, , 1–25 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

On the Accuracy of Meta-learning for Scalable Data Mining

PHILIP K. CHAN pkc@cs.fit.edu
Computer Science, Florida Institute of Technology, Melbourne, FL 32901

SALVATORE J. STOLFO sal@cs.columbia.edu
Department of Computer Science, Columbia University, New York, NY 10027

Received May, 1995

Editor:

Abstract. In this paper, we describe a general approach to scaling data mining applications that we have come to call *meta-learning.* Meta-Learning refers to a general strategy that seeks to learn how to combine a number of separate learning processes in an intelligent fashion. We desire a meta-learning architecture that exhibits two key behaviors. First, the meta-learning strategy must produce an accurate final classification system. This means that a meta-learning architecture must produce a final outcome that is at least as accurate as a conventional learning algorithm applied to all available data. Second, it must be fast, relative to an individual sequential learning algorithm when applied to massive databases of examples, and operate in a reasonable amount of time. This paper focussed primarily on issues related to the accuracy and efficacy of meta-learning as a general strategy. A number of empirical results are presented demonstrating that meta-learning is technically feasible in wide-area, network computing environments.

Keywords: machine learning, meta-learning, scalability, data mining, classifiers.

1. Introduction

Many believe that we are poised once again for a radical shift in the way we learn and work, and in the amount of new knowledge we will acquire. The coming age of high performance network computing, and widely available "data highways" will transform the "information age" into the "knowledge age" by providing new opportunities in defense, commerce, education and science for sharing and utilizing information. However, with this new technological capability comes along a number of hard technical problems, many centered on the issue of scale. It is perhaps obvious that having massive amounts of data and information available anywhere and anytime enables many new opportunities to acquire new knowledge. The field of *data mining* studies how precisely this will be achieved in an efficient and transparent fashion.

One means of acquiring new knowledge from databases is to apply various machine learning algorithms that compute descriptive representations of the data as well as patterns that may be exhibited in the data. The field of machine learning has made substantial progress over the years and a number of algorithms have been popularized and applied to a host of applications in diverse fields. Thus, we may simply apply the current generation of learning algorithms to very large databases and wait for a response! However, the question is how long might we wait? Indeed, do the current generation of machine learning algorithms **scale** from tasks common today that include thousands of data items to new learning tasks encompassing as much as two orders of magnitude or more of data that is physically distributed? Furthermore, many existing learning algorithms require all the data to be resident in main memory, which is clearly untenable in many realistic databases. In certain cases, data is inherently distributed and cannot be localized on any one machine for a variety of practical reasons. In such situations it is infeasible to inspect all of the data at one processing site to compute one primary "global" classifier. We call the problem of learning useful new knowledge from large inherently distributed databases the *scaling problem for machine learning.*

Our approach to solve the scaling problem is to execute a number of learning processes (each implemented as a distinct serial program) on a number of data subsets (a *data reduction* technique) in parallel (eg. over a network of separate processing sites) and then to integrate the collective results through a process we call *meta-learning* [6]. Without any integration, as we discuss later, individual results generated from the data subsets are far from desired. Here, meta-learning serves as the means of "gluing" multiple knowledge sources together.

We note with interest that this general meta-learning approach is independent of the underlying learning algorithms that may be employed. Furthermore, it is independent of the computing platform used. Thus, our meta-learning approach is intended to be *scalable* as well as *portable* and *extensible*. However, we may not be able to guarantee the accuracy of the final result to be as good as an individual learning algorithm applied to the entire data set since a considerable amount of information may not be accessible to each of the separate learning processes. It is this primary issue we study in this paper.

2. Related Work

In a relational database context, a typical *data mining task* is to explain and predict the value of some attribute of the data given a collection of tuples with known attribute values. An existing relation with attribute values drawn from some domain is thus treated as training data for a learning algorithm that computes a logical expression, a concept description or a *classifier*, that is later used to predict a value of the desired attribute for some "test datum" whose desired attribute value is unknown.

Before the details of our approach are discussed, we first summarize closely related work by others in improving the accuracy of learning algorithms applied to large amounts of data.

Machine learning researchers clearly desire more accurate learning algorithms. One recent approach has focussed on integrating by some means multiple strategies or multiple algorithms. Some research has concentrated on methods to improve an existing algorithm by using the algorithm itself to generate purposely biased distributions of training data. The most notable work in this area is due to Schapire [17]. Schapire proves, under the theoretical PAC (*Probabilistic Approximately Correct*) learning model [20], that his *boosting* technique can improve a "weak" learner to achieve arbitrary high accuracy.

Other researchers have proposed implementing learning systems by integrating in some fashion a number of different algorithms to boost overall accuracy. The basic notion behind this integration is to complement the different underlying learning strategies embodied by different learning algorithms by effectively reducing the space of incorrect classifications of a learned concept.

There are mainly two strategies that we may consider in integrating different learning strategies. One strategy is to increase the amount of knowledge in the learning system. For example, some work has been reported on integrating inductive and explanation-based learning [12]. Explanation-based techniques are integrated to provide the appropriate domain knowledge that complements inductive learning, which is knowledge poor. This approach requires a complicated new algorithm that implements both strategies to learning in a single system.

Another strategy is to loosely integrate a number of different inductive learning algorithms by integrating their collective output concepts in some fashion. Some of these techniques are described below and later evaluated from our empirical results.

Many of the simpler techniques that aim to combine multiple evidence into a singular prediction are based on voting. The first scheme we examine is *simple voting*. That is, based on the predictions of different base classifiers, a final prediction is chosen as the classification with a plurality of votes. A variation of simple voting is *weighted voting*. Each classifier is associated with a weight, which is determined by how accurate the classifier performs on a validation set. (A validation set is a set of examples randomly selected from all available data. Since each classifier is trained on only one subset, examples in the other subsets that contribute to the validation set provide a measure of predictiveness.) Each prediction is weighted by the classifier's assigned weight. The weights of each classification are summed and the final prediction is the classification with the most weight.

Littlestone and Warmuth [14] propose several *weighted majority* algorithms for integrating different classifiers. (In their work the classifiers are different prediction algorithms, which are not necessarily learned. The training data are only used for calculating the weights.) These integrating algorithms are similar to the weighted voting method described above; the main difference is how the weights are obtained. The basic algorithm, called WM, associates each learned classifier with an initial weight. Each example in the training set is then processed by the classifiers. The final prediction for each example is generated as in weighted voting. If the final prediction is wrong, the weights of the classifiers whose predictions are incorrect are multiplied by a fixed discount β, where $0 \leq \beta < 1$, that decreases their contribution to final predictions. (A variation of the basic WM algorithm, called WML, does not allow the weights to be discounted beyond a predefined *limit*.)

Xu et al. [22] developed a method for integrating predictions from multiple classifiers based on the Bayesian formalism. The belief function they derived (Equation 32) is simplified as: $bel(class_i, x) \approx \prod_k^{classifiers} P(class_i \mid classifier_k(x))$, where x is an instance and $classifier_k(x)$ is the classification of instance x predicted by $classifier_k$. The final prediction is $class_j$ where $bel(class_j, x)$ is the largest among all classes. In our experiments reported below, we estimate the conditional probabilities from the frequencies generated from the validation set.

A more interesting approach to loosely combine learning programs is to learn how to combine independently learned concepts. Stolfo et al. [18] propose learning rules by training weighted voting schemes, for merging different phoneme output representations from multiple trained speech recognizers. Wolpert [21] presents a theory of *stacked generalization* to combine several classifiers. (Indeed, this work is closest to what we mean by meta-learning as we will describe later.) Several (level 0) classifiers are first learned from the same training set. The predictions made by these classifiers on the training set and the correct classifications form the training set of the next level (level 1) classifier. When an instance is being classified, the level 0 classifiers first make their predictions on the instance. The predictions are then presented to the level 1 classifier, which makes the final prediction. Zhang et al.'s [23] work utilizes a similar approach to learn a *combiner* based on the predictions made by three different classifiers. These latter ideas suggest a general approach that may exhibit favorable scaling characteristics as we discuss later.

Other researchers investigate different characteristics for successful integration of multiple classifiers. Ali and Pazzani [1] empirically show that classifiers with fewer uncorrelated errors reduce the error rate for the integrated model. Krogh and Vedelsby [13] prove that the overall error rate can be reduced by classifiers generating highly independent predictions.

