Learning Arbiter and Combiner Trees from Partitioned Data for Scaling Machine 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

Knowledge discovery in databases has become an increasingly important research topic with the advent of wide area network computing. One of the crucial problems we study in this paper is how to scale machine learning algorithms, that typically are designed to deal with main memory based datasets, to efficiently learn from large distributed databases. We have explored an approach called *meta-learning* that is related to the traditional approaches of data reduction commonly employed in distributed query processing systems. Here we seek efficient means to learn how to *combine* a number of *base classifiers*, which are learned from subsets of the data, so that we scale efficiently to larger learning problems, and boost the accuracy of the constituent classifiers if possible. In this paper we compare the *arbiter tree* strategy to a new but related approach called the *combiner tree* strategy.

Introduction

With the coming age of large-scale network computing, it is likely that orders of magnitude more data in databases will be available for various learning problems of real world importance. Financial institutions and market analysis firms have for years attempted to learn simple categorical classifications of their potential customer base, i.e., relevant patterns of attribute values of consumer data that predict a low-risk (high profit) customer versus a high-risk (low-profit) customer. Many corporations seeking similar added value from their databases are already dealing with overwhelming amounts of global information that in time will likely grow in size faster than available improvements in machine resources. 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 are inherently distributed and cannot be localized on any one machine (even by a trusted third party) for competitive business reasons, as well as statutory constraints imposed by government. In such situations, it may not be possible, nor feasible, to inspect all of the data at one processing site to compute one primary "global" classifier.

*This work has been partially supported by grants from NSF (IRI-94-13847 and CDA-90-24735), NYSSTF, and Citicorp.

Incremental learning algorithms and windowing techniques aim to solve the scaling problem by piecemeal processing of a large data set. Others have studied approaches based upon direct parallelization of a learning algorithm run on a multiprocessor. A review of such approaches has appeared elsewhere (Chan & Stolfo 1995). An alternative approach we study here is to apply *data reduction* techniques common in distributed query processing where cluster of computers can be profitably employed to learn from large databases. This means one may partition the data into a number of smaller *disjoint* training subsets, apply some learning algorithm on each subset (perhaps all in parallel), followed by a phase that combines the learned results in some principled fashion. In the case of inherently distributed databases, each constituent fixed partition constitutes the training set for one instance of a machine learning program that generates one distinct classifier (far smaller in size than the data). The classifiers so generated may be a distributed set of rules, a number of C programs (e.g. "black box" neural net programs), or a set of "intelligent agents" that may be readily exchanged between processors in a network. Notice, however, that as the size of the data set and the number of its partitions increase, the size of each partition relative to the entire database decreases. This implies that the accuracy of each base classifier will likely degrade. Thus, we also seek to boost the accuracy of the distinct classifiers by combining their collective knowledge.

In this paper we study more sophisticated 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. In a previous study (Chan & Stolfo 1995) we demonstrate that our *meta-learning* techniques outperform the voting-based and statistical techniques in terms of prediction accuracy. Here we extend our *arbiter tree* scheme to a related but different scheme called *combiner tree*. We empirically compare the two schemes and discuss the relative merits of each scheme. Surprisingly, we have observed that combiner trees effectively boost the accuracy of the single global classifier that is trained on the entire data set, as well as the constituent base classifiers. (Some of the descriptive material in this paper has appeared in prior publications and is repeated here to provide a self-contained exposition.)

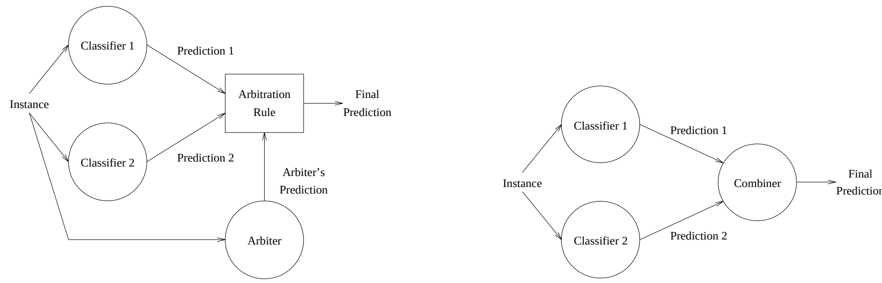

Figure 1: An arbiter and a combiner with two classifiers.

Meta-learning Techniques

Rather than learning weights of some statistical weighting scheme, our approach is to *meta-learn* a set of new classifiers (or *meta-classifiers*) whose training data are sets of predictions generated by a set of base classifiers. *Arbiters* and *combiners* are the two types of meta-classifiers studied here.

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 sometimes 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.

Arbiter An *arbiter* (Chan & Stolfo 1993b) 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 depicts how the final prediction is made with predictions 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 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. 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 the selection rule studied here is as follows:

- An instance is selected if none of the classes 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. 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.

