Addressing the Selective Superiority Problem: Automatic Algorithm/Model Class Selection

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

Abstract

The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a *selective superiority*; it 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 such cases one must search the space of available algorithms to find the one that produces the best classifier. In this paper we present an approach that applies knowledge about the representational biases of a set of learning algorithms to conduct this search automatically. In addition, the approach permits the available algorithms' model classes to be mixed in a recursive tree-structured hybrid. We describe an implementation of the approach, MCS, that performs a heuristic best-first search for the best hybrid classifier for a set of data. An empirical comparison of MCS to each of its primitive learning algorithms, and to the computationally intensive method of cross-validation, illustrates that automatic selection of learning algorithms using knowledge can be used to solve the selective superiority problem.

1 THE PROBLEM OF SELECTIVE SUPERIORITY

Several dozen inductive learning algorithms have been developed over the last few decades, including versions of perceptron, DNF cover, decision tree, instance-based, and neural-net algorithms. In every case, the algorithm can boast one or more superior learning performances over the others, but none is always better. The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a *selective superiority*: it is best for some but not all tasks (Weiss & Kapouleas (1989), Aha, Kibler & Albert (1991), Shavlik, Mooney & Towell (1991), Salzberg (1991).) This is because each algorithm is biased, leading to a good fit for some learning tasks, and a poor fit for others.

An algorithm's success in finding a good generalization for a given data set depends on two factors. The first is whether the algorithm's representation space contains a good generalization. In statistics, this space is called the algorithm's *model class*. For example, the model class of a symbolic, univariate decision tree algorithm is the Disjunctive Normal Form. The model class of a perceptron learning algorithm is the class of linear discriminant functions. Because an algorithm's model class defines the space of possible generalizations, not even an exhaustive search strategy can overcome a poor choice of model class. The second factor of an algorithm's success is its search bias. Even algorithms that search the same model space have shown selective superiority. For example, if a set of instances is not linearly separable then the Least Mean Squares (LMS) training rule provides a better solution than the Absolute Error Correction rule (ACR) for learning the weights of a linear discriminant function (Duda & Hart, 1973). However, when the instances are linearly separable the situation is reversed; LMS is not guaranteed to find a separating hyperplane, whereas ACR is. In many cases, the choice of learning algorithm can be made on subject matter considerations. However, when such prior knowledge is not available, the choice can be made by comparing each algorithm's performance on the data.

2 AUTOMATIC ALGORITHM SELECTION

Selecting the best algorithm for a data set in the absence of prior knowledge is a search problem. One well-known approach from statistics is to use cross-validation (Linhart & Zucchini, 1986). Given a set of data, an n-fold cross-validation splits the data into n equal parts. Each candidate algorithm is run n times; for each run, $n-1$ parts of the data are used to form a classifier, which is then evaluated using the remaining

part. The results of the n runs are averaged and the algorithm that produced classifiers with the highest average classification accuracy is selected. Recently, Schaffer (1993) applied this idea to selecting a classification algorithm. The results of an empirical comparison of a cross-validation method (CV) to each algorithm considered by CV, illustrated that on average, across the test-suite of domains, CV performed best.

A cross-validation strategy performs an exhaustive search through the space of candidate methods. As the number of alternatives increases, the time required to search for the best algorithm may become impractical due to the computational expense of performing a cross-validation. One of the oldest methods in Artificial Intelligence for reducing search effort is to use knowledge about the problem domain. Here the problem domain is selection of the best learning algorithm for a given set of data. The search space is defined by the data set and the set of candidate algorithms. In this domain, knowledge stems from understanding the biases of the candidate algorithms and how this can be applied to guide the search for the best algorithm/model class for a given set of data.