Next we are going to detail our meta-learning approach.

3. Meta-Learning

Meta-learning is loosely defined as learning of meta-knowledge about learned knowledge. In our work we concentrate on learning from the output of concept learning systems. In this case meta-learning means learning from the *predictions* of these classifiers on common *training data*. Thus, we are interested in the output of the classifiers, not the internal structure and strategies of the learning algorithms themselves. Moreover, in several of the schemes we define, the training data presented to the learning algorithms initially are also available to the *meta-learner* under certain circumstances.

3.1. Computing Initial Base Classifiers

We consider two distinct phases in meta-learning in which data reduction is applied in two different fashions. In the first phase, "base level classifiers" are computed from the initial input database. Thus, the initial input database D, where $N = \mid D \mid$, is divided into *s random and unbiased* subsets of training data, each of (roughly) size N/s. These subsets are input to s learning algorithms, executed concurrently. In the second phase when meta-learning over a number of computed base classifiers, we may similarly partition "meta-data" across subsets of classifiers who are integrated in smaller groups. However, here we may compose distributions of meta-level training data that are purposefully biased by the classifications of the underlying base classifiers (i.e., we filter the data according to the predictions of the precomputed classifiers).

There are, however, several important considerations. We must be concerned with the bias introduced by the particular distribution formed by the data reduction method. For example, if the data are partitioned over the "class attribute" (i.e., the target concept of inductive learning) then the resultant classifiers would be specific to only a single class, and no others. This may be a poor strategy for at least two important reasons.

First, under this scheme important information that distinguishes between two classes will not be available to any learning algorithm. Thus, "near-misses", and "counter-factuals" will not be available to a learning algorithm. This may lead to "overly general" inductively inferred descriptions of the data, putting a heavier burden on meta-learning to correct the mistakes of the base classifiers. Indeed, many "discrimination based" learning algorithms require negative training examples to compute useful results. Secondly, the independent subsets of training data may still be too large to process efficiently. For example, for very large N, and a relatively small number of classes, c, the quantity N/c may itself be a large number. This implies that other attributes of the data must participate in the data reduction scheme to distribute the computation. But then we must be concerned with choosing "good distributions" that minimize any potential severe bias or skew that may lead to faulty or misleading classifiers. The importance of choosing the right attributes and the resultant impact on learning cannot be understated.

Random selection of the partitioned data sets with a uniform distribution of classes is perhaps the most sensible solution. Here we may attempt to maintain the same frequency distribution over the "class attribute" so that each partition represents a good but smaller model of the entire training set. Otherwise, a totally random selection strategy may result in the absence of some classes we wish to discriminate among in some of the training subsets. Several experiments have been conducted and are reported below to explore these issues.

3.2. Integrating Base Classifiers

Since different learning algorithms employ different knowledge representations and search heuristics, different search spaces may be explored by each and hence potentially diverse results can be obtained. Mitchell [15] refers to this phenomenon as *inductive bias*; the outcome of running an algorithm is biased towards a certain outcome. Furthermore, different partitions of a data set have different statistical characteristics and the performance of any single learning algorithm might differ substantially over these partitions. These observations imply that great care must be taken in designing an appropriate distributed meta-learning architecture. A number of these issues are explored in this paper.

How precisely do we integrate a number of separately learned classifiers? Bayesian statistics theory provides one possible approach to combining several learned classifiers based upon the *statistics* of the behavior of the classifiers on the training set. Given some set of classifiers, $C_i, i = 1..n$ and a feature vector x, we seek to compute a class label y for x. Bayes theorem suggests an "optimal" strategy as follows:

$$P(y \mid x) = \sum_i P(C_i) \times P(y \mid C_i, x)$$

$P(C_i)$ is the probability that C_i predicts correctly, (i.e., the probability it is the true model), while $P(y \mid C_i, x)$ is the probability that x is of class y given by C_i. Of course, this makes sense only when the probabilities are indeed known, and our classifiers are probabilistic and not categorical. The best we can do to estimate $P(C_i)$ is to calculate the appropriate statistics from observing the behavior of each classifier on the training set as an approximation to the actual probabilities (which may be quite inaccurate.) (Furthermore, Bayes theorem would be optimal if we knew all possible classifiers, not just those that we happen to compute.) This information, however, provides only statistics about each classifier's behavior with respect to the training set, and no information about how the classifiers are related to each other. For example, learning that two classifiers rarely agree with each other when predicting a class label y (meaning that when one classifier predicts y, the other does not) might have much more predictive value (eg. when combined with a third classifier) than merely knowing that the two classifiers predict y with equal probability! We view the purely Bayesian approach as a baseline, and use methods derived from this approach, BAYES [9] and Bayesian-belief [22], for comparative purposes in our experiments reported later. There are many other approaches we might imagine that are based upon learning relationships between classifiers. The manner in which we learn the relationship between classifiers is to *learn a new classifier* (a "meta-level classifier") *whose input is the set of predictions of two or more classifiers on common data.* It is this latter view that we call meta-learning.

In the following sections we detail meta-learning by *arbitration*, and by *combining* where in both cases a variety of inductive learning algorithms are employed to generate the appropriate meta-classifiers. Each strategy is treated in great detail including the variety of training data distributions generated in each scheme.

There are a number of important questions only poorly understood but for which substantial experimental evidence suggests directions for future exploration. In particular:

- Can meta-learning over data partitions maintain or boost the accuracy of a single global classifier?
- How do voting and Bayesian techniques compare to meta-learning in accuracy?

- How do arbiters compare to combiners in accuracy?
- A meta-learned classifier may be treated as a base classifier. Thus, might hierarchically meta-learned classifiers perform better than a single layered meta-learned architecture?
- How much training data and of what distribution should an arbiter or combiner be provided in order to produce accurate results?

A substantial number of exploratory evaluations have been completed and reported in a number of papers suggesting answers to these questions. Several of these results are repeated here for completeness in our exposition.

We have discovered through experimentation three very interesting behaviors exhibited by various meta-learning strategies that warrant further elaboration. We demonstrate that under certain circumstances, a meta-learning architecture can learn effectively with a fraction of the total available information at any one site, that accuracy can be boosted over the global classifier trained from all available data, and that maximal parallelism can be effectively exploited by meta-learning over disjoint data partitions without a substantial loss of accuracy [8]. These results suggest strongly that a "field test" of these techniques over a real world network computing environment (eg. over database server sites on the web) is not only technically feasible, but also an important next step in the development of these ideas.

In the next section we present meta-learning by arbitration and combining. Following this, we present hierarchical meta-learning. We devote considerable depth to these topics to demonstrate the range of issues involved in attempting to scale machine learning systems.

4. Meta-learning by Arbitration and Combining

We distinguish between *base classifiers* and arbiters/combiners as follows. A base classifier is the outcome of applying a learning algorithm directly to "raw" training data. The base classifier is a program that given a test datum provides a prediction of its unknown class. An arbiter or combiner, as detailed below, is a program generated by a learning algorithm that is trained on the predictions produced by a set of base classifiers and the raw training data. The arbiter/combiner is also a classifier, and hence other arbiters or combiners can be computed from the set of predictions of other arbiters/combiners.

4.1. Arbiter Strategies

An *arbiter* [7] is learned by some learning algorithm to arbitrate among predictions generated by different base classifiers. This arbiter, together with an *arbitration rule*, decides a final classification outcome based upon the base predictions. Figure 1

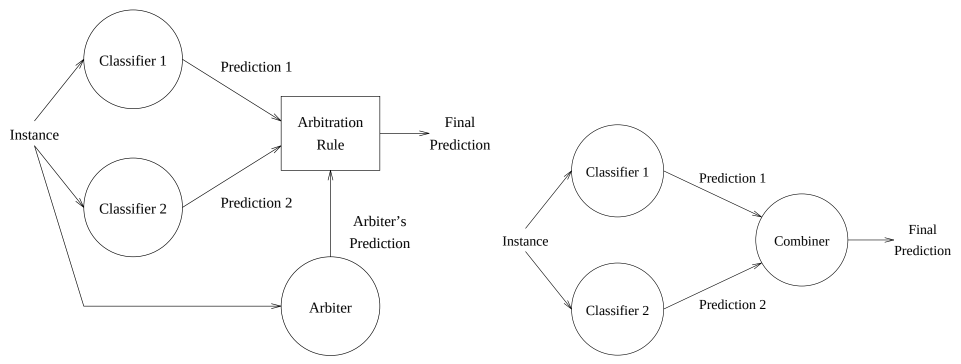

Figure 1. An arbiter and a combiner with two classifiers.

depicts how the final prediction is made with predictions input from two base classifiers and a single arbiter.

