A Comparative Evaluation of Voting and Meta-learning on Partitioned Data

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. Some learning algorithms assume that the entire data set fits into main memory, which is not feasible for massive amounts of data. One approach to handling a large data set is to partition the data set into subsets, run the learning algorithm on each of the subsets, and combine the results. In this paper we evaluate different techniques for learning from partitioned data. Our *meta-learning* approach is empirically compared with techniques in the literature that aim to combine multiple evidence to arrive at one prediction.

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 (Wah, 1993) are perhaps the best examples. Financial institutions and market analysis firms are already dealing with overwhelming amounts of global information that in time will undoubtedly grow in size faster than improvements in machine resources. Some learning algorithms require all the data to be resident in main memory. However, this requirement becomes untenable when the amount of data exceeds the size of main memory, which is obviously possible for any realistic database; we call this the *scaling* problem. Even with large virtual memory, constantly swapping data in and out of memory becomes a significant overhead. Furthermore, it is not inconceivable that the amount of data can exceed the virtual memory. This is also why data are disk-resident in database management systems.

One approach to solve the scaling problem is *data reduction*, meaning to partition the data set into smaller subsets, apply learning algorithms on each subset, followed by a phase that combines the learned results. Each subset is sized to fit into main memory. In addition to alleviating the memory restriction problem, we can speed up the process by running the learning programs in parallel on multiple processors. In fact, parallel and distributed learning motivated us to investigate learning from partitioned data. Our ultimate goal is to develop a sound approach to scalable and accurate learning systems for massive amounts of distributed data. However, in such schemes one may presume that accuracy will suffer; i.e., combining results for separate classifiers may not be as accurate as learning from the entire data set. Thus, it is important to determine which schemes for combining results have minimal impact on the quality of the final result. Furthermore, we note that the partitioned data approach reported here is different from much of the similar work which combines multiple classifiers trained from the "entire" data set for accuracy improvement.

In this paper we study different techniques for combining predictions generated by a set of *base classifiers*, each of which is computed by a learning algorithm applied to a distinct data subset. Common techniques such as voting and statistical schemes are evaluated. These familiar techniques are compared to our proposed meta-learning techniques, which were first presented in (Chan & Stolfo, 1993b). The contributions of this paper are two fold. We systematically compare schemes reported in the literature to our proposed meta-learning techniques. We also demonstrate empirically that our approach produces more accurate trained classifiers than the other schemes.

2 Common Voting and Statistical Techniques

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 the subsets. 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 heaviest weight.

Littlestone and Warmuth (1989) propose several *weighted majority* algorithms for combining 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 combining 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 $limit$. A discount can only occur if the weight is larger than $\frac{\gamma}{number\_of\_classifiers}$ times the total weight of all classifiers, where $0 \leq \gamma < .5$. Another variation, called WMR, produces randomized responses. The probability of a classification selected as the final prediction is the total weight of that classification divided by the total weight of all classifications; i.e., $P(class_x) = \frac{total\_weight(class_x)}{\sum_i total\_weight(class_i)}$. The weights are trained as in the WM algorithm.

Littlestone and Warmuth's (1989) *weighted majority* work is mainly theoretical. Their model assumes that the classifiers make binary predictions. They show that if the best classifier makes m mistakes, the *weighted majority* algorithms will make at most $c(log(number\_of\_classifiers) + m)$ mistakes, where c is a fixed constant. We adapt their techniques in this study to include classifiers that predict an arbitrary number of classes. Again, we use a validation set to train the weights in the weighted majority algorithms.

Xu et al. (1992) developed a method for combining 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. We estimate the conditional probabilities from the frequencies generated from the validation set.

We now discuss our meta-learning techniques for combining classifications produced by multiple classifiers. These techniques differ from the simple voting and statistical methods we just discussed.

3 Meta-learning Techniques

Rather than learning weights, our approach introduced in (Chan & Stolfo, 1993b) is to *meta-learn* a set of new classifiers (or *meta-classifiers*) whose training data are based on predictions of a set of base classifiers. Our techniques fall into two general categories: the *arbiter* and *combiner* schemes.

