Toward Parallel and Distributed Learning by Meta-Learning

Philip K. Chan and Salvatore J. Stolfo

Department of Computer Science
Columbia University
New York, NY 10027
pkc@cs.columbia.edu and sal@cs.columbia.edu

Abstract

Much of the research in inductive learning concentrates on problems with relatively small amounts of data. With the coming age of very large network computing, it is likely that orders of magnitude more data in databases will be available for various learning problems of real world importance. Learning techniques are central to knowledge discovery and the approach proposed in this paper may substantially increase the amount of data a knowledge discovery system can handle effectively. *Meta-learning* is proposed as a general technique to integrating a number of distinct learning processes. This paper details several meta-learning strategies for integrating independently learned classifiers by the same learner in a parallel and distributed computing environment. Our strategies are particularly suited for massive amounts of data that main-memory-based learning algorithms cannot efficiently handle. The strategies are also independent of the particular learning algorithm used and the underlying parallel and distributed platform. Preliminary experiments using different data sets and algorithms demonstrate encouraging results: parallel learning by meta-learning can achieve comparable prediction accuracy in less space and time than purely serial learning.

Keywords: machine learning, inductive learning, meta-learning, parallel and distributed processing, and large databases.

1 Introduction

Much of the research in inductive learning concentrates on problems with relatively small amounts of data. With the coming age of very large network computing, it is likely that orders of magnitude more data in databases will be available for various learning problems of real world importance. The Grand Challenges of HPCC [20] are perhaps the best examples. Learning techniques are central to knowledge discovery [11] and the approach proposed here may substantially increase the amount of data a Knowledge Discovery system can handle effectively.

Quinlan [14] approached the problem of efficiently applying learning systems to data that are substantially larger than available main memory with a windowing technique. A learning algorithm is applied to a small subset of training data, called a *window*, and the learned concept is tested on the remaining training data. This is repeated on a new window of the same size with some of the incorrectly classified data replacing some of the data in the old window until all the data are correctly classified. Wirth and Catlett [21] show that the windowing technique does not significantly improve speed on reliable data. On the contrary, for noisy data, windowing considerably slows down the computation. Catlett [3] demonstrates that larger amounts of data improves accuracy, but he projects that ID3 [15] on modern machines will take several months to learn from a million records in the flight data set obtained from NASA. He proposes some improvements to the ID3 algorithm particularly for handling attributes with real numbers, but the processing time is still prohibitive due to the algorithm's complexity. In addition, typical learning systems like ID3 are not designed to handle data that exceed the size of a monolithic memory on a single processor. Although most modern operating systems support virtual memory, the application of complex algorithms like ID3 to the large amount of disk-resident data we realistically assume can result in intolerable amount of I/O or even thrashing of external disk storage. Clearly, parallel and distributed processing provides the best hope of dealing with such large amounts of data.

One approach to this problem is to parallelize the learning algorithms and apply the parallelized algorithm to the entire data set (presumably utilizing multiple I/O channels to handle the I/O bottleneck). Zhang et al.'s work [23] on parallelizing the backpropagation algorithm on a Connection Machine is one example. This approach requires optimizing the code for a particular algorithm on a specific architecture. Another approach which we propose in this paper is to run the serial code on a number of data subsets in parallel and combine the results in an intelligent fashion thus reducing and limiting the amount of data inspected by any one learning process. This approach has the advantage of using the same serial code without the time-consuming process of parallelizing it. Since the framework for combining the results of learned concepts is independent of the learning algorithms, it can be used with different learners. In addition, this approach is independent of the computing platform used. However, this approach cannot guarantee the accuracy of the learned concepts to be the same as the serial version since clearly a considerable amount of information is not accessible to each of the learning processes. Furthermore, because of the proliferation of networks of workstations and distributed databases, our approach of not relying on specific parallel and distributed environment is particularly attractive.

In this paper we introduce the concept of *meta-learning* and its use in combining results from a set of parallel or distributed learning processes. Section 2 discusses meta-learning and how it facilitates parallel and distributed learning. Section 3 details our strategies for parallel learning by meta-learning. Section 4 discusses our preliminary experiments and Section 5 presents the results. Section 6 discusses our findings and work in progress. Section 7 concludes with a summary of this study.

2 Meta-learning

Meta-learning can be loosely defined as learning from information generated by a learner(s). It can also be viewed as the learning of meta-knowledge on the learned information. In our work we concentrate on learning from the output of inductive learning (or learning-from-examples) systems. Meta-learning, in this case, means learning from the *classifiers* produced by the learners and the *predictions* of these classifiers on *training data*. A classifier (or concept) is the output of an inductive learning system and a prediction (or classification) is the predicted class generated by a classifier when an instance is supplied. That is, we are interested in the output of the learners, not the learners themselves. Moreover, the training data presented to the learners initially are also available to the *meta-learner* if warranted.

Meta-learning is a general technique to coalesce the results of multiple learners. In this paper we concentrate on using meta-learning to combine parallel learning processes for higher speed and to maintain the prediction accuracy that would be achieved by the sequential version. This involves applying the same algorithm on different subsets of the data in parallel and the use of meta-learning to combine the partial results. We are not aware of any work in the literature on this approach beyond what was first reported in [18] in the domain of speech recognition. Work on using meta-learning for combining different learning systems is reported elsewhere [4, 6] and is further discussed at the end of this paper. In the next section we will discuss our approach on the how to use meta-learning for parallel learning using only one learning algorithm.

3 Parallel Learning

The objective here is to speed up the learning process by *divide-and-conquer*. The data set is partitioned into subsets and the same learning algorithm is applied on each of these subsets. Several issues arise here.

First, how many subsets should be generated? This largely depends on the number of processors available and the size of the training set. The number of processors puts an upper bound on the number of subsets. Another consideration is the desired accuracy we wish to achieve. As we will see in our experiments, there may be a tradeoff between the number of subsets and the final accuracy. Moreover, the size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective classifier. We varied the number of subsets from 2 to 64 in our experiments.

