indicated by the number of instances being greater than the hyperplane capacity), then the system fits an LCT based on all  $n$  features and uses the rules shown in Table 6 to decide where next to guide the search. If neither of the above two characteristics is observed then there is no indication that a better model class will be found, and the univariate test is selected.

If the first model class tried (as determined by the rule shown in Table 2) was an LCT based on all  $n$  of the input features, then MCS applies the rule shown in Table 4 to decide whether to continue searching within the class of linear combination tests or whether to explore other model classes. Linear combination tests are ideal for linearly-separable instance spaces. MCS looks for this characteristic by examining the accuracy of the linear combination test that has been fit to the data. If the LCT based on all  $n$  features is 100% accurate then we know that the space is linearly separable, and all that remains to be done is to search for the smallest set of features to include in the test while retaining linear separability. This is achieved by a sequential backward elimination search procedure (DSBE) that eliminates features one by one using the magnitudes of the corresponding weights to determine which features to eliminate (Brodley & Utgoff, 1995).

Table 5. Rules for when candidate set =  $\{LCT_i, LCT_{i-1}\}$ :

---

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

---

Table 6. Rule for when candidate set =  $\{U, LCT_n\}$ :

---

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

---

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

If the accuracy of an LCT based on all  $n$  features indicated that the instance space was linearly separable, then the application of the rule shown in Table 4 created the candidate set =  $\{LCT_i, LCT_{i-1}\}$ , where  $i = n$ . The rules in Table 5 determine whether further feature elimination should take place and if not, what to do next. MCS continues to eliminate features as long as there are more than two features in the smaller of the two linear combination tests and the accuracy of the smaller of the two is 100%. If the system eliminates features until there is only one remaining, then it selects this test. Otherwise it checks to see whether the best  $LCT_i$  does not compress the data, indicating that it may be too complex for the data. In this case, the system fits a univariate test to see if it will compress the data. This rule is motivated by the observation that although an LCT may perfectly partition the *training data*, it may be overfitting to noise in the data. Building a subtree of univariate tests allows MCS to do more fine-grained pruning.

The rule shown in Table 6 handles the point in search where both a best univariate test and an LCT based on all  $n$  features have been added to the candidate set. If both the information score and the accuracy of the univariate test are higher than the LCT, then there is evidence that the best test is univariate. However, MCS explores the option that a small LCT may be a better test, by constructing an LCT using SFS, starting with the feature in the univariate test. This rule is based on results of previous research that illustrate that the bias of an SFS search can lead to a better LCT than an SBF search for some datasets (Brodley & Utgoff, 1995). Otherwise, there is no reason to believe that a better LCT could not be found, and the system searches for one using SBF.

Table 7. Rules for when candidate set =  $\{U, LCT_i\}$ :

---

```

IF ((global-bias = Uni) AND
    (info(U) ≥ info(LCTi) OR accuracy(U) ≥ accuracy(LCTi))) OR
    ((global-bias = lct) AND
    (info(U) ≥ info(LCTi) AND accuracy(U) ≥ accuracy(LCTi))) OR
    ((global-bias = none OR ibc) AND (info(U) ≥ info(LCTi))) THEN
    examine-alt(LCTi)
    IF U compresses the data THEN select(U) AND recurse
    ELSE fit(IBCk=2) AND delete(LCTi)
ELSE examine-alt(U)
    IF LCTi compresses the data THEN select(LCTi) AND recurse
    ELSE fit(IBCk=2) AND delete(U)