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. For purposes of this study, we ignore the representation used by the classifier (to preserve the algorithm-independent property). 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.

3.1 Arbiter

An *arbiter* (Chan & Stolfo, 1993d) is learned by some learning algorithm to arbitrate among predictions generated by different base classifiers. That is, its purpose is to provide an alternate and more educated prediction when the base classifiers present diverse predictions. This arbiter, together with an *arbitration rule*, decides a final classification outcome based upon the base predictions. Figure 1 depicts how the final prediction is made with predictions from two base classifiers and a single arbiter. The details of how the final decision is made follows.

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 votes 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. The training set of an arbiter is generated in a way that it contains the raw training examples whose classifications the base classifiers cannot predict consistently. Formally, a training set T for the arbiter is generated by picking examples from the validation set E. The choice of examples is dictated by a *selection rule*. One version of a selection rule studied here is as follows:

- An instance is selected if none of the classes

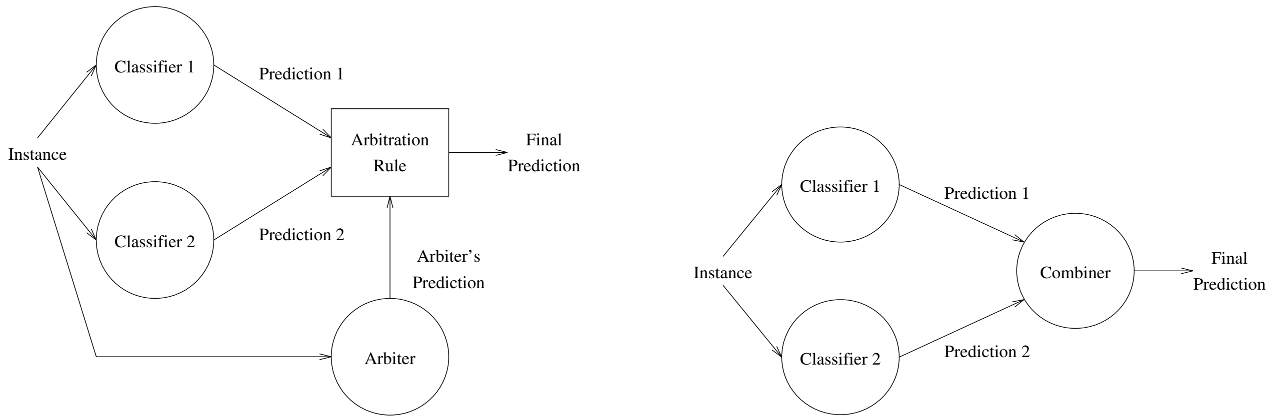

Figure 1: An arbiter and a combiner with two classifiers.

in the k base predictions gathers a majority vote ($> k/2$ votes); i.e., $T = \{x \in E \mid no\_majority(C_1(x), C_2(x), ...C_k(x))\}$.

The purpose of this rule is to choose examples that are confusing; i.e., the majority of classifiers do not agree. Figure 2 presents a sample training set for the arbiter strategy. 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. Other arbiter schemes were investigated in (Chan & Stolfo, 1995).

3.2 Combiner

The aim of the *combiner* strategy (Chan & Stolfo, 1993a) is to coalesce the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction. For example, a base classifier might consistently make the correct predictions for class c; i.e., when this base classifier predicts class c, it is probably correct regardless of the predictions made by the other base classifiers. In the *combiner* 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). We note that 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 (1992). (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*.)

Figure 2 presents sample training sets for these two combiner schemes. Other combiner schemes were studied in (Chan & Stolfo, 1993a).

Experiments were run to compare the accuracy of the different techniques we presented so far. The next section discusses our findings.

4 Experiments and Results

Two inductive learning algorithms were used in our experiments. ID3 (Quinlan, 1986) and CART (Breiman *et al.*, 1984) were obtained from NASA Ames Research Center in the IND package (Buntine & Caruana, 1991). They are both decision tree learning algorithms that require all training examples to be resident in main memory.