Let x be an instance whose classification we seek, $C_1(x)$, $C_2(x)$, ... $C_k(x)$ are the predicted classifications of x from k base classifiers, C_1, C_2, ... C_k, and $A(x)$ is the classification of x predicted by the arbiter. One arbitration rule studied and reported here is as follows:

- Return the class with a plurality of occurrences in $C_1(x)$, $C_2(x)$, ... $C_k(x)$, and $A(x)$, with preference given to the arbiter's choice in case of a tie.

We now detail how an arbiter is learned. A training set T for the arbiter is generated by picking examples from a *validation set E*. The validation set E is *randomly selected from all available data* prior to the onset of arbiter training. The choice of examples selected from E is dictated by a *selection rule*, that purposefully biases the arbiter training data. One version of a selection rule studied here is as follows:

- An instance from E is selected if none of the classes in the k base predictions gathers a majority classification ($> k/2$ votes); i.e., $T = \{x \in E \mid \mathit{no\_majority}(C_1(x), C_2(x), ...C_k(x))\}$.

The purpose of this rule is to choose data that are in some sense "confusing"; i.e., the majority of classifiers do not agree on how the data should be classified. Figure 2 provides an abstract example with three base classifiers trained over data about three classes, a, b and c. In our later discussion, we refer to this set of arbiter training data as *disagreements*. Once the training set is formed, an arbiter is generated by the same learning algorithm used to train the base classifiers. Together with an arbitration rule, the learned arbiter resolves conflicts among the classifiers when necessary.

Class	Attribute vector	Example	Base classifiers' predictions		
class(x)	*attrvec*(x)	x	$C_1(x)$	$C_2(x)$	$C_3(x)$
a	$attrvec_1$	x_1	a	a	a
b	$attrvec_2$	x_2	a	b	c
c	$attrvec_3$	x_3	c	b	a

Training set from the *arbiter* scheme

Instance	Class	Attribute vector
1	b	$attrvec_2$
2	c	$attrvec_3$

Training set from the *class-combiner* scheme

Instance	Class	Attribute vector
1	a	(a, a, a)
2	b	(a, b, c)
3	c	(c, b, a)

Figure 2. Sample training sets generated by the *combiner* and *arbiter* strategies

4.2. Combiner Strategies

In the *combiner* [5] strategy, the predictions of the learned base classifiers on the training set form the basis of the meta-learner's training set. A *composition rule*, which varies in different schemes, determines the content of training examples for the meta-learner. From these examples, the meta-learner generates a meta-classifier, that we call a *combiner*. In classifying an instance, the base classifiers first generate their predictions. Based on the same composition rule, a new instance is generated from the predictions, which is then classified by the combiner (see Figure 1). The aim of this strategy is to "coalesce" the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction. A combiner computes a prediction that may be entirely different from any proposed by a base classifier, whereas an arbiter chooses one of the predictions from the base classifiers and the arbiter itself.

We experimented with two schemes for the composition rule. First, the predictions, $C_1(x)$, $C_2(x)$, ... $C_k(x)$, for each example x in the validation set of examples, E, are generated by the k base classifiers. These predicted classifications are used to form a new set of "meta-level training instances," T, which is used as input to a learning algorithm that computes a combiner. The manner in which T is computed varies as defined below. In the following definitions, *class*(x) and *attribute_vector*(x) denote the correct classification and attribute vector of example x as specified in the validation set, E.

1. Return meta-level training instances with the correct classification and the predictions; i.e.,

$T = \{(class(x), C_1(x), C_2(x), ...C_k(x)) \mid x \in E\}$. This scheme was also used by Wolpert [21]. (For further reference, this scheme is denoted as *class-combiner*.)

2. Return meta-level training instances as in *class-combiner* with the addition of the attribute vectors; i.e., $T = \{(class(x), C_1(x), C_2(x), ...C_k(x), attribute\_vector(x)) \mid x \in E\}$. (This scheme is denoted as *class-attribute-combiner*.)

Note the difference in training data. Arbiters are computed form a distinguished and biased subset of data selected from the input database used to train the base classifiers. Combiners, however, are trained on the predicted classifications of that data generated by the base classifiers, as well as the data itself.

4.3. Issues

Several issues arise from our meta-learning strategies and are detailed as follows.
Number and size of training subsets: The number of initially partitioned training data subsets largely depends on the number of processors available, the inherent distribution of data across multiple platforms (some possibly mobile and periodically disconnected), the total size of the available training set, and the complexity of the learning algorithms. The available resources at each processing sites naturally defines an upper bound on the size of each subset. If the number of subsets exceeds the number of processors available, each processor can simulate the work of multiple ones by serially executing the task of each processor. Another consideration is the desired accuracy we wish to achieve. As we will see in our experimental results, there may be a tradeoff between the number of subsets and the final accuracy of a meta-learning system. Moreover, the size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective base classifier in the initial stage of training.
Distribution of examples, disjoint or replicated: Since a totally random distribution of examples may result in the absence of one or more classes in the partitioned data subsets, the classifiers formed from those subsets will be ignorant about those classes. That is, more "disagreements" may occur between classifiers, which leads to larger arbiter training sets. Maintaining the class distribution in each subset as in the total available training set may alleviate this problem. The classifiers generated from these subsets may be closer in behavior to the global classifier produced from the entire training set than those trained on random class distributions. In addition, disjoint data subsets promote the maximum amount of parallelism and hence are more desirable. Yet partial replication [8] may mitigate the problem of extreme bias potentially introduced by disjoint data.
Strategies: There are indeed many strategies for arbitration and combining as detailed here, each impacting the size of training data required to implement them effectively. Several experiments were run to determine the relative effectiveness of some of these strategies. They vary in the type of information or biased distributions

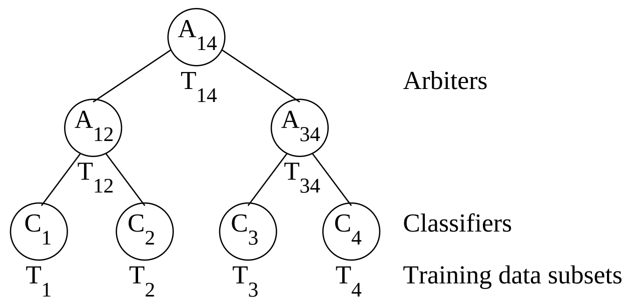

Figure 3. Sample arbiter tree.

of training data the arbiter is allowed to see. Thus far, the meta-learning strategies we discussed are applied solely to a single collection of base classifiers. (These are called "one-level" meta-learners.) We also studied building hierarchical structures in a recursive fashion, i.e., meta-learning arbiters and combiners from a collection of "lower level" arbiters and combiners. These hierarchical classifiers attempt to improve the prediction accuracy that may be achieved by one-level meta-learned classifiers.

5. Hierarchical Meta-learning

The *one-level* meta-learning learning techniques may not produce highly accurate classifiers. Here, we explore hierarchical techniques by applying meta-learning strategies recursively.

5.1. Arbiter Trees

An *arbiter tree* is a hierarchical structure composed of arbiters that are computed in a bottom-up, binary-tree fashion. (The choice of a binary tree is to simplify our discussion. Higher order trees are also studied.) An arbiter is initially learned from the output of a pair of base classifiers and recursively, an arbiter is learned from the output of two arbiters. For k subsets and k classifiers, there are $log_2(k)$ levels generated.

When an instance is classified by the arbiter tree, predictions flow from the leaves to the root. First, each of the leaf classifiers produces an initial classification of the test instance. From a pair of predictions and the parent arbiter's prediction, another prediction is produced by an arbitration rule. This process is applied at each level until a final prediction is produced at the root of the tree. We now proceed to describe how to build an arbiter tree in detail.

