Toward Multistrategy Parallel and Distributed Learning in Sequence Analysis

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

Machine learning techniques have been shown to be effective in sequence analysis tasks. However, current learning algorithms, which are typically serial main-memory-based, are not capable of handling the vast amounts of information being generated by the Human Genome Project. The *multistrategy parallel learning* approach presented in this paper is an attempt to scale existing learning algorithms. Learning speed is improved through running multiple learning processes in parallel and prediction accuracy is improved through multiple learners. Our approaches are independent of the learning algorithms used. This paper focuses on one of the MSPL approaches and preliminary empirical results that we present are encouraging.[1]

Introduction

Various computer systems have been built to facilitate the process of analyzing amino acid and nucleotide sequences (von Heijne 1987). However, most of the systems require translating analysis techniques developed by humans to programs. It is well known that this process, *knowledge engineering*, can be lengthy and problematic (Boose 1986).

Machine learning is an artificial intelligence technique which allows classification systems to be generated automatically by finding patterns and causal relationships in the data obtained from the user or interactions with the environment. In particular, *inductive learning* aims at discovering relationships in data with little or no knowledge about the data. That is, it is feasible that sequence-analysis systems can be built automatically and directly from exemplar sequence information without obtaining and translating human expertise. Furthermore, machine learning techniques allow the possibility of discovering patterns and concepts unknown to the human experts. It has been reported that some of these systems generated by learning techniques outperform *human-designed* systems (Chan 1991; Qian & Sejnowski 1988; Towell et al. 1990; Zhang et al. 1992).

Recently, several researchers (Wolpert 1992; Zhang et al. 1992) have proposed implementing learning systems by integrating in some fashion a number of different strategies and algorithms to boost overall accuracy by reducing what Mitchell (1980) calls *inductive bias*. The basic notion behind this integration is to complement the different underlying learning strategies embodied by different learning algorithms by effectively reducing the space of incorrect classifications of a learned concept. There are many ways of integrating different learning algorithms. For example, work on integrating inductive and explanation-based learning (Flann & Dietterich 1989) requires a complicated new algorithm that implements both approaches to learning in a single system. However, not much work has been done on combining different learning systems in a loose fashion by essentially learning a new system that knows how to combine the collective outputs of the constituent systems. One advantage of this approach is its simplicity in treating the individual learning systems as black boxes with little or no modification required to achieve a final system. Therefore, individual systems can be added or replaced with relative ease. Some work in this direction has been reported by Wolpert (1992) and Zhang et al. (1992).

Furthermore, 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 will be available for various learning problems of real world importance. A notable example is the enormous amounts of data being generated by the Human Genome Project (DeLisi 1988). The complexity of typical machine learning algorithms renders their use infeasible in problems with massive amounts of data (Chan & Stolfo 1993c). For instance, Catlett (1991) projects that ID3 (Quinlan 1986) on modern machines will take several months to learn from a million records in the flight data set obtained from NASA. In addition, typical learning algorithms like ID3 rely on a monolithic memory to fit all the data. However, it is clear that main memory can easily be exceeded with

[1]This work has been partially supported by grants from New York State Science and Technology Foundation, Citicorp, and NSF CISE.

massive amounts of data.

One approach to this problem is to parallelize the learning algorithms. 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 architecture. Our approach is to run the serial code on a number of data subsets in parallel and combine the results in an intelligent fashion. This approach has the advantage of using the same serial code without the time-consuming process of parallelizing it. Since our 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 a considerable amount of information may not be accessible to each of the learners.

In this paper we present our approaches to *multistrategy parallel learning* by *meta-learning*. Meta-learning is used to coalesce the results from multiple learning systems to improve accuracy as well as combine results from a set of parallel or distributed learning processes to improve speed. We first describe the sequence analysis tasks that we investigated. We then discuss our approach to multistrategy parallel learning by meta-learning and detail our strategies for one of the cases. Preliminary experiments on a serial implementation were conducted; a parallel and distributed implementation is under way. These experiments and their results are reported, followed by a discussion on our findings and work in progress. Some of the detailed strategies described below have appeared elsewhere (Chan & Stolfo 1993a, 1993b) and are provided here for a complete treatment of the subject matter. This paper provides new results of applying meta-learning that extends what was perviously reported. In particular, we apply meta learning over multiple learners on disjoint subsets of training data for increased learning speed.