Second, what is the distribution of training examples in the subsets? The subsets can be disjoint or overlap. The class distribution can be random, or follow some deterministic scheme. We experimented with disjoint equal-size subsets with random distributions of classes. Disjoint subsets implies no data is shared between learning processes and thus no communication overhead is paid during training in a parallel execution environment.

Third, what is the strategy to coalesce the partial results generated by the learning processes? This is the more important question. The simplest approach is to allow the separate learners to vote and use the prediction with the most votes as the classification. Our approach is meta-learning *arbiters* in a bottom-up binary-tree fashion. (The choice of a binary tree is discussed later.)

An arbiter, together with an *arbitration rule*, decide a final classification outcome based upon a number of candidate predictions. An arbiter is learned from the output of a pair of learning processes and recursively, an arbiter is learned from the output of two arbiters. A binary tree of arbiters (called an *arbiter tree*) is generated with the initially learned classifiers at the leaves. (The arbiters themselves are essentially classifiers.) For s subsets and s classifiers, there are $log_2(s)$ levels in the generated arbiter tree. The manner in which arbiters are computed and used is the subject of the following sections.

3.1 Classifying using an arbiter tree

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 prediction; i.e., a classification of the test instance. From a pair of predictions and the parent arbiter's prediction, a combined prediction is produced by some *arbitration rule*. These arbitration rules are dependent upon the manner in which the arbiter is learned as detailed below. This process is applied at each level until a final prediction is produced at the root of the tree. Since at each level, the leaf classifiers and arbiters are independent, predictions are generated in parallel. Before we discuss the arbitration process in detail, we first describe how arbiters are learned.

3.2 Meta-learning an arbiter tree

We experimented with several schemes to meta-learn a binary tree of arbiters. The training examples for an arbiter are selected from the original training examples used in its two subtrees.

In all these schemes the leaf classifiers are first learned from randomly chosen disjoint data subsets and the classifiers are grouped in pairs. (The strategy for pairing classifiers is discussed later.) For each pair of classifiers, the union of the data subsets on which the classifiers are trained is generated. This union set is then classified by the two classifiers. A *selection rule* compares the predictions from the two classifiers and selects instances from the union set to form the training set for the arbiter of the pair of classifiers. Thus, the rule acts as a data filter to produce a training set with a particular distribution of the examples. The arbiter is learned from this set with the same learning algorithm. In essence, we seek to compute a training set of data for the arbiter that the classifiers together do a poor job of classifying. The process of forming the union of data subsets, classifying it using a pair of arbiter trees, comparing the predictions, forming a training set, and training the arbiter is recursively performed until the root arbiter is formed.

For example, suppose there are initially four training data subsets ($T_1 - T_4$). First, four classifiers ($C_1 - C_4$) are generated in parallel from $T_1 - T_4$.

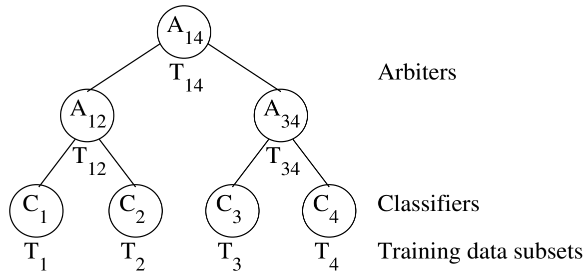

Figure 1: Sample arbiter tree

The union of 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). Based on predictions P_1 and P_2, and the subset U_{12}, a selection rule generates a training set (T_{12}) for the arbiter. The arbiter (A_{12}) is then trained from the set T_{12} using the same learning algorithm used to learn the initial classifiers. Similarly, arbiter A_{34} is generated in the same fashion starting from T_3 and T_4, in parallel with A_{12}, and hence all the first-level arbiters are produced. Then U_{14} is formed by the union of subset 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 completed (see Figure 1).

3.3 Detailed strategies

We experimented with three strategies for the *selection rule*, which generates training examples for the arbiters. Based on the predictions from two arbiter subtrees AT_1 and AT_2 (or two leaf classifiers) rooted at two sibling arbiters, and a set of training examples, E, the strategy generates a set of arbiter training examples, T. $AT_i(x)$ denotes the prediction of training example x by arbiter subtree AT_i. $class(x)$ denotes the given classification of example x. The three versions of this selection rule implemented and reported here are as follows:

1. Return instances with predictions that disagree, i.e., $T = T_d = \{x \in E \mid AT_1(x) \neq AT_2(x)\}$. Thus, the arbiter will be used to decide between conflicting classifications. Note, however, it cannot distinguish classifications that agree but which are incorrect. (For further reference, this scheme is denoted as *meta-different*.)

2. Return instances with predictions that disagree, T_d, as in the first case, but also predictions that agree but are incorrect; i.e, $T = T_d \cup T_i$, where $T_i = \{x \in E \mid (AT_1(x) = AT_2(x)) \wedge (class(x) \neq AT_1(x))\}$. Note that we lump together both cases of data that are incorrectly classified or are in disagreement. (Henceforth, denoted as *meta-different-incorrect*).

3. Return a set of three training sets: T_d and T_i, as defined above, and T_c with examples that have the same correct predictions; i.e., $T = \{T_d, T_i, T_c\}$, where $T_c = \{x \in E \mid (AT_1(x) = AT_2(x)) \wedge (class(x) = AT_1(x))\}$. Here we attempt to separate the data into three cases and distinguish each case by learning a separate "subarbiter." T_d, T_i, and T_c generate A_d, A_i, and A_c, respectively. The first arbiter is like the one computed in the first case to arbitrate disagreements. The second and third arbiters attempt to distinguish the cases when the two predictions agree but are either incorrect or correct. (Henceforth, denoted as *meta-different-incorrect-correct*).

Sample training sets generated by the three schemes are depicted in Figure 2.