Suppose there are initially four training data subsets ($T_1 - T_4$), processed by some learning algorithm, L. First, four classifiers ($C_1 - C_4$) are generated from four instances of L applied to $T_1 - T_4$. The union of the subsets T_1 and T_2, U_{12}, is then classified by C_1 and C_2, which generates two sets of predictions, P_1 and P_2. A *selection rule* as detailed earlier generates a training set (T_{12}) for the arbiter from the predictions P_1 and P_2, and the subset U_{12}. The arbiter (A_{12}) is then trained from the set T_{12} by algorithm L. Similarly, arbiter A_{34} is generated from T_3 and T_4 and hence all the first-level arbiters are produced. Then U_{14} is formed by the union of subsets T_1 through T_4 and is classified by the arbiter trees rooted with A_{12} and A_{34}. Similarly, T_{14} and A_{14} (root arbiter) are generated and the arbiter tree is complete. The resultant tree is depicted in Figure 3.

This process can be generalized to arbiter trees of higher order. The higher the order is, the shallower the tree becomes. In a parallel environment this translates to faster execution. However, there will logically be an increase in the number of disagreements (and hence data items selected for training) and higher communication overhead at each level in the tree due to the arbitration of many more predictions at a single arbitration site.

We note with interest that in a distributed computing environment, the union sets need not be formed at one processing site. Rather, we can classify each subset by transmitting each learned classifier to each site which is used to scan the local data set that is labeled with the classifier's predictions. Each classifier is a computational object far smaller in size than the training sets from which they are derived. For example, in a network computing environment each classifier may be encapsulated as an "agent" that is communicated among sites.

We experimented with several different arbiter strategies besides the one described in Section 4.1. (The entire set of results we obtained for all the various strategies are reported in [7].) Next, we discuss the computational efficiency of the various strategies we explored.

5.1.1. Discussion

Since an arbiter training set is constructed from the results of the arbiter's two subtrees, each node in the arbiter tree is a synchronization point. That is, arbitrary subtrees can be run asynchronously with no communication until a pair of subtrees join at the same parent. The time to learn an arbiter tree is proportional to the longest path in the tree, which is bounded by the path with the most training data. To reduce the complexity of learning arbiter trees, the size of the training sets for arbiters is purposefully restricted to be no larger than the training sets used to compute base classifiers. Thus, the parallel processing time at each level of the tree is relatively equal throughout the tree. However, in several of our experiments, this restriction on the allowable size of the training sets for arbiters was removed to explore two key issues: whether higher accuracy could be achieved by providing more information for each arbiter, and what might be the number of disagreements so generated, and hence the size of training data that would naturally be formed by our selection rules.

Notice that the maximum training set size doubles as one moves up one level in the tree and is equal to the size of the entire training set when the root is reached. Obviously, we do not desire forming a training set at the root as large as the original training set. Indeed, meta-learning in this case is of no use, and at great expense. Therefore, we desire a means to control the size of the arbiter training sets as we move up the tree without a significant reduction in accuracy of the final result.

Since the training sets selected at an arbiter node depends on the classification results from the two descendant subtrees during run time, the configuration of an arbiter tree cannot be optimized during compile time. The size of these sets (i.e., the number of disagreements) is not known until the base classifiers are first computed. However, we may optimize the configuration of a tree during run time by clever *pairing* of classifiers. The configuration of the resulting tree depends upon the manner in which the classifiers and arbiters are paired and ordered at each level. Our goal here is to devise a *pairing strategy* that favors smaller training sets near the root.

One strategy we may consider is to pair the classifiers and arbiters at each level that would produce the fewest disagreements and hence the smallest arbiter training sets (denoted as *min-size*). Another possible strategy is to pair those classifiers that produce the highest number of disagreements (*max-size*). At first glance the first strategy would seem to be more attractive. However, if the disagreements between classifiers are not resolved at the bottom of the tree, the data that are not commonly classified will surface near the root of the tree, which is also where there are fewer choices of pairings of classifiers to control the growth of the training sets. Hence, it may be advantageous to resolve the disagreements near the leaves producing fewer disagreements near the root. That is, it may be more desirable to pair classifiers and arbiters that produce the largest sets lower in the tree, which is perhaps counterintuitive. These sophisticated pairing schemes might decrease the arbiter training set size, but they might also increase the communication overhead in a distributed computing environment. They also create synchronization points at each level, instead of at each node when no special pairings are performed. A compromise strategy might be to perform pairing only at the leaf level. This indirectly affects the subsequent training sets at each level, but synchronization occurs only at each node and not at each level.

5.2. Combiner Trees

The way combiner trees are learned and used is very similar to arbiter trees. A combiner tree is trained bottom-up. A combiner, instead of an arbiter, is computed at each non-leaf node of a combiner tree. To simplify our discussion here, we describe how a binary combiner tree is used and trained. (Our experiments reported later included higher order trees as well.)

To classify an instance, each of the leaf classifiers produces an initial prediction. From a pair of predictions, the composition rule is used to generate a meta-level instance, which is then classified by the parent combiner. This process is applied at each level until a final prediction is produced at the root of the tree.

Another significant departure from arbiter trees is that for combiner trees, a random set of examples (a *validation set*) is selected at each level of learning in generating a combiner tree instead of choosing a set from the union of the underlying data subsets. Before learning commences, a random set of examples is picked from the underlying subsets for each level of the combiner tree. To ensure efficient processing, the size of these random training sets is limited to the size of the initial subsets used to train base classifiers. Base classifiers are learned at the leaf level from disjoint training data. Each pair of base classifiers produce predictions for the random training set at the first level. Following the composition rule, a meta-level training set is generated from the predictions and training examples. A combiner is then learned from the meta-level training set by applying a learning algorithm. This process is repeated at each level until the root combiner is created. Again, in a network computing environment classifiers may be represented as remote agent processes to distribute the meta-learning process.

The arbiter and combiner tree strategies have different impact on efficiency. The arbiter tree approach we have implemented requires the classification of, possibly, the entire data set at the root level. Significant speed up might not be easily obtained. The combiner tree approach, however, always classifies a set of data that is bounded by the size of a relatively small validation set. Therefore, combiner trees can be generated more efficiently than arbiter trees. However, it remains to be seen what impact on accuracy either scheme may exhibit.

A large number of experiments were conducted to evaluate these and several other meta-learning strategies varying the particular learning algorithms and distribution schemes over three learning tasks. Our results are reported next.

6. Experimental Results and Evaluation

One of the more common techniques used in evaluating the accuracy of a learning program is *cross-validation* [2]. In this technique, the entire data set is divided into a training set and a disjoint test set. Classifiers are computed only from the training set and are evaluated only against the test set. This process is repeated n times, in each case using entirely different training and test sets. The accuracies of the n different classifiers measured over the n different test sets are then averaged as the final prediction accuracy for the learning algorithm employed.

The learning algorithms and the learning tasks we evaluate here by cross validation are detailed in the following pages. In most of the experimental results reported below, *the average from 10-fold cross validation runs is plotted.* This represents hundreds of experimental runs over the various meta-learning strategies in-toto. Also, statistical significance in difference of averages is measured by using the one-sided *t-test* with a 90% confidence value.

6.1. Learning Algorithms

Four inductive learning algorithms are used in our experiments. We obtained ID3 [16] and CART [2] as part of the IND package [3] from NASA Ames Research Center; both algorithms compute decision trees. WPEBLS is the weighted version of PEBLS [10], which is a nearest-neighbor learning algorithm. BAYES is a Bayesian classifier that is based on computing conditional probabilities as described in [9]. The latter two algorithms were reimplemented in C.

6.2. Learning Tasks

Two molecular biology sequence analysis data sets, obtained from the UCI Machine Learning Database, were used in our studies.

The DNA splice junctions (SJ) data set [19], courtesy of Towell, Shavlik and Noordewier, contains 3,190 sequences of nucleotides and the type of splice junction, if any, at the center of each sequence (three classes). Each sequence has 60 nucleotides with eight different values each (four base ones plus four combinations).

The protein coding regions (PCR) data set [11], courtesy of Craven and Shavlik, contains 20,000 DNA nucleotide sequences and their binary classifications (coding or non-coding). Each sequence has 15 nucleotides with four different values each.

The two data sets chosen in our experiments represent two different kinds of data sets: one is difficult to learn (PCR at 70+%) and the other is easy to learn (SJ at 90+%).