Sequence Analysis Tasks

Molecular biologists have been focusing on analyzing sequences obtained from proteins, DNA, and RNA strands. Two simplified sequence analysis tasks were investigated in our experiments: *protein secondary structure prediction* and *RNA splice junction prediction*. Although the data sets mentioned below are not very large, they give us an idea on how our strategies perform.

Protein Secondary Structures Scientists have identified structural patterns in proteins and classified basically three structural levels. The primary structure is the sequence of amino acids, a linear chain of specific acids. The main secondary structures are three-dimensional structures formed from this linear sequence called *α-helix*, *β-sheet*, and *coil*. Groups of secondary structures, as well as primary structures, produce tertiary structures.

There have been some research on using machine learning techniques to identify protein secondary structures from amino acid sequences. The task is to learn the rules governing the formation of, say an α-helix, given a particular amino acid sequence. For our experiments the secondary protein structure data set (SS), obtained from the UCI Machine Learning Database, courtesy of Qian and Sejnowski (1988), contains sequences of amino acids and the secondary structures at the corresponding positions. There are three structures (classes) and 20 amino acids (21 attributes because of a spacer (Qian & Sejnowski 1988)) in the data. The amino acid sequences were split into shorter sequences of length 13 according to a windowing technique used in (Qian & Sejnowski 1988). The sequences were then divided into a disjoint training and test set, according to the distribution described in (Qian & Sejnowski 1988). The training set, for this task has 18105 instances and the test set has 3520.

RNA Splice Junctions In eukaryotes' DNA, there are *interrupted* genes. That is, some regions of a gene do not encode protein information. During *transcription*, these non-protein-encoding regions are passed to the RNA. These regions on an RNA strand are called *introns* and are sliced off during *translation*, the process of decoding information on an RNA strand to generate proteins. Before translation begins, the regions that encode protein information, *exons*, are spliced together after the introns are removed.

Given a nucleotide sequence, the task is to identify splice junctions (intron-exon or exon-intron junctions). For our experiments, the RNA splice junction data set (SJ), also obtained from the UCI Machine Learning Database, 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 (three classes). Each sequence has 60 nucleotides with eight 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.

Multistrategy Parallel Learning by 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. In this case meta-learning 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 different learners to improve prediction accuracy and speed. We call this approach *multistrategy parallel learning*. The dual objectives here are to improve accuracy using multiple algorithms and to speed up the learning process by parallel and distributed processing in a *divide-and-conquer* fashion. Multiple learning algorithms are used on different subsets of the data and meta-learning is applied on the results from the different subsets. *Multistrategy parallel learning (MSPL)* is a combination of two other proposed approaches: *parallel learning (PL)* (described in (Chan & Stolfo 1993c)) and *multistrategy hypothesis boosting (MSHB)* (described in (Chan & Stolfo 1993a)). *Parallel learning* focuses on speeding up the learning process using only one strategy, while *multistrategy hypothesis boosting* concentrates on improving prediction accuracy using multiple learning algorithms in a serial fashion. Not much research has been attempted by others in *multistrategy parallel learning*, to our knowledge. Similar ideas were first proposed in (Stolfo et al. 1989) in the domain of speech recognition.

The following sections briefly describe our approaches to the MSPL framework detailed in (Chan & Stolfo 1993b) and for the rest of the paper we focus on the second approach (*coarse-subset MSPL*). There are three general approaches we have explored in achieving these goals that differ in the amount of data used and the manner in which the learning algorithms are applied.