```

---

If the candidate set contains a univariate test and an LCT, based on  $i$  features formed using either an SBE or SFS search (candidate set =  $\{U, LCT_i\}$ ), then MCS's next action depends on whether there is a global model-class bias. If the bias is toward univariate decision trees, then the system forms a univariate subtree of depth one. This is because in many cases a single univariate test will not have as high an information score or accuracy as an LCT. Because an LCT is more complex than a U, a fairer comparison is an LCT to a univariate tree of depth one. In addition, MCS biases the information score and accuracies of the two tests by their complexity. Specifically it uses the following weight:  $\frac{T-v}{T+v}$ , where  $T$  is the number of instances and  $v$  is the number of features in the test (Quinlan, 1993). This has the affect of penalizing the more complex test. MCS uses these weighted information scores and accuracies in the rules shown in Table 7. If the bias is not for univariate trees but the two tests are close in either information score or accuracy (within 10% of one another), then the system creates a subtree for each of the LCT and the univariate test, for which each of the children can be classifier from any of the three model classes. Rather than choose erroneously between the LCT and univariate tests, MCS is exploring which of the two tests make the subspaces easier to learn. To construct the subtree, MCS chooses a classifier for each subspace from the set of a best univariate test, an LCT test and a k-NN ( $k=1$ ) test. The chosen test is the one that maximizes the Information-Gain Ratio.

MCS compares the information score and accuracy of the two tests (possibly subtrees at this point) to decide whether to select one of the tests or whether an instance-based classifier should be examined. MCS prefers U to LCT if one of the following three cases is true: the global bias is toward univariate decision trees and either the accuracy or the information score of U is better; the global bias is toward linear combination tests and both the information score and accuracy of U are higher than those for the LCT; and if there is no global bias or the bias is toward instance-based classifiers, then only the information score is considered. If U is preferred, then the system examines whether U compresses the data. If it does then the search halts and U is selected. If U does not compress the training data, then MCS examines whether an instance-based classifier is

Table 8. Rules for when candidate set =  $\{LCT_i, IBC_k\}$ :

---

```

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

```

---

a better model. Note that MCS retains the ability to select the LCT during pruning by examining whether it would make a good alternative. If the LCT is preferred over U, then the system performs the same compression analysis, making a decision of whether to select the LCT or examine an IBC.

If the candidate set =  $\{LCT_i, IBC_{k=2}\}$ , then MCS decides whether further exploration of IBC tests is required using the rules shown in Table 8. If the global bias is not for instance-based classifiers, then the accuracies of the LCT and IBC are weighted by their complexity as described above. If the accuracy of the IBC is higher than the LCT's, then MCS explores the IBC model class, by increasing the value of  $k$  (the number of nearest neighbors to examine during classification). In addition, MCS examines whether the LCT would make a good alternative. If the IBC's accuracy is not higher than the LCT's accuracy, but one of the following three cases is true then MCS explores higher values of  $k$  for the IBC: the global bias of MCS is toward an IBC; more than half the features remain in the LCT; or there is a less than 10% difference in the accuracies of the IBC and the LCT. The second case captures situations in which IBCs do well because all (or most) of the features are relevant. The third case catches situations for which a 1-NN is a poor choice but a  $k$ -NN is a good choice ( $k > 2$ ). MCS investigates an IBC, exploring increasing values of  $k$  until the accuracy of the IBC drops lower than that of a 2-NN (which is equivalent to a 1-NN) classifier. At this point MCS selects between the IBC or the LCT. Finally, if the LCT is the better test, then it selects LCT and retains the IBC as a possible alternative, but does not investigate other values of  $k$ . Note that only accuracy is used here, because in most cases for which an IBC is selected as a test no subtree will be needed.

When search has led MCS to believe that the best candidate test is a univariate test or an IBC, MCS applies the rules shown in Table 9. If the global bias is for univariate tests, then MCS grows a subtree of depth one for the univariate test in which each node of the subtree is a univariate test. MCS then biases the univariate subtree and the instance-based classifier by their complexities if the global model-class bias is for univariate trees. The system selects U if its accuracy is higher than that of the IBC. Otherwise it explores the

Table 9. Rule for when candidate set =  $\{U, IBC_k\}$ :

---

```

IF accuracy(U)  $\geq$  accuracy(IBCk) THEN
    select(U) AND examine-alt(IBCk) AND recurse
ELSIF accuracy(IBCk) < 100% THEN fit(IBCk+1) AND delete(U)
ELSE select(IBCk) AND examine-alt(U) AND recurse
    
```

---

 Table 10. Rule for when candidate set =  $\{IBC_k, IBC_{k+1}\}$ :

---

```

IF accuracy(IBCk+1) > accuracy(IBCk) THEN fit(IBCk+2) AND delete(IBCk)
ELSE select(IBCk) AND recurse
    
```

---

IBC model class. The test not chosen is examined to see whether it would make a good alternative.