Although these are not very large data sets, they do provide us with an idea of how our strategies behave in practice. Since the data sets are sufficiently small, we are able to generate base line statistics on the accuracy of each learning algorithm we have chosen to use in this study. Otherwise, using a massive database would imply that we have unbounded resources and time in order to compute baseline statistics. As we have noted (as well as [4]) this might take many years of computing. Furthermore, scaling studies are possible on these smaller sets simply by varying the number and size of the subsets formed in the initial data reduction schemes and extrapolating. However, larger data sets are being sought for use in this study which will be the focus of our work after we have exhausted the experiments possible on the smaller test cases. Stated another way, if we cannot display useful and interesting results of meta-learning on these small test cases, then there would not be much point in writing this paper in the first place.

6.3. Voting and Meta-learning

We first consider whether meta-learning performs as well as the common voting and Bayesian techniques reported in the literature. In our experiments, we varied the number of equi-sized subsets of training data from 2 to 64 ensuring each was disjoint but with proportional distribution of examples of each class. The size of a validation

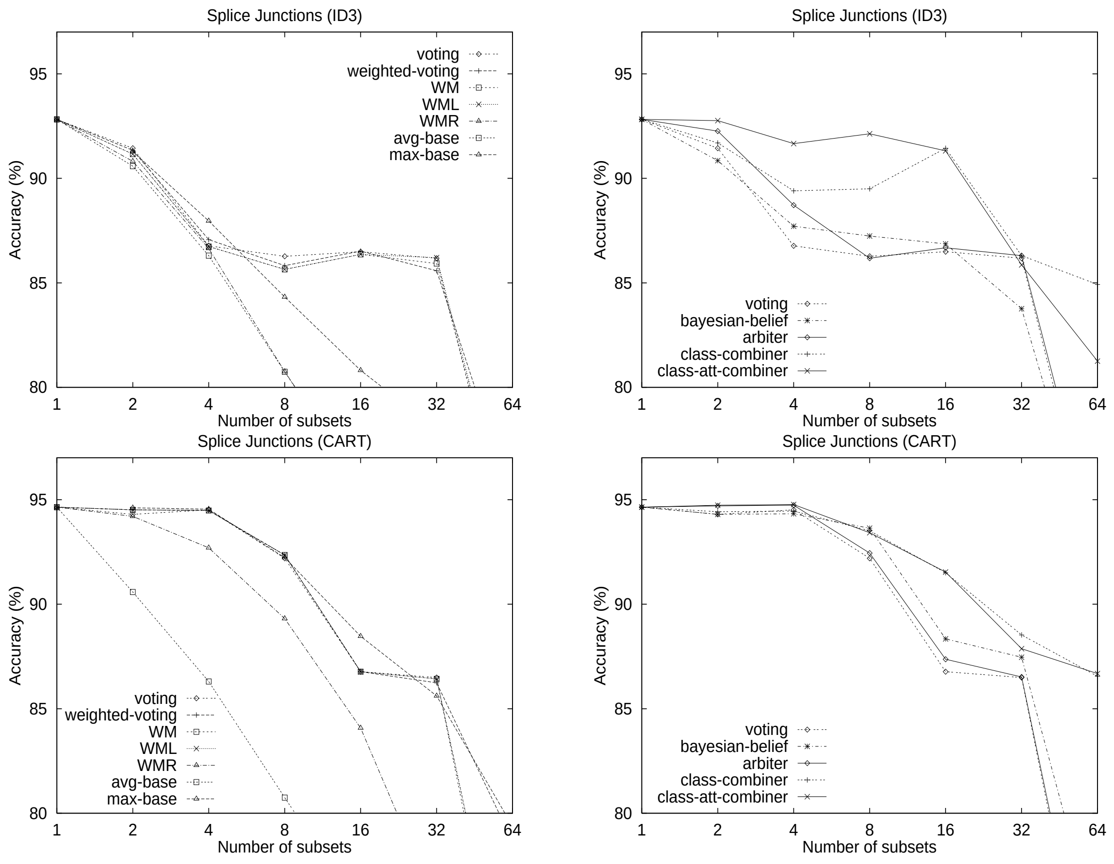

Figure 4. Accuracy for the one-level integrating techniques in the splice junctions domain.

set used for generating the *integrating structures* (weights/probabilities/arbiters/combiners) is twice the size of the underlying training set for a base classifier. The prediction accuracy on a separate test set is our primary comparison measure. The different strategies were run on the two data sets with the two learning algorithms. The results from the splice junctions data set are plotted in Figures4 and the protein coding regions data set in Figure 5. In each figure the first row of graphs depicts results from the different integrating techniques using ID3 and the second row using CART. The accuracy for the global classifier is plotted as "one subset," meaning the learning algorithms was applied to the entire training set to produce the baseline accuracy results for comparison. The average accuracy of the base classifiers for each number of subsets is also plotted, labeled as "avg-base." By way of comparison, the average accuracy of the most accurate base classifiers is plotted as "max-base." The plotted accuracy is the average of 10-fold cross-validation runs.

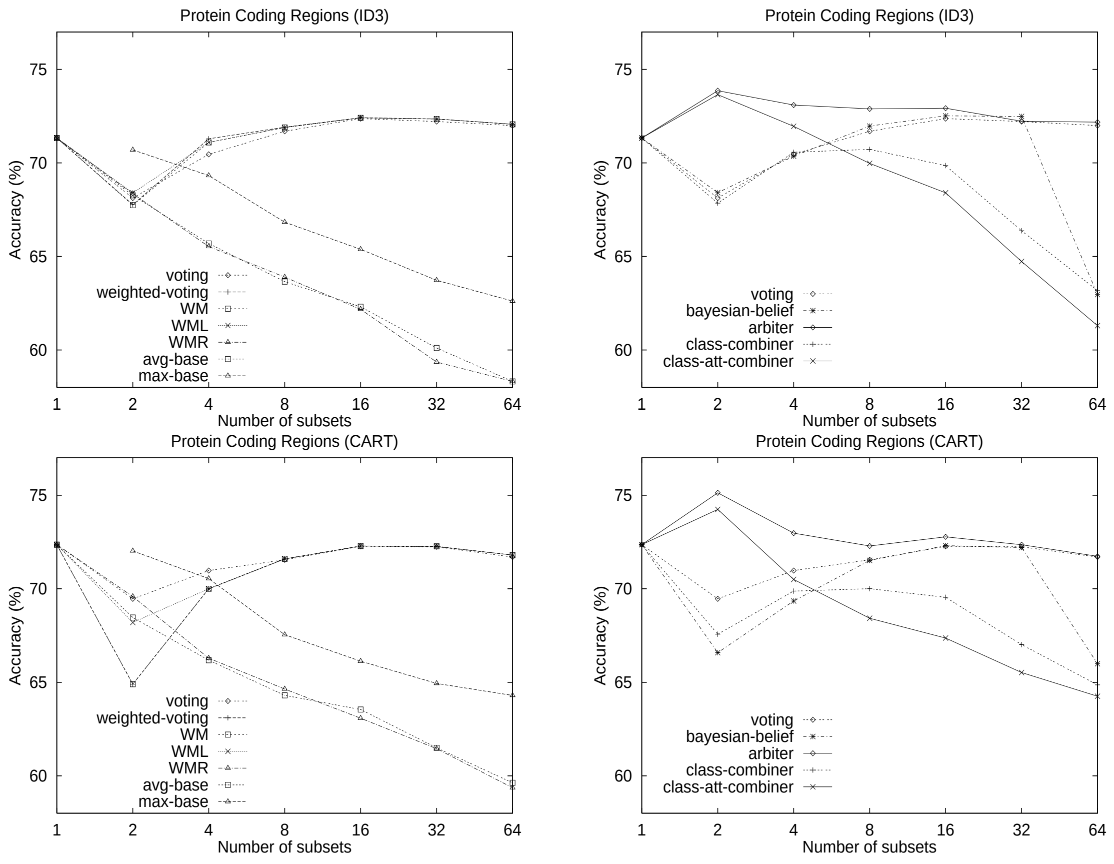

Figure 5. Accuracy for the one-level integrating techniques in the protein coding regions domain.

Experiments run over the splice junctions data set indicate that all the methods sustain a drop in accuracy when the number of subsets increases (i.e., the size of each distinct subset of training data decreases). For either algorithm, the *class-combiner* and *class-attribute-combiner* schemes exhibit higher accuracy than all the other techniques. The difference is statistically significant for ID3 with most subset sizes and for CART with a few subset sizes. At 64 subsets, with $\sim$ 45 examples each, while the other methods sustain significantly more than 10% in accuracy degradation, the *combiner* methods incur around 10% or less decrease in accuracy. The *weighted-majority-random* method performs the worst and significantly worse than the others.