Two data sets were used in our studies. The DNA splice junction (SJ) data set (Towell *et al.*, 1990), courtesy of Towell, Shavlik and Noordewier, contains sequences of nucleotides and the type of splice junction, if any, at the

Class	Attribute vector	Example	Base classifiers' predictions		
$class(x)$	$attribute\_vector(x)$	x	$C_1(x)$	$C_2(x)$	$C_3(x)$
table	$attrvec_1$	x_1	table	table	table
chair	$attrvec_2$	x_2	table	chair	lamp
lamp	$attrvec_3$	x_3	lamp	chair	table

Training set for the *arbiter* scheme		
Instance	Class	Attribute vector
1	chair	$attrvec_2$
2	lamp	$attrvec_3$

Training set for the *class-combiner* scheme		
Instance	Class	Attribute vector
1	table	(table, table, table)
2	chair	(table, chair, lamp)
3	lamp	(lamp, chair, table)

Training set for the *class-attribute-combiner* scheme		
Instance	Class	Attribute vector
1	table	(table, table, table, $attrvec_1$)
2	chair	(table, chair, lamp, $attrvec_2$)
3	lamp	(lamp, chair, table, $attrvec_3$)

Figure 2: Sample training sets generated by the *arbiter* and *combiner* strategies with three base classifiers.

center of each sequence. There are three possible classes in this task. Each sequence has 60 nucleotides with 8 different values each (four base ones plus four combinations). The data set contains 3,190 training instances. The protein coding region (PCR) data set (Craven & Shavlik, 1993), courtesy of Craven and Shavlik, contains DNA nucleotide sequences and their binary classifications (coding or non-coding). Each sequence has 15 nucleotides with four different values each. The PCR data set has 20,000 sequences. The two data sets chosen in our experiments represent two different kinds of data sets: learning algorithms perform well on the SJ data set (90+% accuracy) and not so well on the PCR data set (70+%).

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 set used for generating the *combining structures* (weights/probabilities/arbiters/combiners) is twice the size of a subset. The prediction accuracy on a separate test set is our primary comparison measure. The voting, statistical, and meta-learning strategies discussed in this paper were run on the two data sets with the two learning algorithms. The results are plotted in Figure 3. The accuracy for the serial case 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." The average of the highest accuracy among the base classifiers in each run is plotted with label "max-base." The plotted accuracy is the average of 10-fold cross-validation runs. Statistical significance was measured using the one-sided t-test with 90% confidence value.

For the splice junction data set, 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 ~ 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 region 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 two subsets and climb back to the original accuracy level when the number of subsets increases. The accuracy difference between the arbiter scheme and the other non-meta-learning techniques is statistically significant at two and four subsets. 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.

From our experiments, our meta-learning strategies compare favorably with the weighted voting techniques across domains and learners used in this study. However, the meta-learning techniques do not always outperform the weighted schemes. In the splice junction domain, the *combiner* tech-