To reach the point in the search where the two best candidate tests are each instance-based classifiers, the accuracy of  $IBC_{k=2}$  was higher than either a univariate test, an LCT, or both. The rule shown in Table 10 handles the situation in which MCS has decided that the best model class is instance-based classifiers and is now searching for the value of  $k$  that leads to the best heuristic accuracy on the training data using the leave-one-out scheme described in Section 2.1 above.

#### 2.2.4. Pruning hybrid classifiers

To address the problem of *overfitting* in the hybrid formalism, the system prunes back the classifier to minimize the estimated classification error computed for an independent set of instances (Breiman et al, 1984; Quinlan, 1987). Overfitting occurs when the classifier overfits the training data at the expense of generalization. In the case of domains that contain *noisy instances* (instances for which the class label is incorrect or some of the feature values are incorrect) finding the classifier that maximizes the accuracy for the *training data* may overfit to the noise in the training data, and subsequently perform poorly for previously unseen instances.

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

A hybrid classifier that mixes different model classes has test nodes of varying complexity. This can cause problems when deciding whether to replace a test node with a class label; a complex test may overfit the training data, but removing it may decrease the

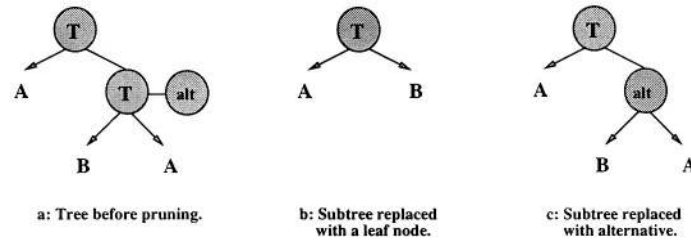


Figure 7. Choices of hybrid classifier pruning method.

accuracy of the classifier. In such cases, one wants to replace the test with one that is less complex. In addition, as described in Section 2.2, test nodes in a decision tree have one of two functions: to partition or classify the instances observed at that node. During the tree construction phase, MCS saves the most accurate candidate test if it was not chosen as the test for that node. Currently, MCS does not bias the storage of alternatives toward less complex tests. An issue for future research is to examine the utility of saving both the most accurate alternative and a less complex test for consideration during pruning.

During pruning, in addition to deciding whether to retain a subtree or replace it with a leaf, our approach examines whether replacing the subtree with the alternative would result in a lower estimated classification error than either of the two other choices. In Figure 7 we show these three options: Figure 7a illustrates the option of retaining the original tree (the alternative would be deleted); Figure 7b illustrates the option of replacing a subtree with a leaf node; and Figure 7c illustrates the option of replacing a subtree with the saved alternative.

### 3. Empirical results of recursive automatic bias selection

Our experiments have two goals. First, we want to determine whether our predictions of a best bias based on dataset characteristics computed during search works well in practice. Second, we want to know whether there exist datasets that require hybrid classifiers. In a previous experiment (Brodley, 1993) we illustrated that MCS is more robust than each of its primitive components, but that it never significantly outperformed the best primitive algorithm for each dataset. We hypothesized that this was due to one of four causes: for the datasets in that analysis, better accuracy could not be achieved due to noise in the data; the rules in MCS needed to be improved; those datasets do not require a hybrid bias; or MCS's model classes are too similar to illustrate the utility of hybrid classifiers. Since that article was published, we have changed MCS as described in Section 2.2. In addition we have added several new datasets to our test suite. Our new results illustrate that for some datasets, MCS does create hybrid classifiers that are statistically significantly more accurate than classifiers constructed from each of its primitive components.