For the protein coding regions data set, only the *arbiter* scheme can maintain, and sometimes exceeds, the original accuracy level. Most other techniques suffer a significant drop in accuracy for 2 subsets and climb back to the original accuracy level when the number of subsets increases. Again, the *weighted-majority-random* method performs much worse than the others.

In general all the methods, except the *weighted-majority-random* scheme, considerably outperform the average base classifier ("avg-base"). The gap is statistically significant. Furthermore, they outperform the average most accurate base classifier ("max-base") except with CART in the splice junction domain. That is, random sampling of the training data is definitely not sufficient to generate accurate classifiers in the two data sets we studied. Hence, combining techniques are necessary.

The results of our experiments indicate that the meta-learning strategies dominate over the weighted voting techniques across domains and learners used in this study. However, the meta-learning techniques do not always outperform the weighted voting schemes. In the SJ domain, the *combiner* techniques are more favorable while in the PCR domain the *arbiter* technique is. It is not clear under what circumstances a particular meta-learning strategy will perform better. Additional studies are underway in an attempt to gain an understanding of these circumstances.

As we observe in the SJ domain, none of the schemes can maintain the baseline accuracy when the number of subsets increases. All the techniques presented so far can be characterized as *one-level* methods. They only perform one level of processing to generate the integrating structures. We next consider the behavior of hierarchical meta-learning structures.

6.4. Arbiter Trees

Here we first examine the results from bounded arbiter training sets and arbiter trees of different orders (from binary trees up to 8-ary trees). This is followed by our results achieved in the case that arbiter training sets are unbounded under different pairing strategies.

6.4.1. Order of the arbiter trees and training set size limit

We performed experiments on the splice junctions and protein coding regions data to evaluate the arbiter tree approach. Again, we varied the number of subsets from 2 to 64 and measured the prediction accuracy on a disjoint test set. The plotted results in Figure 6 are averages from 10-fold cross-validation runs.

We varied the order of the arbiter trees from two to eight. For the SJ data set the plots display a drop in accuracy when the number of subsets increases. Also, the higher order trees are generally less accurate than the lower ones. However, in the PCR data set experiments the accuracy is maintained, or exceeded in some circumstances, regardless of the order of the trees.

Recall that at each tree level, the size of the arbiter training set is fixed to the size of a data subset used in training the base classifiers. If we relax the restriction on the size of the data set for training an arbiter, we might expect an improvement in accuracy at the expense in processing time. To test this hypothesis, a set of

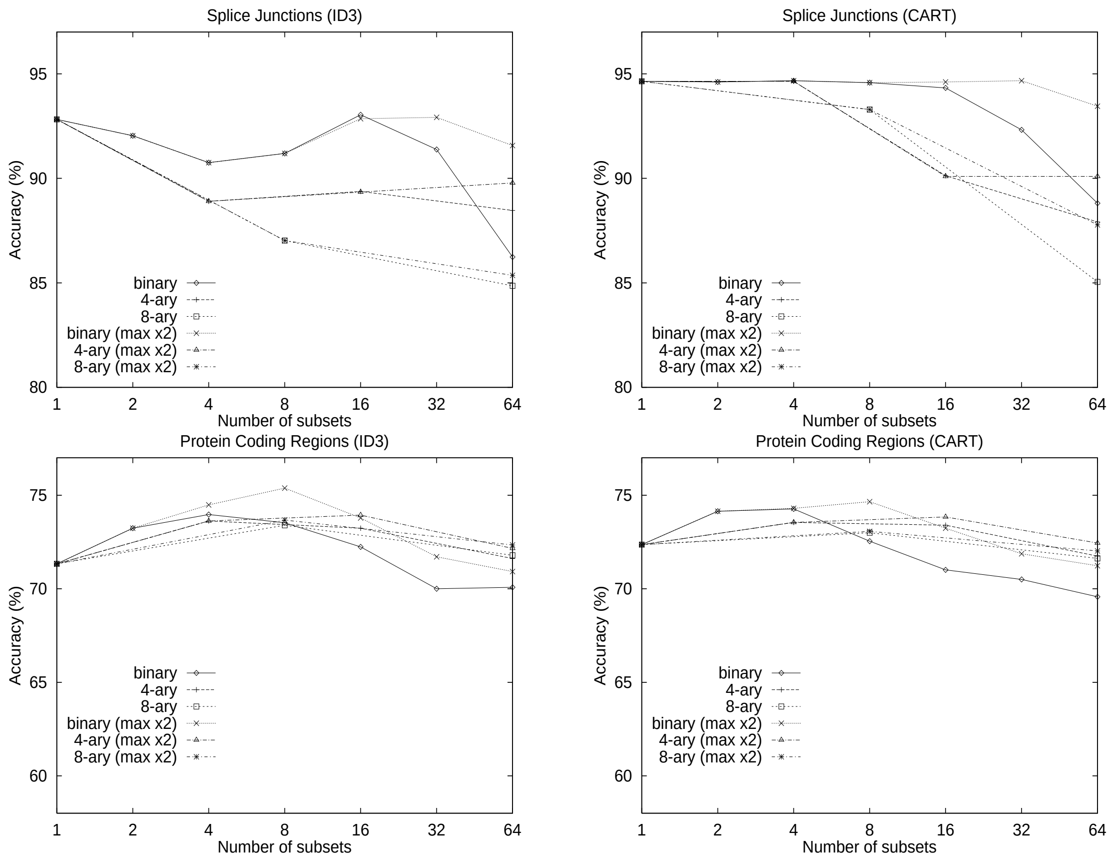

Figure 6. Accuracy for the arbiter tree techniques.

experiments was performed to double the maximum training set size for the arbiters. As we observe in Figure 6, by doubling the arbiter training set size, the original accuracy is roughly maintained by the binary trees in the SJ domain, regardless of the learner. For 4-ary and 8-ary trees, the accuracy results show no significant improvement. However, this multi-level arbiter tree approach does demonstrate an accuracy improvement over the *one-level* techniques, which generally cannot maintain the accuracy obtained from the whole data set in our experiments.

6.4.2. Largest arbiter training set size and classifier pairing

As we observe from Figure 7, when the restriction on the size of the training set for an arbiter is lifted, the same level of accuracy can be achieved. For the SJ data set, empirical results show that the single largest arbiter training set exhibited in the

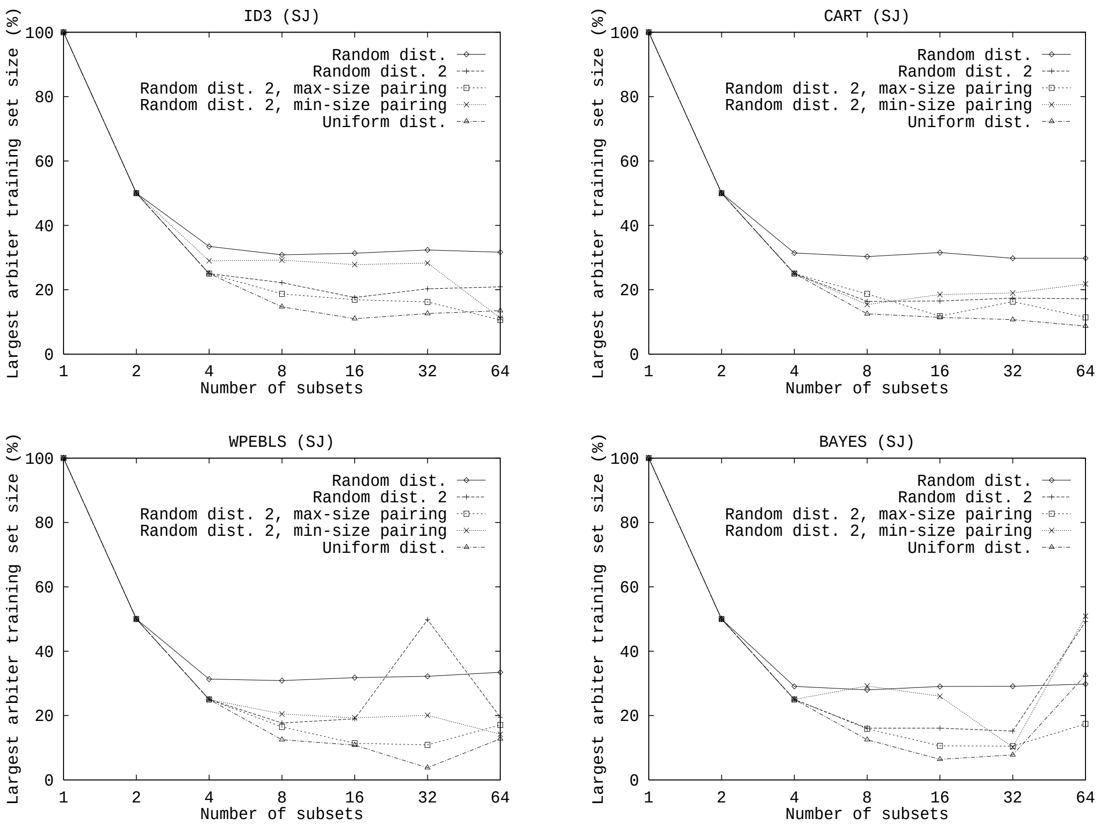

Figure 7. Arbiter training set size with different class distributions and pairing strategies