The learned arbiters are trained on the particular distinguished distributions of training data and are used in generating predictions. (Note that the arbiters are trained by the same learner used to train the leaf classifiers.) Recall, however, at each arbiter we have two predictions, p_1 and p_2, from two lower level arbiter subtrees (or leaf classifiers) and the arbiter's, A, own prediction to arbitrate between. $A_i(x)$ is denoted as the prediction of training example x by arbiter A_i. Two versions of the *arbitration rule* have been implemented. The first version corresponds to the first two *selection* strategies, while the second version corresponds to the third strategy. We denote by $instance$ the test instance to be classified.

1&2. Return the majority vote of p_1, p_2, and $A(instance)$, with preference given to the arbiter's choice; i.e., if $p_1 \neq p_2$ return $A(instance)$ else return p_1.

Class	Attribute vector	Example	Base classifiers' predictions	
$class(x)$	$attrvec(x)$	x	$C_1(x)$	$C_2(x)$
a	$attrvec_1$	x_1	a	a
b	$attrvec_2$	x_2	a	b
c	$attrvec_3$	x_3	b	b
b	$attrvec_4$	x_4	b	b

Training set from the *meta-different* arbiter scheme		
Instance	Class	Attribute vector
1	b	$attrvec_2$

Training set from the *meta-different-incorrect* arbiter scheme		
Instance	Class	Attribute vector
1	b	$attrvec_2$
2	c	$attrvec_3$

Training set from the *meta-different-incorrect-correct* arbiter scheme			
Set	Instance	Class	Attribute vector
Different (T_d)	1	b	$attrvec_2$
Incorrect (T_i)	1	c	$attrvec_3$
Correct (T_c)	1	a	$attrvec_1$
	2	b	$attrvec_4$

Figure 2: Sample training sets generated by the three arbiter strategies

3. if $p_1 \neq p_2$ return $A_d(instance)$
   else if $p_1 = A_c(instance)$
   return $A_c(instance)$
   else return $A_i(instance)$,
   where $A = \{A_d, A_i, A_c\}$.

To achieve significant speed-up, the training set size for an arbiter, in these three schemes, is restricted to be no larger than the training set size for a classifier. That is, the amount of computation in training an arbiter is bounded by the time to train a leaf classifier. In a parallel computation model each level of arbiters can be learned as efficiently as the leaf classifiers. (In the third scheme the three subarbiters are produced in parallel.) With this restriction, substantial speed-up of the learning phase can be predicted. Assume the number of data subsets of the initial distribution is s. Let $t = N/s$ be the size of each data subset, where N is the total number of training examples. Furthermore, assume the learning algorithm takes $O(n^2)$ time in the sequential case. In the parallel case, if we have s processors, there are $\log(s)$ interations in building the arbiter tree and each takes $O(t^2)$ time. The total time is therefore $O(t^2 \log(s))$, which implies a potential $O(s^2/\log(s))$ fold speed-up. Similarly, $O(n\sqrt{n})$ algorithms yield $O(\sqrt{s}/log(s))$ fold speed-up and $O(n)$ yield $O(s/log(s))$. This rough analysis also assumes that each data subset fits in the main memory. In addition, the estimates do not take into account the burden of communication overhead and speed gained by multiple I/O channels in the parallel case (which will be addressed in future papers). Furthermore, we assume that the processors have relatively the same speed; load balancing and other issues in a heterogeneous environment are beyond the scope of this paper. We also note that once an arbiter tree is computed, its application in a parallel environment can be done efficiently according to the scheme proposed in [18].

4 Experiments

Four inductive learning algorithms were used in our experiments. ID3 [15] and CART [1] were obtained from NASA Ames Research Center in the IND package [2]. They are both decision tree learning algorithms. WPEBLS is the weighted version of PEBLS [8], which is a memory-based learning algorithm. BAYES is a simple Bayesian learner based on conditional probabilities, which is described in [7]. The latter two algorithms were reimplemented in C.

Two data sets, obtained from the UCI Machine Learning Database, were used in our studies. The secondary protein structure data set (SS) [13], courtesy of Qian and Sejnowski, contains sequences of amino acids and the secondary structures at the corresponding positions. There are three structures (three classes) and 20 amino acids (21 attributes because of a spacer) in the data. The amino acid sequences were split into shorter sequences of length 13 according to a windowing technique used in [13]. The sequences were then divided into a training and test set, which are disjoint, according to the distribution described in [13]. The training set has 18105 instances and the test set has 3520. The DNA splice junction data set (SJ) [19], courtesy of Towell, Shavlik and Noordewier, contains sequences of nucleotides and the type of splice junction, if any, (three classes) at the center of each sequence. Each sequence has 60 nucleotides with 8 different values each (four base ones plus four combinations). Some 2552 sequences were randomly picked as the training set and the rest, 638 sequences, became the test set. Although these are not very large data sets, they give us an idea on how our strategies perform.

As mentioned above, we varied the number of subsets from 2 to 64 and the equal-size subsets are disjoint with random distribution of classes. The prediction accuracy on the test set is our primary comparison measure. The three meta-learning strategies for arbiters were run on the two data sets with the four learning algorithms. In addition, we applied a simple voting scheme on the leaf classifiers. The results are plotted in Figure 3. The accuracy for the serial case is plotted as "one subset."

If we relax the restriction on the size of the data set for training an arbiter, we might expect an improvement in accuracy, but a decline in execution speed. To test this hypothesis, a number of experiments were performed varying the maximum training set size for the arbiters. The different sizes are constant multiples of the size of a data subset. The results plotted in Figure 4 were obtained from using the *meta-different-incorrect* strategy on the SJ data.