We present the results of two experiments that illustrate that the rules in MCS choose an appropriate bias using characteristics of the dataset, computed from feedback from the learning process. Our experiments were designed to test the two hypotheses put forth in Section 1. The hypotheses were:

1. *Domain independent knowledge about characteristics in the form of feedback from learning can effectively guide an automatic algorithm selection search.*
2. *Our knowledge-based search strategy for finding a hybrid classifier will produce a classifier that is never worse than, and for some data sets is better than, any homogeneous classifier produced by its primitive components.*

We report the results of two comparisons. The first compares MCS to its primitive components. The results, reported in Section 3.3, illustrate that MCS is sometimes more accurate and is never less accurate than each of its primitive components, demonstrating that the heuristic rules solve the selective superiority problem for these algorithms. In Section 3.4 we report the results of a comparison of MCS to three other hybrid classifier construction algorithms. Each of the three applies all of the primitive algorithms in the construction of each test node in the tree. The results of the comparison illustrate that merely increasing the search space by permitting hybrids will not increase accuracy. MCS's knowledge-based approach is demonstrated to be both more accurate and less time-consuming than the alternative three hybrid methods.

### 3.1. Datasets

In this section we describe the datasets used in our empirical comparisons. The Breast Cancer, Heart Disease, Iris Plants, and Pixel datasets were used for rule development. The results of an earlier version of MCS for the Hepatitis, LED, Road Segmentation, Congressional Votes and Vowel Recognition datasets were reported in Brodley (1993).

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

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

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

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

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

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

**Iris Plants:** Fisher's classic dataset (Fisher, 1936), contains three classes of 50 instances each. Each class is a type of iris plant. Each instance is described by four numeric attributes.

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

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

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

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

**Lymphography:** This dataset consists of 148 instances, each described by nineteen attributes and labeled as one of four classes. This data was obtained from the University Medical Centre Institute of Oncology, Ljubljana, Yugoslavia.

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

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

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

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

### 3.2. *Experimental method*

In this section we describe the experimental method used in each of the two comparisons. In each experiment we compare two or more different learning methods across a variety of learning tasks. For each learning method, we performed ten runs on each dataset.<sup>3</sup> For each run we split the original data randomly into 70% training data, 20% pruning data and 10% testing data. To ensure that the distribution of instances across the classes of a dataset is the same in the training, pruning and test sets, we first sorted the data into their classes. We then dealt the instances out randomly to the training, pruning and test sets in the specified proportions (70, 20 and 10). Each method in a comparison was run using this partition.<sup>4</sup>

To estimate the accuracy of classifiers produced by each method, we average, for each method, the results of the ten runs. In the experiments we report both the sample average and standard deviation of each method’s classification accuracy for the independent test sets. To determine the significance of the differences between two learning methods we used paired *t*-tests. Because the same random splits of each dataset were used for each method, the variances of the errors for any two methods are each due to effects that are point-by-point identical.

One learning algorithm, Thermal Training, has a random component: the resulting linear combination test’s weights depend on the order in which the instances are observed. To eliminate this effect when comparing MCS to the primitive class of linear discriminant functions, we ran both of these algorithms with the same random seed. Therefore, if MCS determines that a single linear combination test (LCT) is the best classifier for the data, then the linear combination test’s weights will be identical to those produced by running the primitive learning algorithm. We used the same random seed for each of the hybrid algorithms (described in Section 3.4) so that if any of the three determines that a single LCT is the best classifier for the data, then the LCT’s weights will be identical to those produced by primitive learning algorithm for an LCT.