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; Rendell & Cho, 1990). Systems that allow the user to specify an inductive policy, require that the biases of the available learning algorithm be known and be represented explicitly (for manipulation during search) (Provost & Buchanan, 1992). The knowledge resulting from such efforts can be used to form a heuristic search procedure for automatic algorithm selection. To this end, we present a knowledge-based approach to algorithm/model class selection. Our approach differs from cross-validation in two fundamental ways. Firstly, algorithm selection is guided by various kinds of hypothesis pathology that develop when using an algorithm that is a poor choice. We define hypothesis pathology to be the recognizable symptoms of when a hypothesis fits the data poorly. When an algorithm and its underlying model class is inappropriate, one needs to find a different one that is appropriate. The ability to do this depends both on recognizing whether and why the algorithm is a poor choice, and on using this information to select a better one. We have encoded this knowledge as a set of heuristic rules. Secondly, our approach is applied recursively, providing a mechanism for mixing the available model classes to form a hybrid classifier.

3 KNOWLEDGE-BASED SEARCH

Our approach of using hypothesis pathology to guide the search has the ability to assess why a particular model class or search bias is inappropriate and from this assessment can make an informed choice as to which to try next. If a hypothesis representation is judged inappropriate, then we want to redirect the search to a part of the space that will produce a good generalization. The type of pathology dictates where in the space to redirect the search.

A simple approach was illustrated in the work on perceptron trees (Utgoff, 1989). The algorithm first tried a simple perceptron as the test, but if it was doing poorly, it was replaced by a single variable test. This approach detected hypothesis pathology, and selected a new representation, but the pathology was not used to guide the selection, because there was only a single alternate choice.

FRINGE (1990) uses a different approach. The concept representation is changed by adding compound terms, based on those found at the fringe of the decision tree, and rebuilding the tree. When a compound term would be useful, and is missing, the tree contains many replicated subtrees. When such pathology exists, the terms that would be useful in a compound term often appear near the fringe of the tree. Thus, the hypothesis pathology indicates a specific change in bias through the addition of compound terms.

Our approach to searching the model space is dynamic; given a set of training instances, a best-first search for the appropriate classifier is performed. A set of heuristic rules is used to decide which model class to try next, which search bias to use, and to determine when the best classifier has been found. The best classifier found is then used to partition the instance space, and the search is applied recursively to each resulting partition. If during the best-first search, a *subtree* of the partially formed classifier exhibits some type of pathology, the approach can backtrack and replace the subtree with a different classifier.

The heuristic rules are created from practical knowledge about how to detect when a generalization is a good fit, or whether a better one could be found by changing the model class or search bias. For example, the model class of a univariate decision tree is a poor choice when the features are related numerically. In this case, the features will be tested repeatedly 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. The following rule not only detects this situation, but also directs the search to the appropriate model class.

IF two or more features are tested repeatedly
in a path of the tree
THEN switch from univariate test to linear test
AND fit a linear test of these features to the
instances at the top of the path

4 RECURSIVE COMBINATION OF MODEL CLASSES

Our method builds a hybrid decision tree, with the test at a node being created by selecting and applying a primitive learning algorithm, each with a different underlying model class. The ability to combine the different model classes and apply different learning algorithms allows for the possibility that the data set is not best represented by a homogeneous classifier. For some data sets, combining heterogeneous model classes will produce the best classifier. The ability to combine model classes in a single tree structure allows a space of classifiers that is strictly richer than the union of its constituent primitive model classes. The case of picking a single model class is not lost, but the approach also permits hybrid classifiers. In our approach, each model class is permitted anywhere in the recursive tree structure. Both perceptron trees (Utgoff, 1989) and CRL (Yerramareddy, Tcheng, Lu & Assanis, 1992) restrict where in the tree certain model classes are permitted. For example, in perceptron trees, internal test nodes must be univariate and symbolic, whereas leaf-node tests (tests right above the leaves) must be perceptrons.