5 Results

In Figure 3, for the three arbiter strategies, we observe that the accuracy stayed roughly the same for the SS data and slightly decreased for the SJ data when the number of subsets increased. With 64 subsets, most of the learners exhibited approximately a 10% drop in accuracy, with the exception of BAYES and one case in CART. The sudden drop in accuracy in those cases was likely due to the lack of information in the training data subsets. In the SJ data there are only $\sim$ 40 training examples in each of the 64 subsets. If we look at the case with 32 subsets ($\sim$ 80 examples each), all the learners sustained a drop in accuracy of at most 10%. As we observe, the big drops did not happen in the SS data. This shows that the data subset size cannot be too small. The voting scheme performed poorly for the SJ data, but was better than most schemes in the SS data. Basically, the accuracy was roughly the same percentage of the most frequent class in the data and in the SS data case, simple voting performed relatively better. The behavior on the training set was similar to the test set and those results are not presented here due to space limitations. Furthermore, the three strategies had comparable performance and since the first strategy produces fewer examples in the arbiter training sets, it is the preferred strategy.

As we expected, by increasing the maximum arbiter training set size, higher accuracy can be obtained (see Figure 4, only the results on the SJ data are presented). When the maximum size was just two times the size of the original subsets, the largest accuracy drop was less than 5% accuracy, cutting half of the 10% drop occurred when the maximum size was the same as the subset size as mentioned above. Furthermore, when the maximum size was unlimited (i.e., at most the size of the entire training