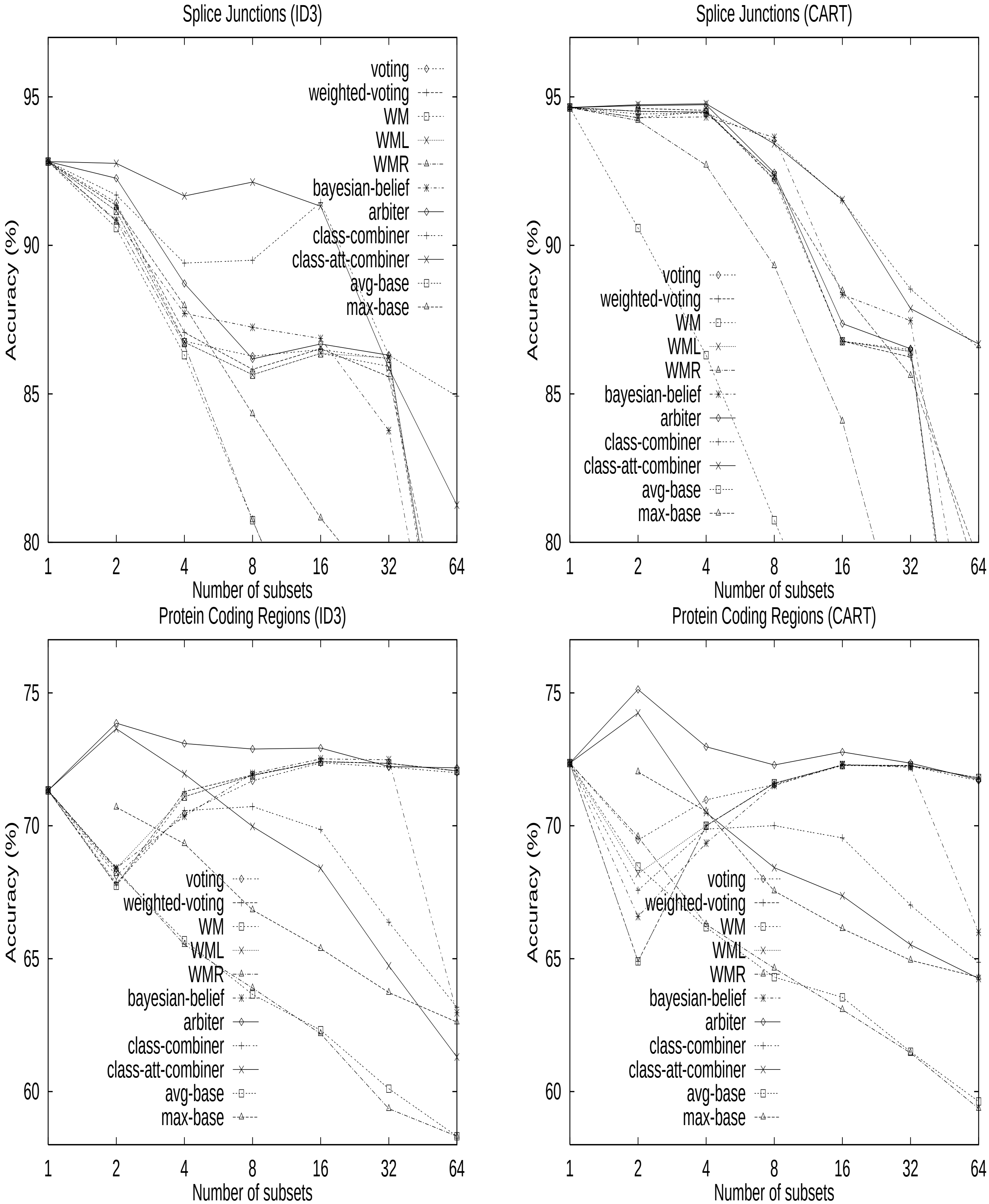

Figure 3: Accuracy for the one-level combining techniques.

niques are more favorable while in the protein coding region domain the *arbiter* technique is. It is not clear under what circumstances a particular meta-learning strategy will perform favorably. More experiments and studies are underway in an attempt to gain an understanding of these circumstances.

As we observe in the splice junction domain, none of the schemes can maintain the original 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 combining structures. In the next section we discuss a more sophisticated meta-learning approach, called *arbiter tree*, that generates multiple levels of combining structures.

5 Arbiter Tree

In Section 3 we discussed how an arbiter is learned and used. The *arbiter tree* approach learns *arbiters* in a bottom-up, binary-tree fashion. (The choice of a binary tree is to simplify our discussion.) An arbiter is learned from the output of a pair of learned classifiers 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 base classifiers at the leaves. 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 prediction; i.e., a classification of the test instance. From a pair of predictions and the parent arbiter's prediction, a 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. 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 base 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 base classifiers. To ensure efficient computation, we bound the size of the arbiter training set to the size of each data subset; i.e., the same data reduction technique is applied to learning arbiters. 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 ($T_1 - T_4$), processed by some learning algorithm, L. First, four classifiers ($C_1 - C_4$) are generated from $T_1 - T_4$. 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). A selection rule 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} using the same learning algorithm (L) used to learn the initial classifiers. Similarly, arbiter A_{34} is generated in the same fashion starting from T_3 and T_4 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. The resultant tree is depicted in Figure 4.