### 3.3. *Comparison of MCS to its primitive components*

Our first experiment is a baseline comparison of MCS to a univariate decision tree algorithm, a linear discriminant algorithm (which constructs a linear machine for multiclass tasks) that builds a classifier using all of the input features, a *k*-nearest neighbor algorithm ( $k = 1$ ), and to a method that uses a ten-fold crossvalidation (CV) over the training data to select the best method. Our goal in this first comparison is to illustrate what the accuracy of a classifier produced by MCS would be if MCS had selected a single homogeneous representation. For each of the sixteen datasets, Table 11 shows the sample average and standard deviation of the classification accuracy for ten runs. In Table 12

Table 11. MCS and its primitive components: Accuracy

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

Table 12. Comparison of MCS to primitives: Results of paired  $t$ -tests

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

we show the results of a paired  $t$ -test between MCS and each of the primitive learning methods.<sup>5</sup>

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

Table 13. Model classes in the MCS classifiers

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

The problem that no single model class will be best for all tasks is illustrated by the results for the individual model classes; for some datasets the difference between the accuracy of best and the worst of the primitive algorithms is greater than 20. In contrast, the classifiers found by MCS for each dataset were never significantly less accurate and were sometimes significantly more accurate than the *best* of the primitive algorithms, indicating that a hybrid can provide a significant performance improvement for some datasets. This property of robustness is desirable in an automated algorithm selection system. Note that these sixteen datasets are not biased toward one of the primitive algorithms; each of the primitive algorithms is best for at least three of the datasets.

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

Table 13 shows the average over the ten runs of the number of leaves and the number of test nodes, of each type of model class, in the MCS classifiers. The last two columns are not averages; the second to last column shows, out of the ten runs, the number of classifiers produced by MCS that were hybrids and the last column shows the primitive algorithm(s) that was chosen by MCS when it did not produce a hybrid. For example, MCS produced a hybrid in three of the ten runs for the Iris Plants dataset. Of the remaining seven runs, MCS selected an LCT six times and an IBC once. For one dataset, MCS produced a hybrid for each of the ten runs. For four datasets MCS produced a hybrid for nine of the ten runs. Except for the Road Segmentation data, this did not result in higher accuracy for MCS than the best of the primitive algorithms. For three cases, Pixel, Vowel Recognition and Waveform, MCS found a classifier that is identical to the best primitive algorithm. For the remainder of the datasets, the results are mixed.

Table 14. Comparison of MCS to other hybrid methods: Accuracy

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

In many cases MCS formed a hybrid classifier, but its accuracy was not significantly different from the best of the primitive algorithms. In previous work (Brodley, 1993) we reported results of an experiment that calculated the percentage overlap in the classification decisions made by MCS and the best primitive algorithm for each dataset. Our conclusion in that experiment was: because all of the primitive algorithms are piecewise linear, for many datasets they will each form classifiers that define the same decision boundaries (Figure 4 illustrates one such case).

### 3.4. Comparison of MCS to other hybrid algorithms

In our second experiment we compared MCS to three hybrid algorithms, which have the same primitive components as MCS, but different search strategies. The three algorithms, RAcc, RInfo and RCV each search for a best classifier for each node by trying all possible algorithms. All three have the same general recursive procedure shown in Table 1. They differ from MCS in how a best algorithm is chosen for a dataset. The classifiers compared at each node in the tree are a univariate test, a  $k$ -nearest neighbor (they search for the best value of  $k$ ) and two linear combination tests, one constructed using SFS the other using SBE.

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

The results of the experiment are shown in Table 14. In Table 15 we show the results of paired *t*-tests of MCS and each of the other hybrid methods. MCS is statistically significantly more accurate than RAcc for three datasets. MCS is significantly more accurate than RInfo for three datasets and more accurate than RCV for three datasets. None of the methods is significantly more accurate than MCS for any of these sixteen datasets.

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

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

In contrast, MCS is never worse than the best of the primitive algorithms. MCS’s strategy for selecting an algorithm is itself a static bias; the rules do not change for a particular dataset. However, MCS’s strategy does not depend on only one source of feedback; the rules are designed to account for situations in which two or more measures yield conflicting results. For example, even if an LCT (based on *i* features) is more accurate than an IBC ( $k=1$ ) for the training data, MCS continues to explore IBCs if more than half of the features remain in the LCT or the difference in accuracy is less than 10% (see Table 9).