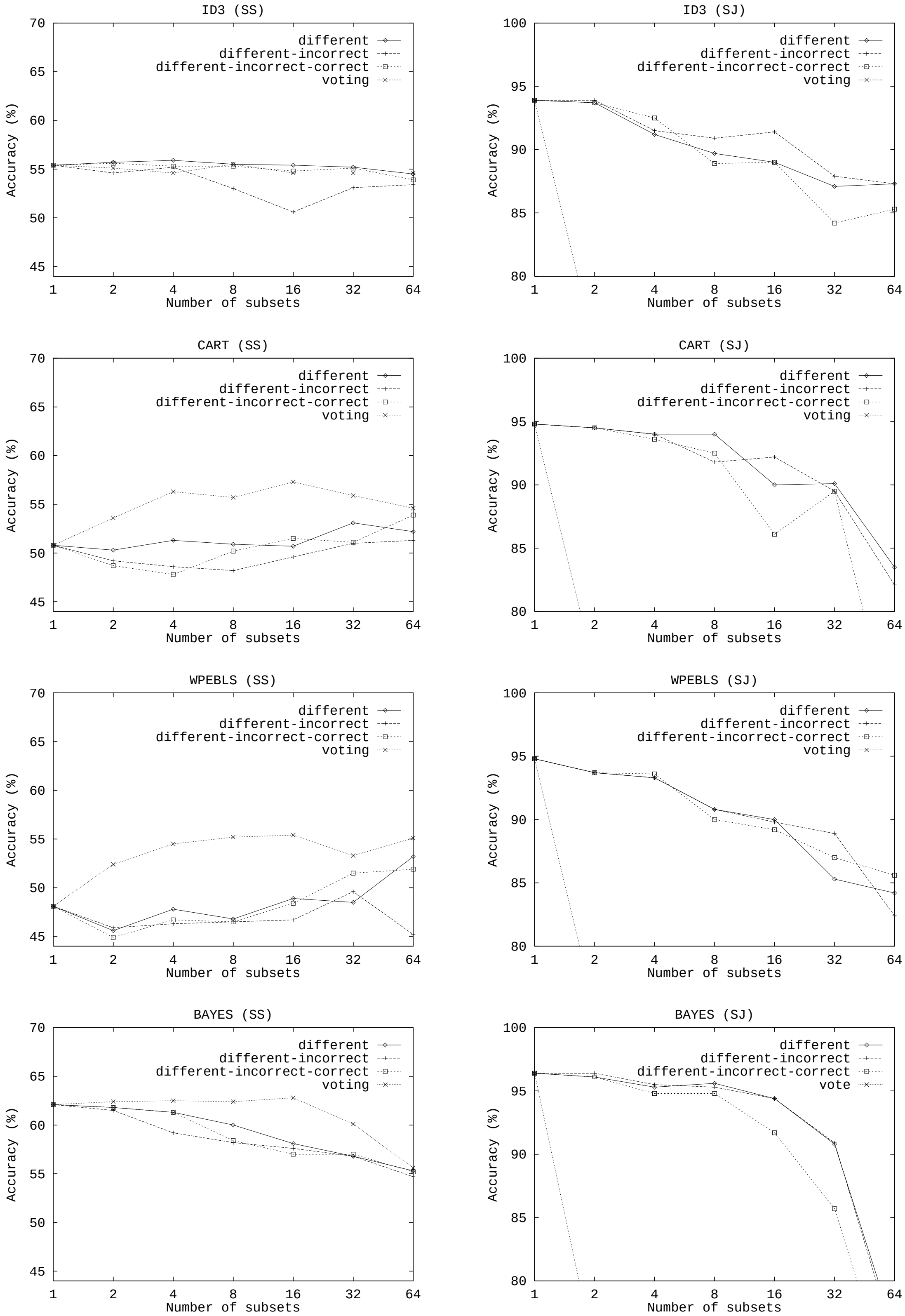

Figure 3: Results on different strategies

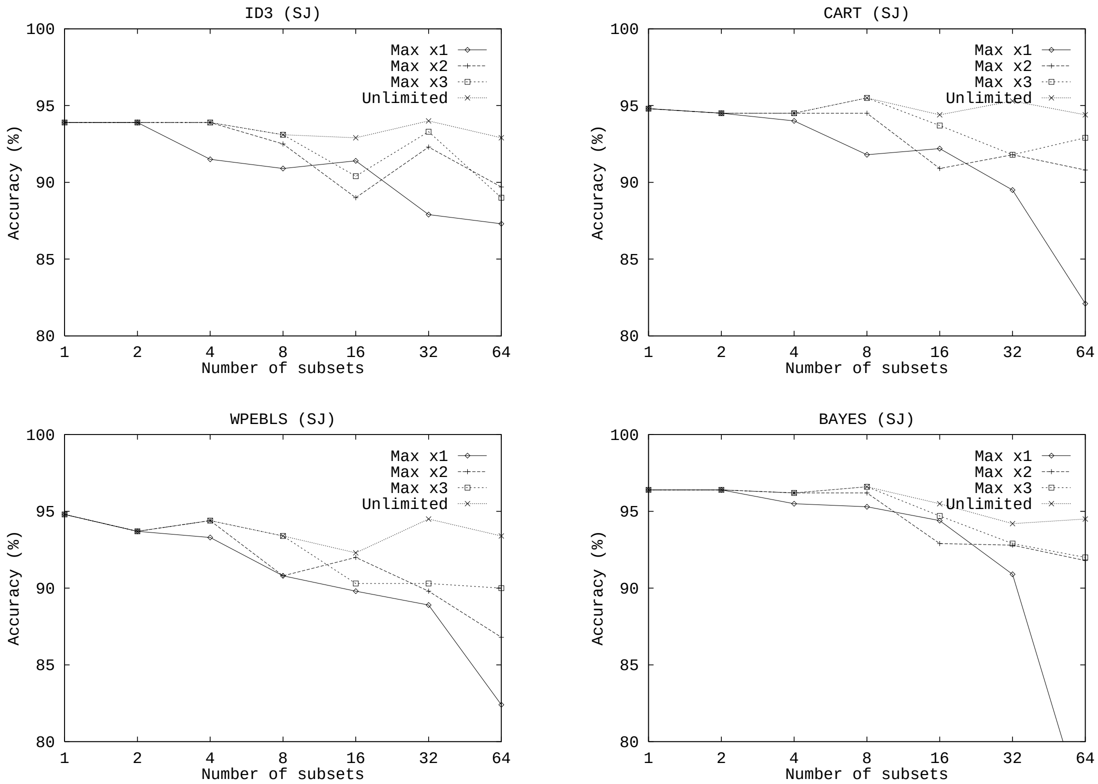

Figure 4: Results on different maximum arbiter training set sizes

set), the accuracy was roughly the same as in the serial case.

Since our experiments are still preliminary, the system is not implemented on a parallel/distributed platform and we do not have relevant timing results. However, according to the theoretical analysis, significant speed-up can be obtained when the maximum arbiter training set size is fixed to the subset size (see Section 3.3; ID3 and CART is $O(nl)$ [15], where l is the number of leaves, and assuming l is proportional to $\sqrt{n}$, the complexity becomes $O(n\sqrt{n})$; WPEBLS is $O(n^2)$; BAYES is $O(n)$).

When the maximum arbiter training set size is unlimited, we calculate the speed-ups based on the largest training set size at each level (which takes the longest to finish). In this case, since we assume we do not have a parallelized version of the learners available, the computing resource being utilized is reduced in half at each level. That is, at the root level, only one processor will be in use. The following discussion is based on theoretical calculations using recorded arbiter training set sizes obtained from the SJ data. Again, the effects of communication and multiple I/O channels on speed are not taken into account. For WPEBLS ($O(n^2)$), the speed-up steadily increased as the number of subsets increased and leveled off after eight subsets to a factor of six (see Figure 5). (This case is closest to a linear speed-up.) For ID3 and CART ($O(n\sqrt{n})$), the speed-up slowly increased and leveled off after eight subsets to a factor of three. BAYES did not exhibit parallel speed-up due to its linear complexity. The leveling-off was mainly due to the bottleneck at the root level, which had the largest training set.

Next, we investigate the size and location of the largest training set in the entire arbiter tree. This gives us a notion of the memory requirement at any processing site and the location of the main bottleneck. Our empirical results indicates that the largest training set size was always around 30% of the total training set and always happened at the root level, independent of the number of subsets that was larger than two. (Note when the number of subsets was two, the training set size was 50% of the original set at the leaves and became the largest in the tree.) This implies that the bottleneck was in