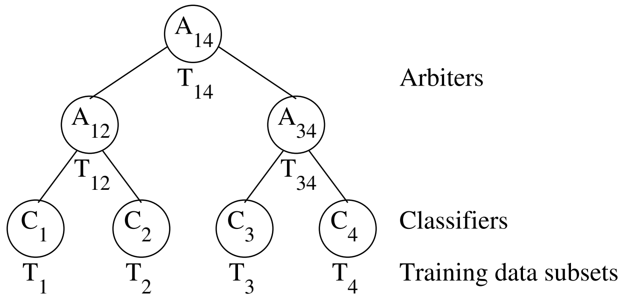

Figure 4: Sample binary arbiter tree.

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 higher communication overhead at each level in the tree due to the arbitration of many more predictions at a single arbitration site.

5.1 Empirical Results

Using the two data sets described in Section 4, we performed experiments 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 5 are averages from 10-fold cross-validation runs.

We varied the order of the arbiter trees from two to eight. For the splice junction data set, there is 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 protein coding region domain, 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. 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 experiments was performed to double the maximum training set size for the arbiters. As we observe in Figure 5, by doubling the arbiter training set size, the original accuracy is roughly maintained by the binary trees in

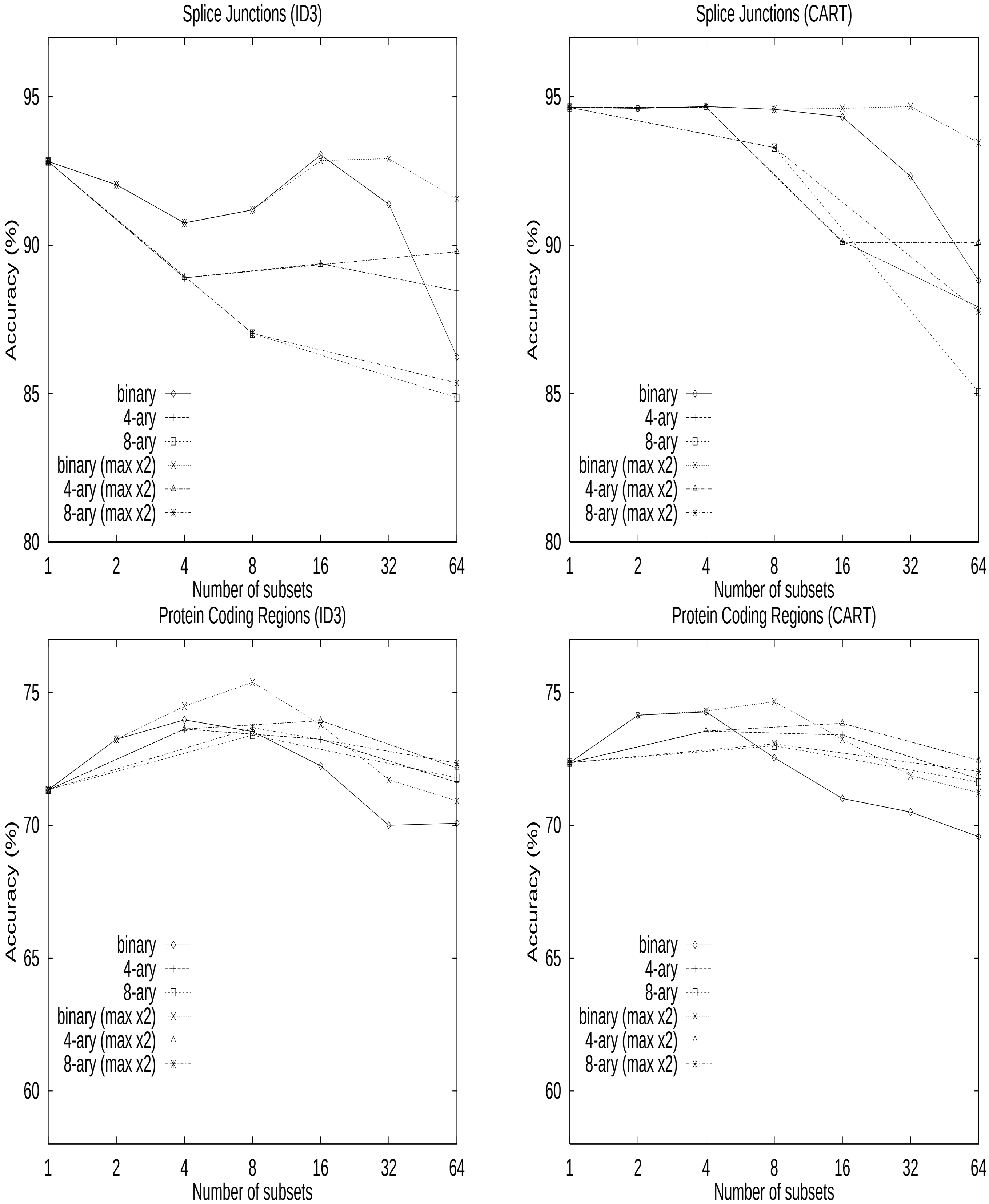

Figure 5: Accuracy for the arbiter tree techniques.

the splice junction domain, regardless of the learner. For 4-ary and 8-ary trees, the accuracy results show no significant improvement. However, more importantly, when the data set is partitioned, this multi-level arbiter tree approach does demonstrate an accuracy improvement over the *one-level* techniques, which might not maintain the accuracy obtained from the whole data set in our experiments.

6 Discussion

Incremental learning algorithms have been proposed that allow the flexibility of not requiring all training examples to be inspected at once. However, some incremental algorithms do require the storage of all examples for future examination during learning, for example, ID5 (Utgoff, 1989). That is, these incremental learning algorithms still demand that all examples fit in the main memory, which is not plausible for massive amounts of data. For those incremental algorithms that do not require all examples to be resident in memory, like neural nets, many demand multiple passes over the data to achieve convergence, which usually consumes substantial processing time. Incremental IBL (Aha & Kibler, 1989) makes only one pass over the data and stores only a subset of the training examples; however, it does not bound the number of examples retained during training.