5 MCS: A MODEL CLASS SELECTION SYSTEM

We have implemented the approach in a system that we call the *Model Class Selection (MCS) System*. Given a set of data, MCS builds a hybrid classifier using a set of heuristic rules to guide a best-first search for the best model class for each node in the hybrid classifier. To avoid overfitting, the classifier is pruned back to reduce the estimated classification error, as computed for an independent set of instances (Breiman, Friedman, Olshen & Stone, 1984). MCS's pruning procedure differs from the traditional pruning phase of decision tree algorithms. During MCS's tree-formation stage, if an alternative classifier to the one selected appears almost as good, it is stored for the pruning phase. When considering whether to prune back a subtree in the hybrid classifier, MCS computes the classification errors of both the subtree and the alternative classifier. MCS compares the error rates to choose among: keeping the subtree, replacing it with the alternative classifier or replacing it with a leaf node containing the most frequently observed class for the instances used to form the subtree. In the next two sections we describe MCS's model classes and rule base.

5.1 MODEL CLASSES

MCS combines three primitive representations that have been used extensively in both machine learning and statistics algorithms: linear discriminant functions, decision trees and instance-based classifiers. 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. A univariate decision tree splits the instance space with cuts orthogonal to each of the axes, forming a set of hyper-rectangular regions, each labeled with a class name. An instance-based classifier defines a piece-wise linear partition of the instance space; the number of pieces is determined by the number and distribution of the instances. The result is a set of regions, each labeled by a different class name, separated by piece-wise linear boundaries.

Each of the three model classes is a subclass of the model class of piece-wise linear partitions. A univariate test in a decision tree is a special case of a linear discriminant function: a linear discriminant function can be based on all n of the input features, only one feature or some subset of the n input features. A linear discriminant function defines only one hyperplane boundary in the instance space, whereas an instance-based classifier forms a series of hyperplane boundaries. In Figure 1a we show an instance space for which classifiers from each of the three model classes would define an identical partition of the instance space.

For many data sets, classifiers from each of the three model classes will define different partitions. Figures 1b, 1c, and 1d 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.

For each model class there are many different algorithms for searching for the classifier that best fits the data. The *fitting algorithms* 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).

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 a linear discriminant function the system uses the Recursive Least Squares procedure (Young, 1984) for two-class tasks and the Thermal Training rule (Frean, 1990; Brodley & Utgoff, 1992) for multiclass tasks. To select the terms to use with a linear discriminant function, one of four search procedures is used: sequential backward elimination (SBE) (Kittler, 1986), a variation of SBE that uses the form of the function to determine which terms to eliminate (Brodley & Utgoff, 1992), sequential forward selection (SFS) (Kittler, 1986), and a method that examines a decision tree for replicated tests of a subset of the features, to suggest which terms to consider. The choice of which of these search biases to use

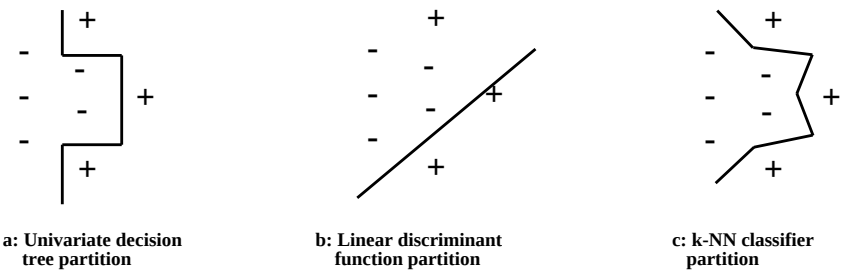

Figure 1: Partitions of the Instance Space

is determined dynamically during learning, depending on the hypotheses that have already been formed and what their pathologies suggested.

Instance-Based Classifiers: MCS uses two algorithms for fitting an instance-based classifier to the data. The first is the k-nearest neighbor algorithm (Duda & Hart, 1973), which stores each instance of the training data. To find k, the system uses the following measure of accuracy: for each instance in the training data, classify that instance using the remaining instances. MCS searches for the value of k that produces the most correct classifications on the training data. The second algorithm tries to eliminate instances that do not contribute to the classifier's accuracy and is similar in spirit to IB3 (Aha, Kibler & Albert, 1991). To this end, when the accuracy of the k-NN is computed MCS keeps track, for each instance, of the number of times it was a k-nearest neighbor and whether its class matched that of the instance being classified. Instances for which the number of incorrect matches was higher than the number of correct matches are removed.