Coarse-grain diversity with the entire training set

For experimental purpose, we first consider an approach in which each learning algorithm is applied to the entire training data set. The learners are run concurrently and each follows the *parallel learning (PL)* framework and each learner produces an *arbiter tree* as described in (Chan & Stolfo 1993c). The predictions of the arbiter trees on the training set become part of the training examples for the combiner in the MSHB framework. Since the training of the combiner has to be performed after the arbiter trees are formed, all the processors used to train the arbiter trees will be available for training the combiner. Again, the combiner is trained using the PL framework, which generates another arbiter tree. During the classification of an instance, predictions are generated from the learned arbiter trees and are coalesced by the learned combiner (an arbiter tree itself). Figure 1(a) schematically depicts the arbiter trees formed. Each triangle represents a learned arbiter tree. L_1, L_2, and L_3 represent the three different learners, and L_c is the learning algorithm used for the combiner. E is the original set of training examples. For further reference, this approach is denoted as *coarse-all MSPL*.

Coarse-grain diversity with data subsets

This approach is similar to the previous one except that each learner is applied concurrently to a different data subset, rather than the entire data set; i.e., here we apply data reduction. The data set is divided into l subsets, where l is the number of learning algorithms available, and a different learning algorithm is applied to each subset. The PL framework is then applied to each algorithm-subset pair. As a result, l arbiter trees will be formed. The predictions of the arbiter trees on the training set become part of the training examples for the combiner in the MSHB framework. Similar to the previous approach, the combiner is generated as another arbiter tree. When an instance is classified, the learned arbiter trees first produce their predictions, which are then coalesced by the learned combiner. Figure 1(b) schematically depicts the arbiter trees formed. L_1, L_2, and L_3 represent the different learning algorithms and L_c is the learning algorithm that computes the combiner. E_1, E_2, and E_3 are subsets of the original set of training examples. This approach is denoted as *coarse-subset MSPL*.

Fine-grain diversity

In this approach the data set is divided into p subsets, where p is the number of processors available, and a different learning algorithm is applied to each subset. The subsets are paired and an arbiter tree is formed in a similar fashion as for the single-strategy arbiter tree. Rather than using the same algorithm for training a node and its two children in an arbiter tree, as in the PL framework, different algorithms are used. This generates only one arbiter tree rather than the multiple arbiter trees in the previous two approaches. When an instance is classified, the prediction of the learned arbiter tree is the final prediction. Figure 1(c) schematically depicts the arbiter tree formed. Again, L_1, L_2, and L_3 represent the different learners. E is the original set of training examples. This approach is denoted as *fine-grain MSPL*.

Issues in Coarse-subset MSPL

Load balancing is essential in minimizing the overall training time due to the variance in completion times of different algorithms. However, in *coarse-subset MSPL* we have to determine how to allocate the data subsets as well as the processors. One approach is to evenly distribute the data among the learners and allocate processors according to their relative speeds. Another approach is that each learner has the same number of processors and data are distributed accordingly. That is, we have to decide whether we allocate a uniform

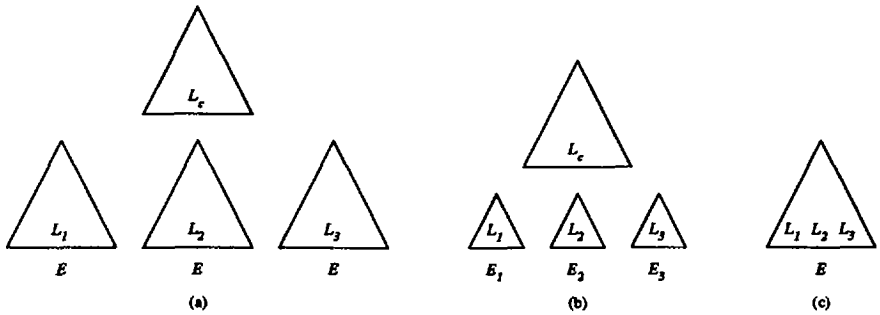

Figure 1: Learned arbiter trees from the MSPL approaches with three different learners

number of processors or a uniform amount of data to each learner. Since the amount of data affects the quality of the learned concepts, it is more desirable to evenly distribute the data so that the learners are not biased at this stage. That is, slower learners should not be penalized with less information and thus they should be allocated more processors.