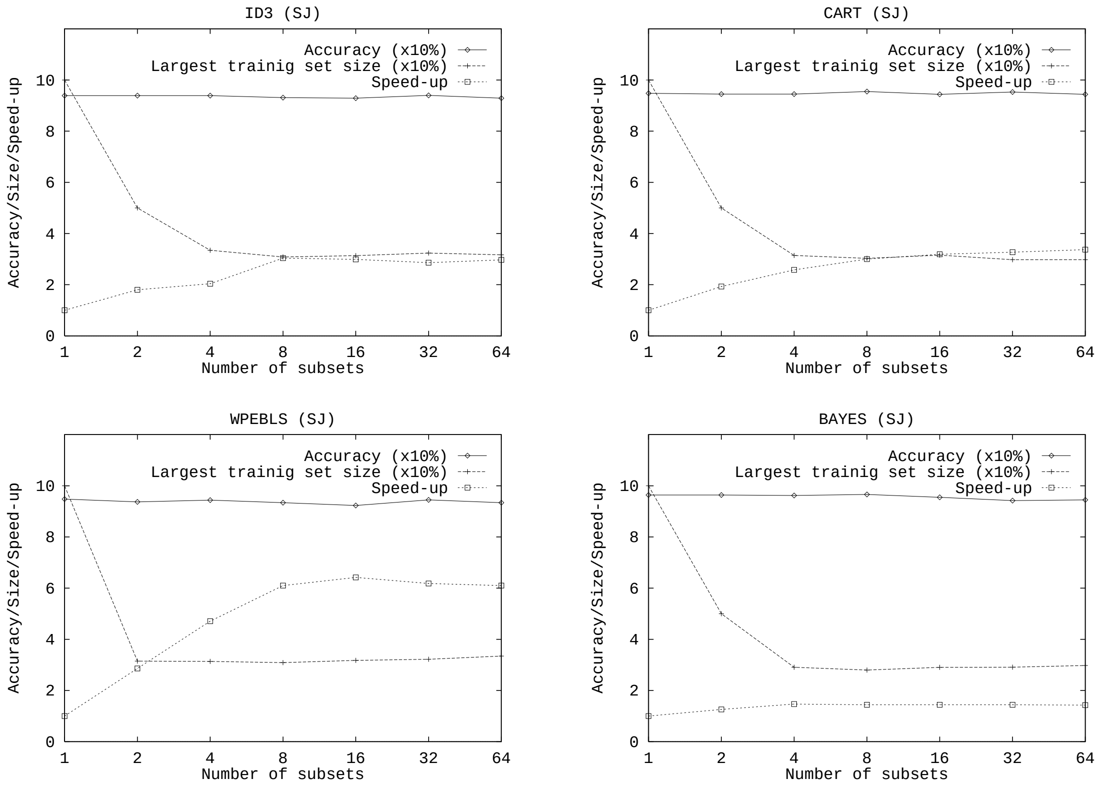

Figure 5: Results on unlimited maximum arbiter training set size

processing around 30% of the entire training data set at the root level. This also implies that this parallel meta-learning strategy required only around 30% of the memory used by the serial case at any single processing site. Strategies for reducing the largest training set size is discussed in the next section. Recall that the accuracy level of this parallel strategy is roughly the same as the serial case. Thus, the parallel meta-learning strategy (with no limit on the arbiter training set size) can perform the same job as the serial case with less time and memory without parallelizing the learning algorithms.

6 Discussion

The two data sets chosen in our experiments represent two different kinds of data sets: one is difficult to learn (SS with 50+% accuracy) and the other is easy to learn (SJ with 90+% accuracy). Our arbiter schemes maintained the low accuracy in the first case and mildly degraded the high accuracy in the second case with a restriction on the arbiter training set size. When the restriction on the size of the training set for an arbiter was lifted, the same level of accuracy could be achieved with less time and memory for the second case. Since we assert that this approach is scalable due to the independence of each learning process, this indicates the robustness of our strategies and hence their effectiveness on massive amounts of data.

Largest arbiter training set size As mentioned in the previous section, we discovered that our scheme required at most 30% of the entire training set at any moment to maintain the same prediction accuracy as in the serial case for the SJ data. However, the percentage 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.

If the prediction accuracy is high, the arbiter training sets will be small because the predictions will usually be correct and few disagreements will occur. In our experiments, the distribution of data in the subsets was random and later we discovered that

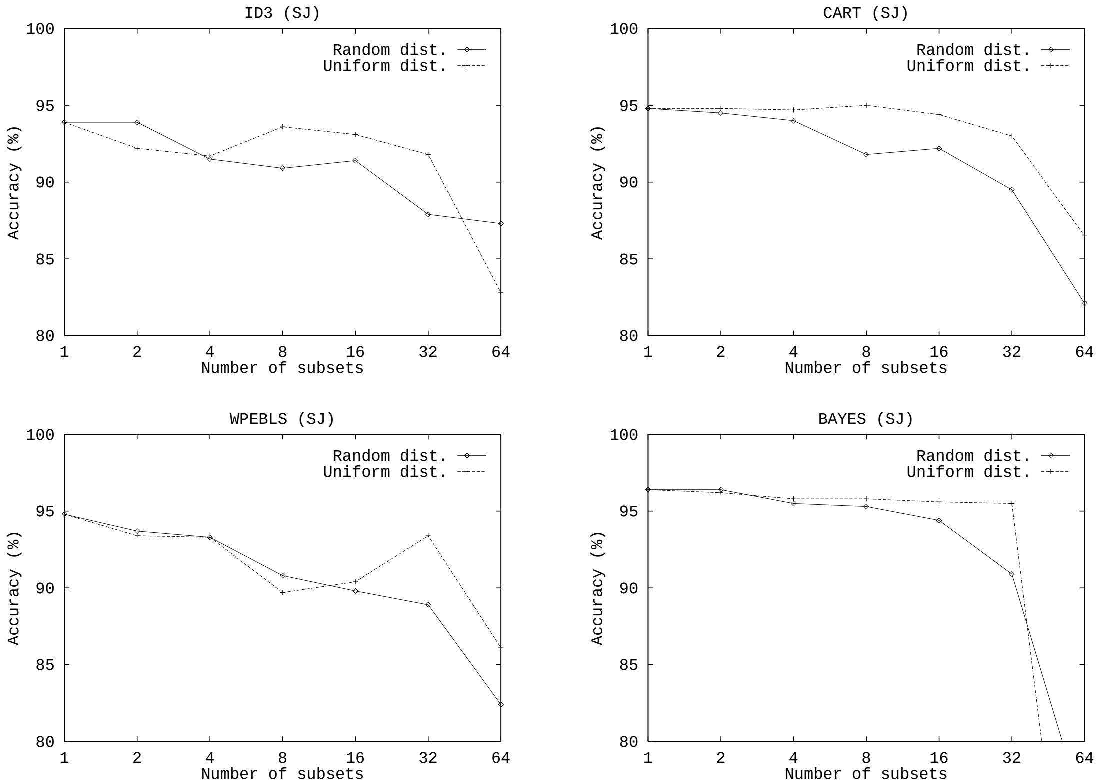

Figure 6: Accuracy with different class distributions

half of the final arbiter tree was trained on examples with only two of the three classes. That is, half of the tree was not aware of the third class appearing in the entire training data. We postulate that if the class distribution in the subsets is uniform, the leaf classifiers and arbiters in the arbiter tree will be more accurate and hence the training sets for the arbiter will be smaller. Indeed, results from our additional experiments on different class distributions (using the *meta-different-incorrect strategy* on the SJ data set), shown in Figure 6, indicate that uniform class distributions can achieve higher accuracy than random distributions.

And lastly, the "neighboring" leaf classifiers and arbiters were paired in our experiments. One might use more sophisticated schemes for pairing to reduce the size of the arbiter training sets. One scheme is to pair classifiers and arbiters that agree most often with each other and produce smaller training sets (called *min-size*). Another scheme is to pair those that disagree the most and produce larger training sets (called *max-size*). At first glance the first scheme would seem to be more attractive. However, since disagreements are present, if they do not get resolved at the bottom of the tree, they will all surface near the root of the tree, which is also when the choice of pairings is limited or nonexistent (there are only two arbiters one level below the root). Hence, it might be more beneficial to resolve conflicts near the leaves leaving fewer disagreements near the root.