5.2 RULE BASE

The heuristic rules use several types of pathology detection criteria to guide the search for the best classifier. The following criteria are used in the rules to detect the success or failure of a hypothesis and provide information about the reasons of failure:

1. **Information Gain-Ratio:** This measure is used to compare any two partitions of the instances; a partition can be defined by a univariate test, a linear discriminant function, a k-NN classifier, or even an entire subtree.
2. **Codelength:** Our use of this measure is based on the Minimum Description Length Principle, which states that the best "hypothesis" to induce from a data set is the one that minimizes the length of the hypothesis plus the length of the exceptions (Rissanen, 1989). The *codelength* of a classifier (the hypothesis) is the number of bits required to code the classifier and the the error vector resulting from using the classifier to classify the instances. This metric is used to determine by how much a symbolic or linear test *compresses* the data. We define the amount of compression to be the codelength of the errors without the test minus the codelength of the test plus the codelength of the errors resulting from using the test to classify the data. We use the exact error code for exceptions given in Rissanen and encode integers using Elias's asymptotically optimal encoding (Rissanen, 1989).
3. **Concept Form:** Analysis of a classifier's form can be used to guide the search for the best classifier. For example, MCS examines a partially formed tree to determine whether a set of univariate tests is tested repeatedly in the tree. The action that is taken depends on the form of the replication. Examination of a linear combination test yields information about the relative importance of the different features to classification, which in turn yields information about which search bias and model class will be most appropriate.
4. **Information about the Data Set:** We use one measure of training instances: the number of instances relative to the number of features.

The process of selecting an algorithm to apply to a data set, often requires knowledge about how to determine whether the best classifier has been found, and if not, which algorithm to try next. Many learning practitioners and statisticians make these decision by using statistics (diagnostics) computed about the resulting classifier and by examining the classifier's form. In MCS this knowledge is encoded explicitly in the rule base. In Table 1 we show a selection of rules from MCS. Rule 1 begins the search and is based on Duda and Hart's (1973) observation that when there are fewer than $2n$ instances (the capacity of a hyperplane) a linear test will *underfit* the training instances; the fewer the instances, the higher the probability that the data will be fit by chance. Rule 2 examines the univariate test to determine whether the initial decision was correct. If the test is not 100% accurate, then MCS searches for a multivariate test using SFS if the average squared difference between the informa-

Table 1: A Selection of Rules from MCS's Rule Base

1.) IF the number of instances is less than the capacity of a hyperplane
THEN find the best univariate test
ELSE form linear test LT_n of all n features

2.) IF the best univariate test is 100% accurate
THEN select it and recurse.
ELSE
IF the test does not compress the data
THEN fit a k-NN (k=2)
ELSIF the average difference in info-score of each feature to best feature is $< \epsilon$
THEN start with best feature and search for a better test using SFS
ELSE select the univariate test and recurse

3.) IF the instances are linearly separable (determined by accuracy of LT_i)
THEN continue trying to eliminate noisy and irrelevant features
ELSE examine the magnitudes of the weights of the linear test:
IF a few weights are larger than the others
THEN find best univariate test
ELSIF many of the weights are close to zero
THEN use greedy SBE
ELSE use SBE to find the best linear test based on the fewest features

4.) IF info-score(LT_i) $>$ info-score(LT_i) OR $i = 1$
THEN IF best linear test compresses data
THEN select that test and recurse
ELSE find the best univariate test
ELSE continue trying to eliminate noisy and irrelevant variables

5.) IF neither a univariate nor a linear test compress the data (based on MDLP)
THEN find the best value of k for an instance based classifier AND select from available tests, the one with max info-score, recurse