Combiner In the *combiner* (Chan & Stolfo 1993a) 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. In essence 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

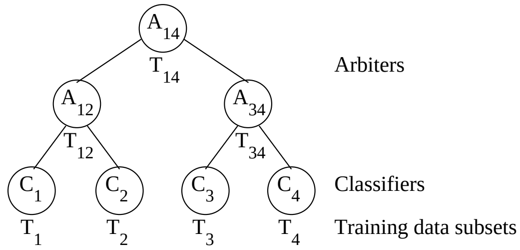

Figure 2: Sample arbiter tree.

as *class-attribute-combiner*.)

Arbiter Tree

In the previous section 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 (the *validation 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 2.

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.

Combiner Tree

The way combiner trees are learned and used is very similar to arbiter trees. A combiner tree is trained bottom-up and classifications also propagate bottom-up. A combiner, instead of an arbiter, is at each non-leaf node of a combiner tree. To simplify our discussion, we describe how a binary combiner tree is used and trained.

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 (the *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 according to a selection rule. Before learning commences, a random set of examples is picked for each level of the combiner tree.[1] 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. According to 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. This process is repeated at each level until the root combiner is created, similar to how an arbiter tree is produced.

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. This might be expensive for certain learning algorithms whose classification time is not relatively insignificant to the training time. The combiner tree approach, however, always classifies at most the size of the meta-level training set. Therefore, combiner trees can be generated more efficiently than arbiter tress in certain cases.

[1]In earlier experiments, for efficiency reasons, we only randomly picked one set of training examples for all levels and the results obtained were not as robust as those reported here.

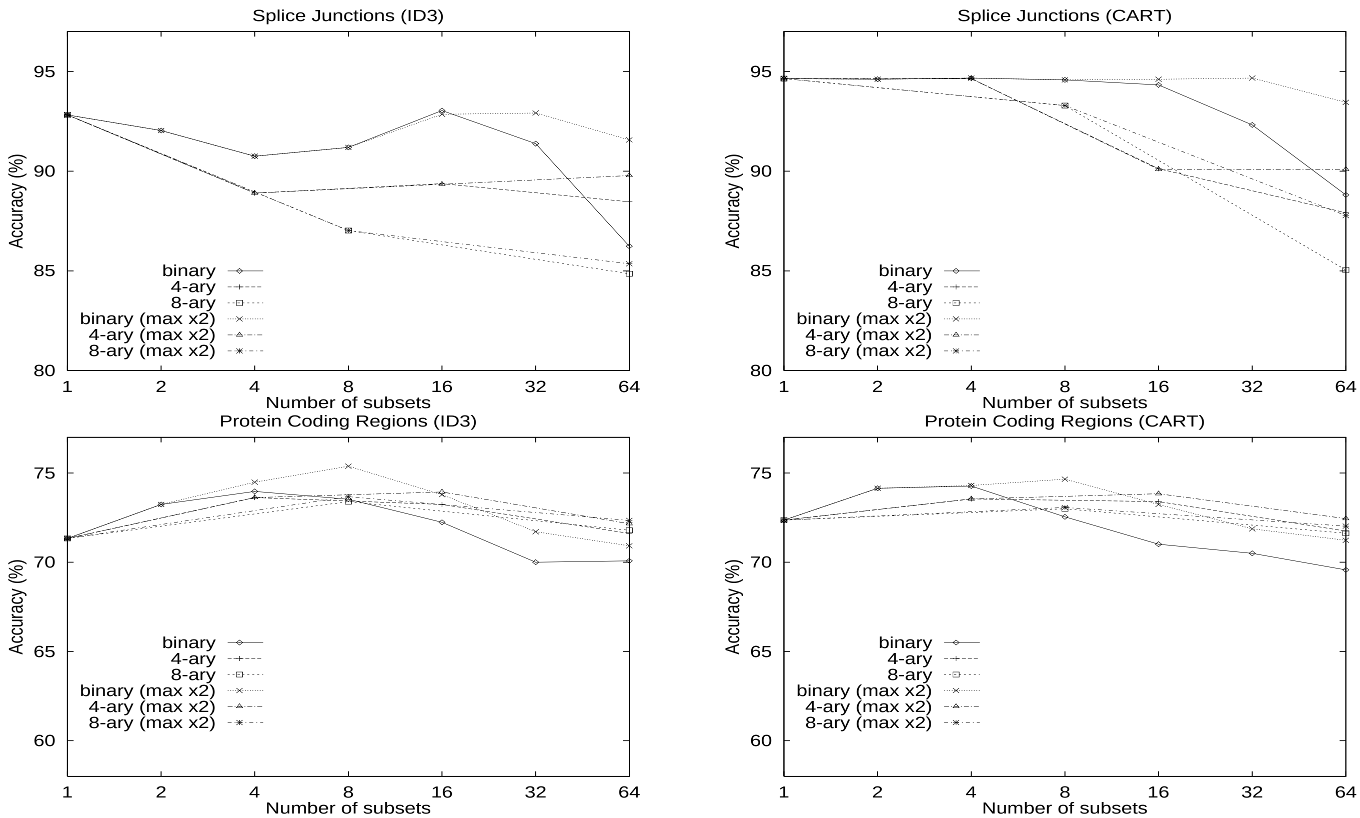

Figure 3: Accuracy for the arbiter tree techniques.