From the results of this experiment we conclude that MCS is more robust than each of the other methods. In addition, MCS is, on average, less time consuming than each of the other hybrid classifier construction algorithms. In Table 16 we show the average across the ten runs of the number of seconds that each method used to form a hybrid classifier.<sup>6</sup> MCS required less time than RAcc in twelve cases, RInfo in thirteen cases and RCV in all sixteen cases. For cases in which RAcc and RInfo took less time than MCS, the difference can be attributed to RAcc and RInfo selecting tests early on that preclude further search. For example, when an instance-based classifier is chosen for which *k* is equal to 1, no subtree will be grown below such a node. If for many nodes

Table 15. Comparison of MCS to other hybrid methods: Results of paired *t*-tests

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

in the tree an IBC is selected over a linear combination or a univariate test, and if none of the three is 100% accurate for the training data, search terminates more quickly than if the linear combination or univariate test were chosen. For example, RAcc produced hybrid trees for the Diabetes dataset that on average have 4.6 leaves (the trees produced by MCS have on average 12.7 leaves) and for which the majority of the test nodes are instance-based classifiers.

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

### 3.5. Implications of the results

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

In addition, our results support the claim that our knowledge-based search strategy for finding a hybrid classifier produces classifiers that are sometimes better, but never worse than a homogeneous classifier. Although we have no analytical proof of this claim, our results across sixteen datasets in the empirical evaluation of MCS provide strong support for such a claim. MCS created classifiers that were significantly more accurate than each of the homogeneous classifiers for three of the sixteen datasets. For the remaining

Table 16. Training time of hybrid algorithms (seconds)

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

thirteen, the accuracy of the hybrid classifiers produced by MCS was equal to the *best* of the homogeneous learning algorithms.

#### 4. Conclusion

In this article we have presented a recursive heuristic approach to automatic bias selection for classifier construction. We demonstrated that a recursive automatic algorithm selection system, MCS, can evaluate a dataset, using feedback from the learning process, to select an appropriate representation-language bias from a set of candidate biases. A fundamental aspect of our approach is the use of knowledge about the biases of machine learning algorithms and their representation languages to guide a search for a best learning bias for a given dataset. Our experimental results demonstrated that MCS performed as well as or better than each of its primitive learning algorithms across a variety of learning tasks, illustrating that MCS is a robust method for choosing among the biases of decision trees, linear discriminant functions and instance-based classifiers. In MCS we have solved the selective superiority problem for these algorithms by providing a robust automatic algorithm selection system.

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

that using MCS increases the number of datasets for which a good generalization can be found over its primitive components.

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

One limitation of the current implementation of our approach is that all three representation languages are piecewise linear. We conjecture that with the addition of non-linear model classes, the hybrid classifiers will outperform the homogeneous classifiers more frequently. The addition of a new model class to MCS will require the addition of new heuristic rules, which in turn requires understanding the biases of the new model class in relation to the biases of the existing model classes in MCS.

### Acknowledgments

This material is based on work supported by the National Science Foundation under Grant No. IRI-9222766, and by the National Aeronautics and Space Administration under Grant No. NCC 2-658. The Landsat and Road datasets were donated by B. Draper from the Vision Lab at UMASS. M. Zwitter and M. Soklic donated the Lymphography dataset. All other datasets are from the UCI machine learning database. Discussions with Jeff Clouse helped to clarify the issues presented in this article. The anonymous reviewers' and the editors' suggestions greatly improved the clarity and content of this article. Thanks to Paul Utgoff, whose comments improved both the content and presentation of this article, and for providing the inspiration for this research.