6.) IF the best univariate test's info-score is $>$ than that of a linear test of n features
THEN search for a better test starting with the univariate test and using SFS
ELSE examine form of the linear test to choose SBE or greedy SBE (see Rule 3)

tion score of each of the other features and the best feature is less than a threshold (10% of the magnitude of the best score). Using Rules 2 and 5, MCS checks to see that the best test found thus far compresses the training data, which occurs when the codelength of the test plus the codelength of the error vector, resulting from using the test to classify the data, is less than the codelength of the error vector of the instances. If the number of bits to code the test is not less than the number of bits to code the errors that the test corrects, then MCS fits a k-NN classifier to the data. Rule 3 determines whether the initial selection of a linear test was appropriate. If the instances are linearly separable then MCS tries to eliminate any noisy or irrelevant features from the test; otherwise MCS examines the form of the test to decide where next to direct the search. Rule 4 evaluates the best linear test found in the search; if the test compresses the data then it is selected; otherwise a search for the best univariate test is conducted (if one has not been found yet). Rule 6 starts a SFS for the best linear test if the information score of a univariate test is higher than that of a multivariate test based on all n features.

Currently, MCS's rule base contains twenty-five rules, built using an iterative approach. Candidate rules were derived based on the author's knowledge of how to detect hypothesis pathology and whether a hypothesis is a good fit to the data. Each rule was then tested using a set of four development data sets (the Breast, Bupa, Cleveland and Segmentation data sets from the UCI data base). Based on statistics kept of the rule's usage, in comparison to the other rules' usage, and on a detailed trace of the search through the generalization space, the rule was either kept, rejected, or altered and re-evaluated for inclusion into the rule base. The rules were constructed such that only one rule will match; no conflict resolution is required. The author makes no claim that the current set of rules is complete or that better rules do not exist. However, as the results of the next section illustrate, general rules can be found that work well across many data sets.

6 ILLUSTRATION

To illustrate that knowledge about the biases of learning algorithms can be used successfully by an automatic algorithm selection system, we compare MCS to each of the individual model classes: a k-NN, a linear test based on all of the input features (LT) and a univariate decision tree (Utree). In addition, we include a hybrid algorithm LMDT (Brodley & Utgoff, 1992), which combines linear machine tests with decision trees. To determine how a knowledge-based approach compares to the traditional cross-validation scheme we report results for an algorithm that we call CV-10, which performs a ten-fold cross-validation on the training data to select the best of k-NN, LT, Utree or LMDT. CV-10 chooses the algorithm that achieves

Table 2: Accuracy for Test Data (Percent Correct)

Method	Hep.	LED	Road	Votes	Vowel
k-NN	82.63	62.12	78.95	93.24	94.66
LT	83.95	68.80	70.69	95.46	45.22
LMDT	83.42	72.56	83.01	95.37	75.18
Utree	78.95	73.36	83.19	95.09	68.02
CV-10	81.58	72.40	83.19	94.63	94.66
MCS	82.89	72.76	82.79	95.28	94.66

the highest average accuracy and then runs that algorithm on the entire training set.

None of the data sets used in this illustration were used during development of the rule base. The data sets (all available in the UCI repository) are:

Hepatitis: The task for this domain is to predict from test results whether a patient will die from hepatitis. There are 155 instances described by 19 numeric and Boolean attributes.

LED: The data for the 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.

Road: 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 8212 instances in this data set and 381 values are missing.

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.

Vowel: The task is to recognize the eleven steady state vowels of British English. There are ten numeric features describing the 990 instances.

To compare the performance of the six learning methods we ran each ten times. For each run, the data were split randomly into a training and a test set, with 75% of the data in the training set and 25% in the test set. To determine the significance of the differences among the learning methods we used paired t-tests. Because the same random splits of each data set were used for each method, the variances of the errors for any two methods are each due to effects that are point-by-point identical. For algorithms that require a separate data file for pruning (Utree, LMDT and MCS) one third of the training data was reserved for pruning.