Experiments and Results

Two inductive learning algorithms were used in our experiments reported here. 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 (courtesy of Towell, Shavlik, and Noordewier (1990)) contains sequences of nucleotides and the type of splice junction, if any, at the center of each sequence. *Exon-intron*, *intron-exon*, and *non-junction* are the three classes in this task. Each sequence has 60 nucleotides with eight different values per nucleotide (four base ones plus four combinations). The data set contains 3,190 training instances. The protein coding region (PCR) data set (courtesy of Craven and Shavlik (1993)) contains DNA nucleotide sequences and their binary classifications (*coding* or *non-coding*). Each sequence has 15 nucleotides with four different values per nucleotide. The PCR data set has 20,000 sequences. The two data sets chosen in our experiments represent two different kinds of data sets: one is difficult to learn (PCR at 70+% accuracy) and the other is easy to learn (SJ at 90+%).

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 (i.e., the ratio of examples in each class in the whole data set is preserved). We also varied the order of the arbiter/combiner trees from two to eight. For the combiner trees, both the *class-combiner* and *class-attribute-combiner* strategies were evaluated. 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 arbiter trees are plotted in Figure 3, which were reported in an earlier study (Chan & Stolfo 1995) and are included here for comparison purposes. The results from the combiner trees are in Figure 4. The accuracy for the serial case is plotted as "one subset," meaning the learning algorithms were applied to the entire training set to produce the baseline accuracy results for comparison. The classifier learned from the entire training set is called the *global classifier*. 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.

We first examine the results from the *arbiter tree* strategy. For the splice junction data set, there is a drop in accuracy, compared to the *global classifier*, 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