Quinlan (1979) approached the problem of scaling with a *windowing* technique. Wirth and Catlett (1988) 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. Using a NASA data set, Catlett (1991) demonstrates that larger amounts of data improve accuracy, but the time for ID3 to process a million records might take several months on a MIPS workstation, which is intolerably slow. In addition, windowing does speed up computation considerably for this data set, but introducing noise to the data slows down learning dramatically.

Another approach to solving the scaling problem is simply to increase the number of processors and available memory, parallelize the learning algorithms and apply the parallelized algorithm to the entire data set. Zhang et al.'s (1989) work 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 parallel architecture. Our approach is to run the serial code on a number of data subsets in parallel and combine the results with meta-learning thus reducing and limiting the amount of data inspected by any one learning process (Chan & Stolfo, 1993d). This approach has the advantage of using the same serial code without the time-consuming process of parallelizing it. Since the meta-learning framework for combining the results of learned concepts is independent of the learning algorithm, it can be used with different algorithms.

Our arbiter approach has some relation to Schapire's (1990) theoretical work on *hypothesis boosting* under the PAC learning model. Based on an initial learned hypothesis for some concept derived from a random distribution of training data, Schapire successively generates two additional distributions of examples, to which the learning algorithm is then applied. Half of the examples in the first additional distribution is classified incorrectly by the initial hypothesis. Examples in the second additional distribution are classified differently by the hypotheses learned from the first two distributions. Unlike Schapire's approach, the first two distributions in our arbiter approach are generated independently (concurrently if desired) and the third distribution can be smaller than the first two.

Lastly, our *class-combiner* technique is very similar to Zhang et al.'s (1992) and Wolpert's (1992) work. However, their work focuses on improving accuracy by employing multiple learning algorithms.