Table 2 shows the sample average of each method's classification accuracy on the independent test sets.

Table 3: Time Used to Select a Classifier (Seconds)

Method	Hep.	LED	Road	Votes	Vowel
CV-10	390	598	29,593	838	1,263
MCS	174	130	12,237	246	218

Table 4: Model Classes in the MCS Classifiers

Model Class	Hep.	LED	Road	Votes	Vowel
k-NN	0.5	1.7	4.0	0.1	1.0
Linear	0.6	3.5	9.3	1.0	0.0
Univariate	1.7	2.2	23.4	0.9	0.0

Looking first at the individual algorithms, we observe typical selective superiority results. Each of k-NN, LT and UTree was best for at least one of the data sets (LMDT was close to the best for several data sets), and each was statistically significantly worse (at the 0.05 level) than the best method for other data sets. MCS's and CV-10's performances on the other hand, do not show a selective superiority for these data sets - they are robust across the set of tasks. Both CV-10 and MCS outperform each of the other methods *on average*. MCS yields slightly higher accuracies than CV-10 for four data sets, but these differences are not significant. For the Hepatitis, LED and Votes data sets, one of the individual methods achieved slightly higher accuracy, but in each case the difference between the best individual method and MCS is not significant. However both CV-10 and MCS are always statistically significantly more accurate than the worst methods for each data set.

MCS required far less time than CV-10 and achieved the same level of accuracy on all test problems. In Table 3 we report the number of CPU seconds used by CV-10 and MCS on a DEC Station 5000/200. This result illustrates that applying knowledge reduces the search required to find the best classifier without compromising the quality of the solution.

The main point to take away from this comparison is not that MCS "beats" the individual algorithms in terms of accuracy. MCS achieves significantly better accuracies than the worst methods and is competitive with the best method for each data set. Therefore, we conclude from this comparison that MCS's performance demonstrates that knowledge can be successfully applied to solve the selective superiority problem and at a much lower cost than cross-validation.

One issue that requires further investigation is why the hybrid classifiers are not significantly more accurate than the individual model classes. In Table 4 we show the average number of test nodes from each model class in the classifiers found by MCS. For four out of the five data sets, MCS selected hybrid classifiers. This

Table 5: Classification Overlap (Percentage)

Pair	Hep.	LED	Road	Votes	Vowel
MCS-k-NN	86	66	83	94	**100**
MCS-LT	**91**	80	72	**99**	43
MCS-Utree	84	**85**	**85**	**99**	66
LT-Utree	84	75	70	**99**	26
LT-k-NN	86	58	66	94	43
Utree-k-NN	78	64	79	94	66

raises the question of why the hybrid classifiers did not achieve *higher* accuracy than each of the homogeneous classifiers for these four data sets.

There are several possible answers. Firstly, for these data sets it may be impossible to achieve higher accuracy than that obtained by the best individual method due to noise in the instances. A second possibility is that MCS needs better rules to show the advantages of a hybrid classifier. Finally, MCS's model classes may be too similar to show a difference. MCS creates hybrid classifiers that are piece-wise linear. Each of the primitive model classes is also piece-wise linear, albeit with different restrictions on how they can partition the instance space and with different search biases.

To cast insight into which of the possible explanations seems most likely, we examine the differences among the decision boundaries each classifier forms. To this end, we compute a heuristic measure of the similarity between two classifiers' boundaries: we compute the percentage overlap in the classification decisions that each makes. Although this does not tell us the exact difference in the decision boundaries, it gives a rough measure of how different these boundaries are for classification of the types of instances observed.