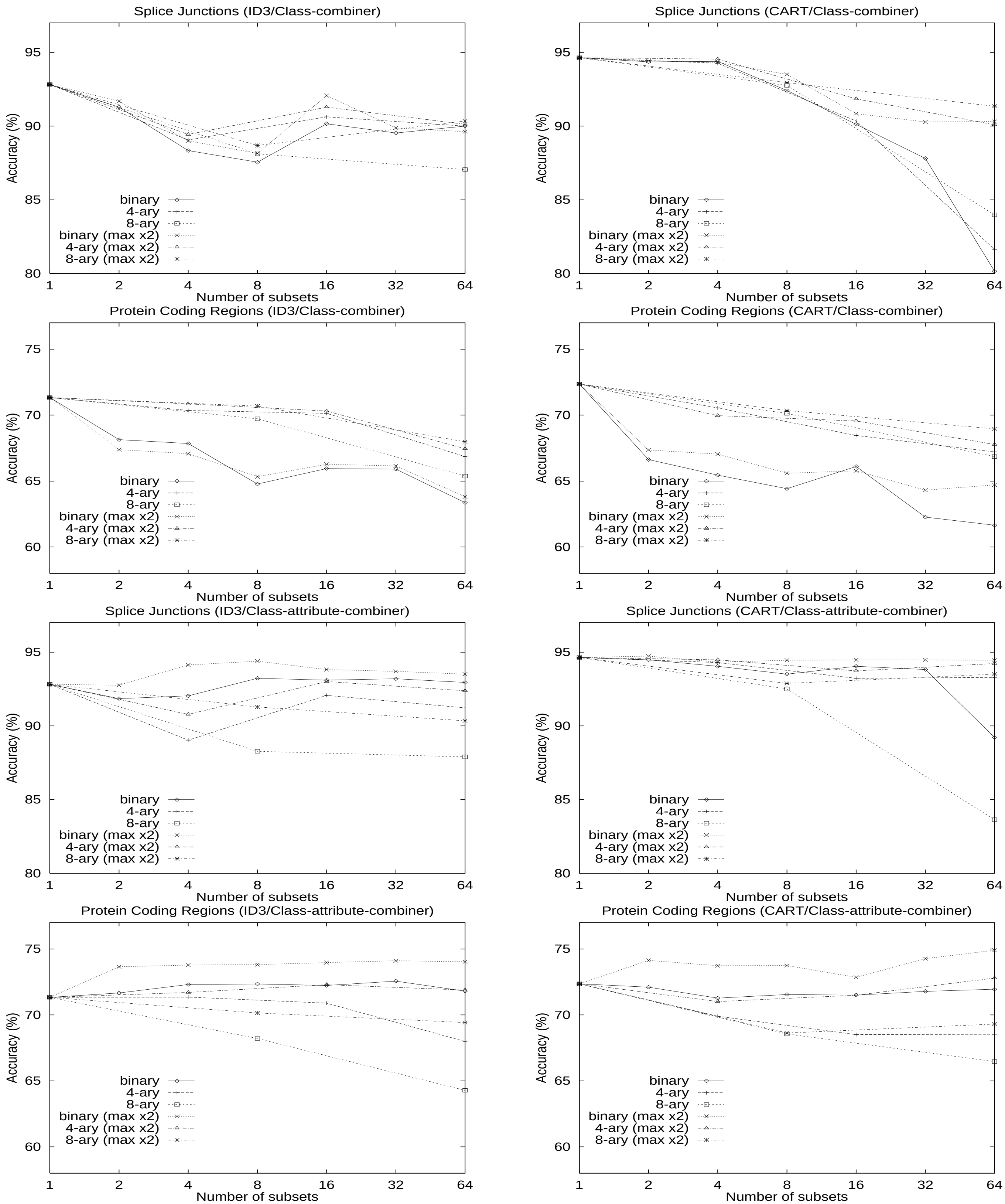

Figure 4: Accuracy for the combiner tree techniques.

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 the initial data subset partition used to train 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 experiments was performed to double the maximum training set size for the arbiters (denoted as “max x2”). As we observe in Figure 3, by doubling the arbiter training set size, the original accuracy is roughly maintained by the binary trees in the splice junction domain, regardless of the learner. The binary trees are generally more accurate, with statistical significance, than higher-order trees in the splice junction domain.

For the *class-combiner tree* strategy (Figure 4), there is 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 binary combiner trees are statistically less accurate than higher order trees. This might be due to the lack of information for finding correlations among only two predictions and the correct classification. As in the experiments for arbiter trees, we doubled the size of meta-level training sets. Statistically significant improvements were observed in the splice junction case with CART as the learner. In general, the class-combiner tree strategy tends toward lower accuracy in most of our experiments.

For the *class-attribute-combiner tree* strategy (Figure 4), the binary trees appear to maintain the accuracy except in the splice junction data set with CART as the learner. Higher-order trees were generally less accurate. Doubling the size of the training sets for combiners improved accuracy. For the protein coding regions data set, the accuracy of the binary trees was consistently higher than that from the global classifiers; 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, we have demonstrated a means of combining the collective knowledge distributed among the individual base classifiers.

In summary, from our experiments, the *class-combiner* tree strategy does not perform as well in maintaining or boosting accuracy as the *arbiter* or *class-attribute-combiner* tree strategies. Relatively less information in the meta-level training sets is likely the contributing factor. Higher order trees are usually less accurate. This is probably due to the decrease in opportunities for correcting predictions when the height of the tree decreases. The relatively poor performance of *one-level* (non-tree) meta-learning techniques compared to the *multi-level* (tree) schemes in our earlier study (Chan & Stolfo 1995) also provides support for this observation. 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 are available for training. The experimental data convincingly demonstrate that doubling the training set size of the meta-level partitions is sufficient to maintain the same level of accuracy as the global classifier, and indeed may boost accuracy as well.

Concluding Remarks

In a previous study (Chan & Stolfo 1995) we demonstrated that the meta-learning strategies outperform the voting-based and statistical techniques reported in the literature. We also showed that the *arbiter tree* approach is viable in sustaining the same level of accuracy as the global classifier learned from the entire data set. Empirical results presented in this paper show that the *class-attribute-combiner* tree strategy can also sustain the accuracy level achieved by the global classifier. In a few cases the global classifier's accuracy was consistently exceeded; i.e., meta-learning can boost the accuracy of a single classifier trained on the entire data set. The combiner tree strategies might also have an advantage of faster tree construction over the arbiter tree strategy for certain learning algorithms. Furthermore, our techniques are also data and algorithm-independent, which enable any learning algorithm to train on large data sets.

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 overall prediction accuracy. Experiments on testing our techniques in a parallel and distributed environment are in progress.

References

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

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

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

Chan, P., and Stolfo, S. 1993b. Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Know. Disc. Databases*, 227–240.

Chan, P., and Stolfo, S. 1995. A comparative evaluation of voting and meta-learning on partitioned data. In *Proc. Twelfth Intl. Conf. Machine Learning*. To appear.

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

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

Towell, G.; Shavlik, J.; and Noordewier, M. 1990. Refinement of approximate domain theories by knowledge-based neural networks. In *Proc. AAAI-90*, 861–866.

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