tree is about *30% of the entire training set.* This is a significant result. Less time and memory than a serial version is needed to reach the same accuracy under arbitration demonstrating significant scaling properties. The percentage of total training data exhibited in the tree is dependent on several factors: the prediction accuracy of the algorithm on the given data set, the distribution of the data in the subsets, and the pairing of learned classifiers and arbiters at each level. As we mentioned in Section 4.3, the pairing of classifiers and arbiters affects the arbiter training set sizes. Several experiments were performed on the two pairing strategies (*max-size* and *min-size*) applied only at the leaf level and the results are displayed in Figure 7. All these experiments were conducted on the SJ data set. The initial random class distribution, a uniform class distribution and a second random class distribution were used in training the base classifiers. The second random distribution was composed in such a way to ensure that no half of the learned arbiter tree was ignorant of one of the classes, as was the case in the initial random distribution. Different pairing strategies were used on the uniform distribution and the second random distribution. As shown in Figure 7, the uniform distribution achieved smaller training sets than the other two random distributions. The largest training set size in this case was approximately *only 10% of the total available data* when the number of subsets was larger than eight, except for BAYES with 64 subsets (BAYES seemed to be not able to gather enough statistics on small subsets). (Note that when the number of subsets is eight or fewer, the training sets for the leaf classifiers are larger than 10% of the original data set and become the largest in the arbiter tree.)

The two pairing strategies did not affect the sizes for the uniform distribution and are not shown in the figure. One possible explanation is that the uniform distribution produced the smallest training sets possible and the pairing strategies did not matter. However, the *max-size* pairing strategy did generally reduce the sizes of the training subsets for the second random distribution. The *min-size* pairing strategy, on the other hand, did not affect, or sometimes even increased, the sizes of the generated subsets. In summary, uniform class distribution tends to produce the smallest training sets and the *max-size* pairing strategy can reduce the size of the subsets in random class distributions.

6.5. Combiner Trees

Here we consider the accuracy of combiner trees. In our experiments, we varied the number of equi-sized subsets of training data from 2 to 64 ensuring each was disjoint but with proportional distribution of examples of each class. We also varied the order of the combiner trees from two to eight. The results of our experiments on the combiner trees (under two different training strategies) are displayed in Figure 8 and 9. The baseline accuracy for comparative evaluation is plotted as "one subset," meaning the learning algorithms were applied to the entire training set in-toto to produce the global classifier. The plots are derived from the average of 10-fold cross-validation runs.

Results from the *class-combiner* tree strategy displayed in Figure 8 show a drop in accuracy in both data sets in most cases, compared to the global classifier, when the number of subsets increases. The drop varies from 3% to 15%. (The percentage decrease in the amount of data in each training subset is far larger!) The binary combiner trees seems to be less accurate than higher order trees in this case. This might be due to the lack of information for finding correlations among only two sets of predictions. As in the experiments for arbiter trees, we doubled the size of meta-level training sets. Statistically significant improvements were observed in the SJ data set with CART as the learner.

In another experiment using the *class-attribute-combiner* tree strategy, Figure 9 suggests that the binary trees appear to maintain the accuracy of the global classifier except in the splice junctions data set with CART as the learner. Higher-order trees were generally less accurate.

We note with interest that doubling the size of the training sets for combiners improved accuracy significantly. For the protein coding regions data set, the accu-

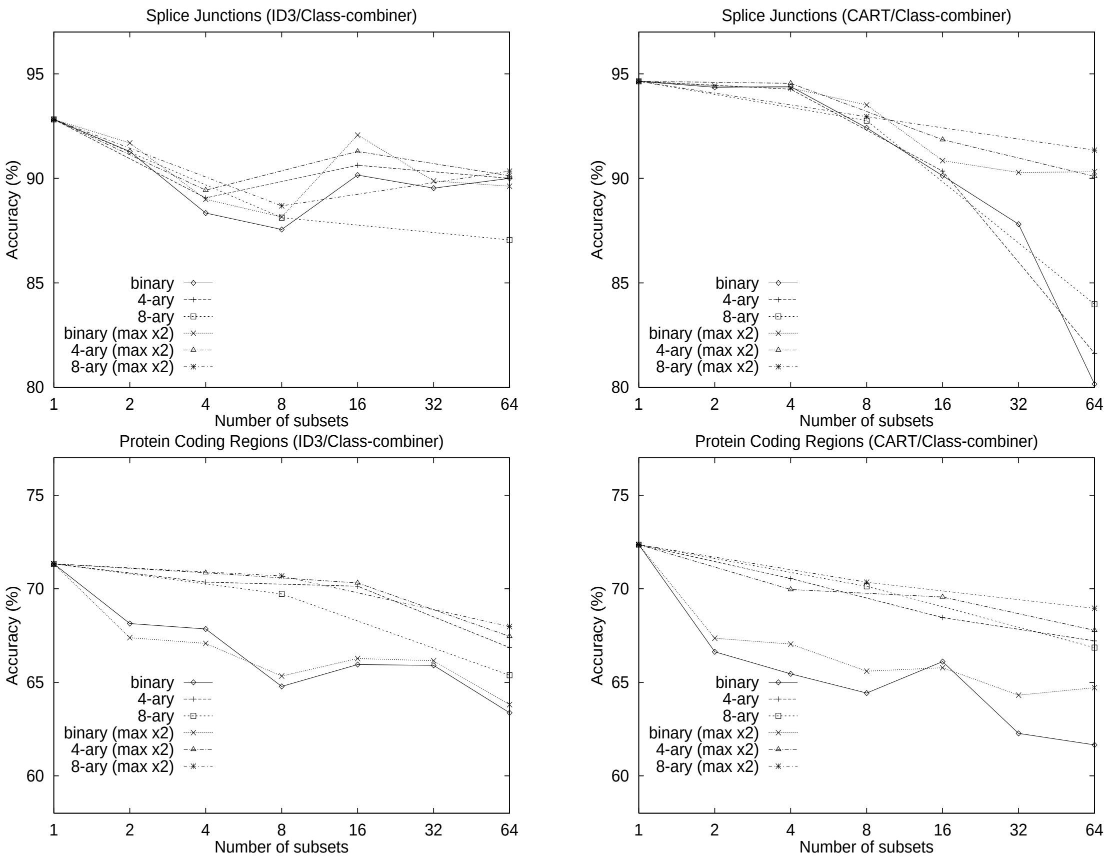

Figure 8. Accuracy for the class-combiner tree techniques.

racy of the binary trees was consistently higher than that from the global classifier; i.e., *this meta-learning strategy has demonstrated a means of boosting accuracy of a single classifier* trained on the entire data set. The improvement is statistically significant. This is a particularly interesting finding since the information loss due to data partitioning was more than recovered by the combiner tree. Thus, this scheme demonstrates a means of integrating the collective knowledge distributed among the individual base classifiers.