7 Concluding Remarks

The moral of the story is simply that attacking the scaling problem by data reduction does have a negative impact on accuracy. However, among the combining schemes, empirical results presented in this paper show that our *meta-learning* strategies can outperform the other more common *one-level* voting-based techniques. The results also demonstrate that training on randomly sampled subsets of examples, without combining, is definitely not sufficient to maintain high accuracy. In the two domains we studied, the one-level meta-learning schemes cannot generally achieve our goal of maintaining the accuracy across different numbers of subsets. To solve this problem, we proposed a more sophisticated meta-learning strategy, called *arbiter tree*. Results from our investigation show that the arbiter tree approach is viable in sustaining the same level of accuracy as a base classifier given the entire data set. Furthermore, our techniques are also data and algorithm-independent, which enable any learning algorithm to train on large data sets (preliminary results from another data set and two other algorithms are reported in (Chan & Stolfo, 1993d).)

We are investigating meta-learners that are specialized in combining decisions. Learners that search M-of-N concepts and other counting-related decision rules might be useful in locating effective combining rules. We are also studying the use of multiple learning algorithms in generating base classifiers to improve the overall prediction accuracy (Chan & Stolfo, 1993c). Moreover, the arbiter tree approach can also be applied to combiners to generate *combiner trees* (Chan & Stolfo, 1995).

Acknowledgements

We thank the anonymous reviewers for their comments. This work has been partially supported by grants from NSF (IRI-94-13847 and CDA-90-24735), New York State Science and Technology Foundation, and Citicorp.

References

Aha, D. & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. *Proc. IJCAI-89* (pp. 794–799).

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Buntine, W. & Caruana, R. (1991). *Introduction to IND and Recursive Partitioning*. NASA Ames Research Center.

Catlett, J. (1991). Megainduction: A test flight. *Proc. Eighth Intl. Work. Machine Learning* (pp. 596–599).

Chan, P. & Stolfo, S. (1993a). Experiments on multistrategy learning by meta-learning. *Proc. Second Intl. Conf. Info. Know. Manag.* (pp. 314–323).

Chan, P. & Stolfo, S. (1993b). Meta-learning for multistrategy and parallel learning. *Proc. Second Intl. Work. on Multistrategy Learning* (pp. 150–165).

Chan, P. & Stolfo, S. (1993c). Toward multistrategy parallel and distributed learning in sequence analysis. *Proc. First Intl. Conf. Intel. Sys. Mol. Biol.* (pp. 65–73).

Chan, P. & Stolfo, S. (1993d). Toward parallel and distributed learning by meta-learning. *Working Notes AAAI Work. Know. Disc. Databases* (pp. 227–240).

Chan, P. & Stolfo, S. (1995). Learning arbiter and combiner trees from partitioned data for scaling machine learning. *Proc. Intl. Conf. Knowledge Discovery and Data Mining*. To appear.

Craven, M. & Shavlik, J. (1993). Learning to represent codons: A challenge problem for constructive induction. *Proc. IJCAI-93* (pp. 1319–1324).

Littlestone, N. & Warmuth, M. (1989). *The weighted majority algorithm*. (Technical Report UCSC-CRL-89-16): Univ. Cal., Santa Cruz.

Quinlan, J. R. (1979). *Induction over large data bases*. (Technical Report STAN-CS-79-739): Comp. Sci. Dept., Stanford Univ.

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

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5, 197–226.

Towell, G., Shavlik, J., & Noordewier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. *Proc. AAAI-90* (pp. 861–866).

Utgoff, P. (1989). Incremental induction of decision trees. *Machine Learning*, 4, 161–186.

Wah, B. (1993). 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.

Wirth, J. & Catlett, J. (1988). Experiments on the costs and benefits of windowing in ID3. *Proc. Fifth Intl. Conf. Machine Learning* (pp. 87–99).

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, 5, 241–259.

Xu, L., Krzyzak, A., & Suen, C. (1992). Methods of combining multiple classifires and their applications to handwriting recognition. *IEEE Trans. Sys. Man. Cyb.*, 22, 418–435.

Zhang, X., Mckenna, M., Mesirov, J., & Waltz, D. (1989). *An Efficient Implementation of the Backpropagation Algorithm on the Connection Machine CM-2*. (Technical Report RL89-1): Thinking Machines Corp.

Zhang, X., Mesirov, J., & Waltz, D. (1992). A hybrid system for protein secondary structure prediction. *J. Mol. Biol.*, 225, 1049–1063.