After each algorithm was run on a random partition of a data set, we computed the overlap between the classification decisions made by each pair of learning algorithms; we counted the number of instances that were classified the same way by each of two classifiers constructed using different algorithms. In Table 5 we report the average for each data set, over the ten runs, of the classification overlap percentage between each pair of classifiers. The higher the percentage, the closer the decision boundaries are for classification of instances likely to be observed in the domain. For example, the largest pair-wise overlap for the Hepatitis data set is between MCS and LT. This indicates that MCS defines decision boundaries that are more similar to LT's boundaries than Utree's or k-NN's. Combining the results reported in Tables 2 and 5, we observe that MCS's classification overlap is highest with the best individual method and therefore its decision boundaries are in highest agreement with those of the best individual method. In addition, MCS's overlap with the best method was higher than any pair-wise overlap between the primitive algorithms for four of the data sets. The one exception is the Votes data set; both LT and Utree have an almost perfect overlap with MCS and therefore show a similarly high pair-wise overlap.

Given that, on average, the hybrid classifiers classify test instances most similarly to the best of the primitive algorithms for each data set, we find that the two most likely explanations for why the hybrid classifiers did not outperform the best of LT, Utree and k-NN on each data set are: it is impossible to achieve higher accuracies on these data sets than the best of the homogeneous classifiers or MCS's *model classes* are too similar to improve classification performance over the best homogeneous classifier for a given data set.

7 FUTURE WORK

In our experiments the hybrid classifiers produced by MCS did not achieve higher accuracy than the best homogeneous classifier for each of the five data sets. Therefore, we were not able to demonstrate that using different *model classes* for different subspaces of a learning task (to form a hybrid classifier) leads to the best classifier. Our classification overlap analysis suggests two possible explanations that need further investigation. Firstly, greater benefits of hybrid classifiers may be realized through the inclusion of model classes that do not form piece-wise linear partitions; with the inclusion of classes such as splines or quadratic discriminant functions, the benefits of heterogeneous classifiers may be more clearly illustrated. A second issue to investigate is whether data sets exist that are best represented by a hybrid classifier. Certainly, we can create artificial data sets for which this is true, but from the results of the illustration it is unclear whether better accuracies can be achieved for the five data sets, than the accuracy of the best of the set of classifiers constructed using the primitive algorithms.

As more knowledge about the biases of machine learning, symbolic and statistical classification algorithms becomes available, the efficiency of the search and the quality of the solutions will improve. One direction in which to extend the recursive knowledge-based is to incorporate the results of efforts such as the StatLog project to reduce the amount of search required. One of the results of the StatLog project is a study that relates statistical measures of a data set to the performance of different algorithms (Feng, et al. 1993). Measures such as homogeneity of covariances and skewness were used to explain differences in the performances of a set of machine learning, statistical and neural net algorithms on a test-suite of classification tasks. The knowledge resulting from this type of study could be used to begin the best-first search. By starting the search from a promising part of the space, it stands to reason that the time required to find the best classifier would be greatly reduced.

8 CONCLUSION

In this paper we have presented a recursive heuristic approach to algorithm selection for classifier construction. We illustrated empirically that an automated algorithm selection system, MCS, can use knowledge about the biases of machine learning algorithms and representations to solve the selective superiority problem. Clearly, when prior knowledge of which model class will be best for a data set exists, it should be applied. Such knowledge does not preclude the use of systems such as MCS; it can be used to focus the search in the most promising areas, thereby reducing the amount of search required. As we as researchers gain more insight into the biases of the algorithms that we create, we will be able to improve the performance of systems that use knowledge to perform automatic algorithm selection.

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NCC 2-658. The Hepatitis, LED, Votes and Vowel data sets are from the UCI machine learning database. B. Draper provided the Road data. Discussions with J. Callan, J. Clouse, T. Fawcett, C. Schaffer and P. Smyth helped to clarify the issues presented in this paper. Thanks to Paul Utgoff, whose comments improved both the content and presentation of this paper, and for providing the inspiration for this research.

References

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.

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

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).

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.

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

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

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

Provost, F. J., & Buchanan, B. G. (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.

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

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

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

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.

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

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

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

Young, P. (1984). *Recursive estimation and time-series analysis.* New York: Springer-Verlag.