These sophisticated pairing schemes might decrease the arbiter training set size, but they might also increase the communication overhead. When pairing is performed at every level, the overhead is incurred at every level. The schemes 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.

Some experiments were performed on the two pairing strategies applied only at the leaf level and the results are shown in Figure 7. All these experiments were conducted on the SJ data set and

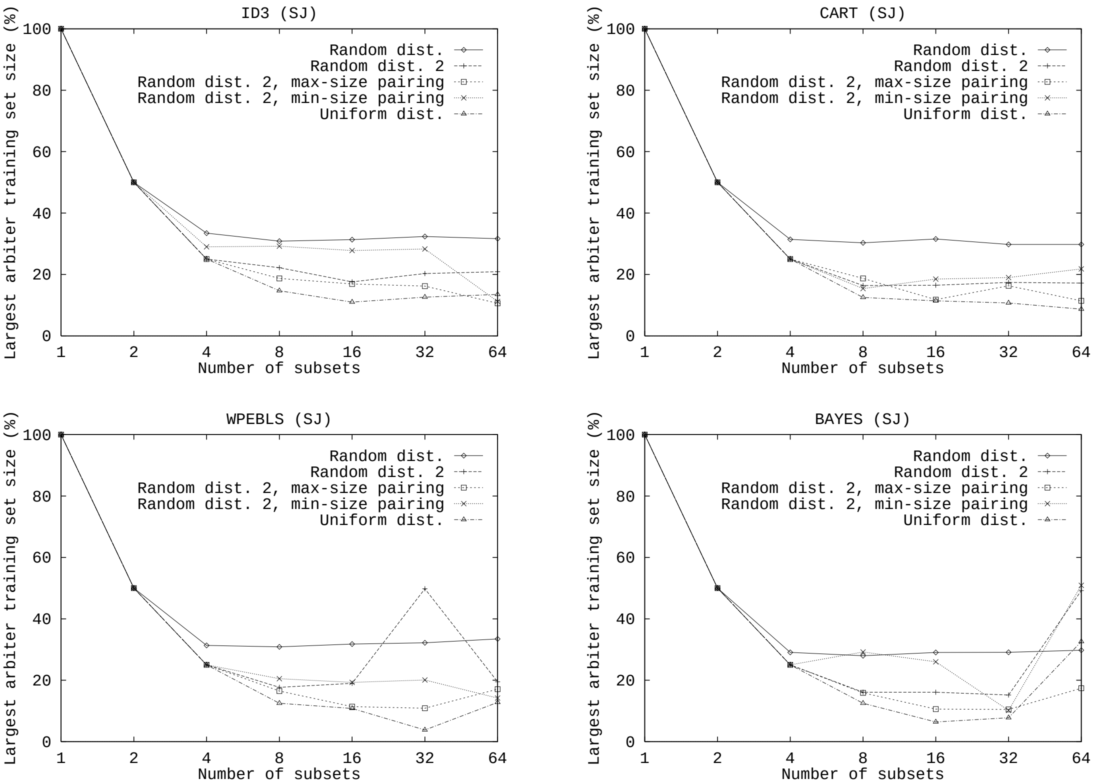

Figure 7: Arbiter training set size with different class distributions and pairing strategies

used the *meta-different-incorrect* strategy for meta-learning arbiters. In addition to the initial random class distribution, a uniform class distribution and a second random class distribution (*Random dist. 2*) was used. The second random distribution does not have the property that half of the learned arbiter tree was not aware of one of the classes, as 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 was around 10% of the original 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, which can also be observed from results presented earlier). (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 for the second random distribution. The *min-size* pairing strategy, on the other hand, did not affect, or sometimes even increased, the sizes. In summary, uniform class distribution tends to produce the smallest training sets and the *max-size* pairing strategy can reduce the set sizes in random class distributions.

In our discussion so far, we have assumed that the arbiter training set is unbounded in order to determine how the pairing strategies may behave in the case where the training set size is bounded. The *max-size* strategy aims at resolving conflicts near the leaves where the maximum possible arbiter training set size is small (the union of the two subtrees) leaving fewer conflicts near the root. If the training set size is bounded at each node, a random sample (with the bounded size) of a relatively small set near the root would be representative of the set chosen when the size is unbounded.

Order of the arbiter tree A binary arbiter tree configuration was chosen for experimental purposes. There is no apparent reason why the arbiter tree cannot be n-ary. However, the different strategies proposed above are designed for n to be equal to two. When n is greater than two, a majority classification from the n predictions might be sufficient as an arbitration rule. The examples that do not receive a majority classification constitute the training set for an arbiter. It might be worthwhile to have a large value of n since the final tree will be shallow, and thus training may be faster. However, more disagreements and higher communication overhead will appear at each level in the tree due to the arbitration of many more predictions at a single arbitration site.

Alternate approach An anonymous reviewer of another paper proposed an "optimal" formula based on Bayes Theorem to combine the results of classifiers, namely, $P(x) = \sum_c P(c) \times P(x|c)$, where x is a prediction and c is a classifier. $P(c)$ is the prior which represents how likely classifier c is the true model and $P(x|c)$ represents the probability classifier c guesses x. Therefore, $P(x)$ represents the combined probability of prediction x to be the correct answer. Unfortunately, to be optimal, Bayes Theorem requires the priors $P(c)$'s to be known, which are usually not, and it also requires the summation to be over all possible classifiers, which is almost impossible to achieve. However, an approximate $P(x)$ can still be calculated by approximating the priors using various established techniques on the training data and using only the classifiers available. This technique is essentially a "weighted voting scheme" and can be used as an alternative to generating arbiters. This and the aforementioned strategies and issues are the subject matter of ongoing experimentation.