### Notes

1. Anorexia patients typically have a potassium deficiency due to lack of food, which in turn may cause heart failure.
2. Lack of compression is used to decide whether an instance-based classifier should be tried. Therefore the rules do not test whether an IBC compresses the data.
3. For the Vowel dataset, we used the original training and test sets and therefore did not perform multiple runs.
4. Our goal in ensuring an even distribution of classes among the training, testing and pruning sets was to reduce variation in performance across different runs. We wanted both the data used for training and testing to mirror the distribution of the entire dataset.

5. For the Vowel dataset, we used the original training and test sets and therefore ran each algorithm once.
6. All algorithms were performed on a DEC 3000 running under OSF/1.

## References

- 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., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6, 37-66.
- Aha, D. W. (1992). Generalizing from case studies: A case study. *Machine Learning: Proceedings of the Ninth International Conference* (pp. 1-10). San Mateo, CA: Morgan Kaufmann.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.
- Breiman, L. (1992). *Stacked regressions*, (Technical Report No. 367), University of California, Berkeley.
- Brodley, C. E. (1993). Addressing the selective superiority problem: Automatic algorithm/model class selection. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 17-24). Amherst, MA: Morgan Kaufmann.
- Brodley, C. E. (1994). *Recursive automatic algorithm selection for inductive learning*. Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.
- Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning*, 19, 45-77.
- Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3, 261-283.
- Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64, 304-310.
- Dietterich, T. G. (1990). Machine learning. *Annual Review of Computer Science*, 4.
- Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley & Sons.
- Feng, C., Sutherland, A., King, R., Muggleton, S., & Henry, R. (1993). Comparison of machine learning classifiers to statistics and neural networks. *Preliminary Papers of the Fourth International Workshop on Artificial Intelligence and Statistics* (pp. 41-52).
- Fisher, R. A. (1936). Multiple measures in taxonomic problems. *Annals of Eugenics*, 7, 179-188.
- Frean, M. (1990). *Small nets and short paths: Optimising neural computation*. Doctoral dissertation, Center for Cognitive Science, University of Edinburgh.
- Kittler, J. (1986). Feature selection and extraction. In Young & Fu (Eds.), *Handbook of pattern recognition and image processing*. New York: Academic Press.
- LeBlanc, M., & Tibshirani, R. (1993). *Combining estimates in regression and classification*, (no number), University of Toronto.
- Linhart, H., & Zucchini, W. (1986). *Model selection*. NY: Wiley.
- Mangasarian, O. L., & Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News*, 23, 1-18.
- Nilsson, N. J. (1965). *Learning machines*. New York: McGraw-Hill.
- Provost, F. J., & Buchanan, B. G. (1992). Inductive policy. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 255-261). San Jose, CA: MIT Press.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1, 81-106.
- Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies*, 27, 221-234.
- Quinlan, J. R. (1993). Combining instance-based and model-based learning. *Machine Learning: Proceedings of the Tenth International Conference* (pp. 236-243). Amherst, MA: Morgan Kaufmann.
- Rendell, L., & Cho, H. (1990). Empirical learning as a function of concept character. *Machine Learning*, 5, 267-298.
- Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. New Jersey: World Scientific.
- Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning*, 6, 251-276.
- Schaffer, C. (1993). Selecting a classification method by cross-validation. *Preliminary Papers of the Fourth International Workshop on Artificial Intelligence and Statistics* (pp. 15-25).
- Shavlik, J. W., Mooney, R. J., & Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6, 111-144.

- 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.
- Utgoff, P. E. (1989). Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1, 377-391.
- Utgoff, P. E., & Brodley, C. E. (1991). *Linear machine decision trees*, (COINS Technical Report 91-10), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.
- Weiss, S. M., & Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 781-787). Detroit, Michigan: Morgan Kaufmann.
- Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5, 241-259.
- Yerramareddy, S., Tcheng, D. K., Lu, S., & Assanis, D. N. (1992). Creating and using models for engineering design. *IEEE Expert*, 3, 52-59.
- Zhang, X., Mesirov, J. P., & Waltz, D. L. (1992). Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225, 1049-1063.

Received November 15, 1993

Accepted August 12, 1994

Final Manuscript September 1, 1994