In summary, our experimental results suggest that increasing the size of the meta-level training sets improves the accuracy of the learned trees, a likely result from the simple observation that more data is available for training leading to better information about the correlation among base classifiers. The experimental data convincingly demonstrate that doubling the training set size of the meta-level partitions relative to the underlying subsets used in training base classifiers is sufficient

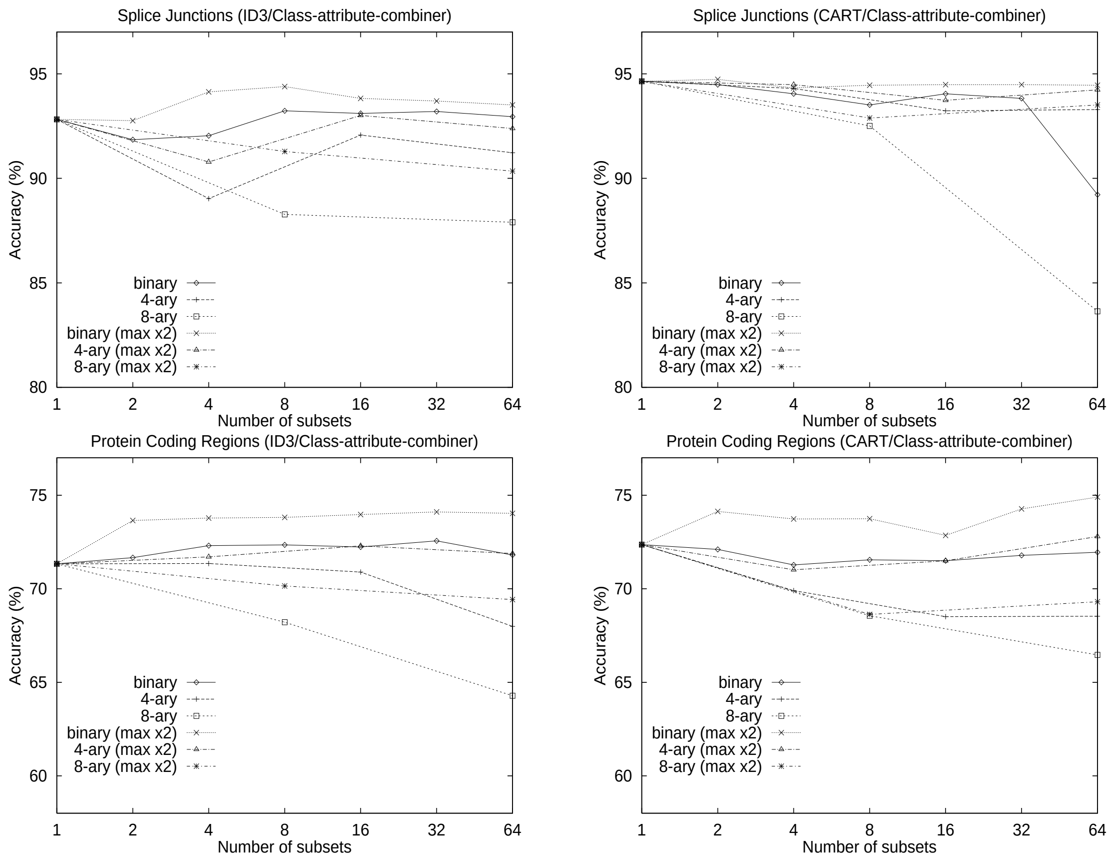

Figure 9. Accuracy for the class-attribute-combiner tree techniques.

to maintain the same level of accuracy as the global classifier, and indeed may boost accuracy as well.

7. Concluding Remarks

By way of summary, we have demonstrated several interesting behaviors of the various meta-learning architectures studied to date. Our results are based on many empirical experiments using all the different combinations of a small number of data sets and learning algorithms.

- The meta-learning strategies do show a consistent improvement in classification accuracy over any of the base classifiers trained on a subsets of available training data. Our studies show that classifiers trained individually from random subsets of a large data set are not as accurate as integrating a collection of separately learned classifiers.

- The meta-learning strategies can outperform the other more common *one-level* voting-based or Bayesian techniques. In the learning tasks and domains we studied, the one-level meta-learning schemes do not consistently maintain high accuracy as the number of subsets increases (and the amount of available data thus decreases). However, the results show that the hierarchical meta-learning approach is able to sustain the same level of accuracy as a global classifier trained on the entire data set distributed among a number of sites.

- Under the arbiter tree strategy allowing unbounded meta-level training sets, we determined that, over the variety of algorithms employed, at most 30%, and in certain cases at most 10%, of the entire training data was required at any one processing site to maintain the equivalent predictive accuracy of a single global classifier computed from all available data. In other words, with the arbiter tree strategy, a site can process a larger learning task (at least 3 times in the domain we studied) without increasing memory resources.

- Unbounded meta-level training sets are not necessary to achieve good results. Limiting the meta-level training set size to twice the size of the data subsets used to compute base classifiers usually yielded a system able to maintain the same level of accuracy achieved by the global classifier. This is important from a complexity perspective.

- Combiner and arbiter trees of lower order perform better than ones with higher order. This seems mainly attributed to the increase in the number of opportunities in correcting the base classifiers since there are more levels in the lower order trees to filter and compose good training data.

- Finally, the combiner tree strategy was demonstrated to consistently boost the predictive accuracy of a global classifier under certain circumstances. This suggests that a properly configured meta-learning strategy combining multiple knowledge sources provides a more accurate view of all available data than any one learning algorithm alone can achieve.

We believe the concepts embodied by the term meta-learning proposed here provide an important first step in understanding and developing systems that learn from massive widely dispersed databases, and that scale. Meta-learning architectures may provide the means of using large numbers of low cost networked computers who collectively learn from massive databases useful and important new knowledge, that would otherwise be prohibitively expensive, in cost and time, to achieve. We believe meta-learning systems will be an important contributing technology if the future infrastructures envisioned for the Intelligent Integration of Information Systems is to be realized.

Acknowledgments

This work was performed at Columbia University and was partially supported by grants from NSF (IRI-94-13847 & CDA-90-24735), New York State Science and Technology Foundation, and Citicorp. We thank David Wolpert for many useful discussions about this work.

References

1. K. Ali and M. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 1996. to appear.
2. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.
3. W. Buntine and R. Caruana. *Introduction to IND and Recursive Partitioning*. NASA Ames Research Center, 1991.
4. J. Catlett. Megainduction: A test flight. In *Proc. Eighth Intl. Work. Machine Learning*, pages 596–599, 1991.
5. P. Chan and S. Stolfo. Experiments on multistrategy learning by meta-learning. In *Proc. Second Intl. Conf. Info. Know. Manag.*, pages 314–323, 1993.
6. P. Chan and S. Stolfo. Meta-learning for multistrategy and parallel learning. In *Proc. Second Intl. Work. on Multistrategy Learning*, pages 150–165, 1993.
7. P. Chan and S. Stolfo. Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Know. Disc. Databases*, pages 227–240, 1993.
8. P. Chan and S. Stolfo. Scaling learning by meta-learning over disjoint and partially replicated data. In *Proc. Ninth Florida AI Research Symposium*, pages 151–155, 1996.
9. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–285, 1989.
10. S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.
11. M. Craven and J. Shavlik. Learning to represent codons: A challenge problem for constructive induction. In *Proc. IJCAI-93*, pages 1319–1324, 1993.
12. N. Flann and T. Dietterich. A study of explanation-based mehtods for inductive learning. *Machine Learning*, 4:187–266, 1989.
13. A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Info. Proc. Sys. 7*, pages 231–238. MIT Press, 1995.
14. N. Littlestone and M. Warmuth. The weighted majority algorithm. Technical Report UCSC-CRL-89-16, Univ. Cal., Santa Cruz, 1989.
15. T. M. Mitchell. The need for biases in learning generalizaions. Technical Report CBM-TR-117, Dept. Comp. Sci., Rutgers Univ., 1980.
16. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
17. R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–226, 1990.
18. S. Stolfo, Z. Galil, K. McKeown, and R. Mills. Speech recognition in parallel. In *Proc. Speech Nat. Lang. Work.*, pages 353–373. DARPA, 1989.
19. G. Towell, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proc. AAAI-90*, pages 861–866, 1990.
20. L. Valiant. A theory of the learnable. *Comm. ACM*, 27:1134–1142, 1984.
21. D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
22. L. Xu, A. Krzyzak, and C. Suen. Methods of combining multiple classifires and their applications to handwriting recognition. *IEEE Trans. Sys. Man. Cyb.*, 22:418–435, 1992.
23. X. Zhang, J. Mesirov, and D. Waltz. A hybrid system for protein secondary structure prediction. *J. Mol. Biol.*, 225:1049–1063, 1992.