Schapire's hypothesis boosting Our ideas are related to using meta-learning to improve accuracy. The most notable work in this area is due to Schapire [16], which he refers to as *hypothesis boosting*. Based on an initial learned hypothesis for some concept derived from a random distribution of training data, Schapire's scheme iteratively generates two additional distributions of examples. The first newly derived distribution includes randomly chosen training examples that are equally likely to be correctly or incorrectly classified by the first learned classifier. A new classifier is formed from this distribution. Finally, a third distribution is formed from the training examples on which both of the first two classifiers disagree. A third classifier (in effect, an arbiter) is computed for this distribution. The predictions of the three learned classifiers are combined using a simple arbitration rule similar to the one of the rules we presented above. Schapire rigorously proves that the overall accuracy is higher than the one achieved by simply applying the learning algorithm to the initial distribution under the PAC learning model. In fact, he shows that arbitrarily high accuracy can be achieved by recursively applying the same procedure. However, his approach is limited to the PAC model of learning, and furthermore, the manner in which the distributions are generated does not lend itself to parallelism. Since the second distribution depends on the first and the third depends on the second, the distributions are not available at the same time and their respective learning processes cannot be run concurrently. We use three distribution as well, but the first two are independent and are available simultaneously. The third distribution, for the arbiter, however, depends on the first two. Freund [9] has a similar approach, but with potentially many more distributions. Again, the distributions can only be generated iteratively.

Work in progress In addition to applying meta-learning to combining results from a set of parallel or distributed learning processes, meta-learning can also be used to coalesce the results from multiple different inductive learning algorithms applied to the same set of data to improve accuracy [5]. The premise is that different algorithms have different representations and search heuristics, different search spaces are being explored and hence potentially diversed results can be obtained from different algorithms. Mitchell [12] refers to this phenomenon as *inductive bias*. We postulate that by combining the different results intelligently through meta-learning, higher accuracy can be obtained. We call this approach *multistrategy hypothesis boosting*.

Preliminary results reported in [4] are encouraging. Zhang et al.'s [24] and Wolpert's [22] work is in this direction. Silver et al.'s [17] and Holder's [10] work also employs multiple learners, but no learning is involved at the meta level. Since the ultimate goal of this work is to improve both the accuracy and efficiency of machine learning, we have been working on combining ideas in *parallel learning*, described in this paper, with those in *multistrategy hypothesis boosting*. We call this approach *multistrategy parallel learning*. Preliminary results reported in [6] are encouraging. To our knowledge, not much work in this direction has been attempted by others.

7 Concluding Remarks

Several *meta-learning* schemes for *parallel learning* are presented in this paper. In particular, schemes for building arbiter trees are detailed. Preliminary empirical results from bounded arbiter training sets indicate that the presented strategies are viable in speeding up learning algorithms with small degradation in prediction accuracy. When the arbiter training sets are unbounded, the strategies can preserve prediction accuracy with less training time and required memory than the serial version.

The schemes presented here is a step toward the *multistrategy parallel learning* approach and the preliminary results obtained are encouraging. More experiments are being performed to ensure that the results we have achieved to date are indeed statistically significant, and to study how meta-learning scales with much larger data sets. We intend to further explore the diversity and possible "symbiotic" effects of multiple learners to improve our meta-learning schemes in a parallel environment.

Acknowledgements

This work has been partially supported by grants from New York State Science and Technology Foundation, Citicorp, and NSF CISE. We thank David Wolpert for many useful and insightful discussions that substantially improved the ideas presented in this paper.

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[2] W. Buntine and R. Caruana. *Introduction to IND and Recursive Partitioning*. NASA Ames Research Center, 1991.

[3] J. Catlett. Megainduction: A test flight. In *Proc. Eighth Intl. Work. Machine Learning*, pages 596–599, 1991.

[4] P. Chan and S. Stolfo. Experiments on multistrategy learning by meta-learning. Submitted to CIKM93, 1993.

[5] P. Chan and S. Stolfo. Meta-learning for multistrategy and parallel learning. In *Proc. Second Intl. Work. on Multistrategy Learning*, 1993. To appear.

[6] P. Chan and S. Stolfo. Toward multistrategy parallel and distributed learning in sequence analysis. In *Proc. First Intl. Conf. Intel. Sys. Mol. Biol.*, 1993. To appear.

[7] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–285, 1987.

[8] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.

[9] Y. Freund. Boosting a weak learning algorithm by majority. In *Proc. 3rd Work. Comp. Learning Theory*, pages 202–216, 1990.

[10] L. Holder. Selection of learning methods using an adaptive model of knowledge utility. In *Proc. MSL-91*, pages 247–254, 1991.

[11] C. Matheus, P. Chan, and G. Piatesky-Shapiro. Systems for knowledge discovery in databases. *IEEE Trans. Know. Data. Eng.*, 1993. To appear.

[12] T. M. Mitchell. The need for biases in learning generalizaions. Technical Report CBM-TR-117, Dept. Comp. Sci., Rutgers Univ., 1980.

[13] N. Qian and T. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, 202:865–884, 1988.

[14] J. R. Quinlan. Induction over large data bases. Technical Report STAN-CS-79-739, Comp. Sci. Dept., Stanford Univ., 1979.

[15] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[16] R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–226, 1990.

[17] B. Silver, W. Frawley, G. Iba, J. Vittal, and K. Bradford. ILS: A framework for multi-paradigmatic learning. In *Proc. Seventh Intl. Conf. Machine Learning*, pages 348–356, 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] B. Wah et al. High performance computing and communications for grand challenge applications: Computer vision, speech and natural language processing, and artificial intelligence. *IEEE Trans. Know. Data. Eng.*, 5(1):138–154, 1993.

[21] J. Wirth and J. Catlett. Experiments on the costs and benefits of windowing in ID3. In *Proc. Fifth Intl. Conf. Machine Learning*, pages 87–99, 1988.

[22] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[23] X. Zhang, M. Mckenna, J. Mesirov, and D. Waltz. An efficient implementation of the backpropagation algorithm on the connection machine CM-2. Technical Report RL89-1, Thinking Machines Corp., 1989.

[24] X. Zhang, J. Mesirov, and D. Waltz. A hybrid system for protein secondary structure prediction. *J. Mol. Biol.*, 225:1049–1063, 1992.