This raises the question of whether data should be distributed at all; that is, should each learner have all the data (i.e., *coarse-all MSPL*)? Obviously, if each learning algorithm has the entire set of data, it would be slower than when it has only a subset of the data. It is also clear that the more data each learner has, the more accurate the generated concepts will be. Thus, there is a tradeoff between speed and quality. But in problems with very large databases, we may have no choice but to distribute subsets of the data. Another question is what the data distribution is for the data subsets. The subsets can be disjoint or overlapped according to some scheme. We prefer disjoint subsets because it allows the maximum degree of parallelism. The classes represented in the subsets can be distributed randomly, uniformly, or according to some scheme. Since maintaining the same class distribution in each subset as in the entire set does not create the potential problem of missing classes in certain subsets, it is our preferred distribution scheme.

Detailed Strategies for Coarse-subset MSPL

The objective here is to improve prediction accuracy and speed by exploring the diversity of multiple learning algorithms on data subsets through meta-learning. This is achieved by a basic configuration which has several different *base learners* and one *meta-learner* that learns from the output of the base leaners. The meta-learner may employ the same algorithm as one of the base learners or a completely distinct algorithm. Each of the base learners is provided with the entire training set of raw data. However, the training set for the meta-learner varies according to the strategies described below, and is quite different than the data used in training the base classifiers. Note that the meta-learner does not aim at picking the "best" classifier generated by the base learners; instead it tries to combine the classifiers. That is, the prediction accuracy of the overall system is not limited to the most accurate base learner. It is our intent to generate an overall system that outperforms the underlying base learners. The strategies discussed here are comparable to those for multistrategy hypothesis boosting (MSHB) described in (Chan & Stolfo 1993a).

There are in general two types of information the meta-learner can combine: the learned *base classifiers* and the *predictions* of the learned base classifiers. The first type of information consists of *concept descriptions* in the base classifiers (or concepts). Some common concept descriptions are represented in the form of decision trees, rules, and networks. Since we are aiming at diversity in the base learners, the learning algorithms chosen usually have different representations for their learned classifiers. Hence, in order to combine the classifiers, we need to define a common representation to which the different learned classifiers are translated. However, it is difficult to define such a representation to encapsulate all the representations without losing a significant amount of information during the translation process. For instance, it is very difficult to define a common representation to integrate the discriminant functions and exemplars computed by a nearest-neighbor learning algorithm with the tree computed by a decision tree learning algorithm. Because of this difficulty, one might define a uniform representation that limits the types of representation that can be supported and hence the choice of learning algorithms.

An alternative strategy is to integrate the *predictions* of the learned classifiers for the training set leaving the internal organization of each classifier completely transparent. These predictions are hypothesized classes present in the training data and can be categorical or associated with some numeric measure (e.g., probabilities, confidence values, and distances). In this case, the problem of finding a common ground is much less severe. For instance, classes with numeric measures can be treated as categorical (by picking the class with the highest value). Since any learner can be employed in this case, we focus our work on combining predictions from the learned classifiers. Moreover, since converting categorical predictions to ones with numeric measures is undesirable or impossible, we concentrate on combining categorical predictions.

We experimented with three types of meta-learning strategies (*combiner*, *arbiter*, and *hybrid*) for combining predictions, which we discuss in the following sections. For pedagogical reasons, our discussion assumes three base learners and one meta-learner.

Combiner strategy

In the *combiner* strategy, the predictions for the training set generated by a two-fold cross validation technique using the base learners form the basis of the meta-learner's training set (details in (Chan & Stolfo 1993a)). 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. 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.

We experimented with three schemes for the composition rule. First, three predictions, $C_1(x)$, $C_2(x)$, and $C_3(x)$, for each example x in the original training set of examples, E, are generated by three separate classifiers, C_1, C_2, and C_3. 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 according to the schemes as defined below. In the following definitions, $class(x)$ denotes the correct classification of example x as specified in training set E.

1. Return meta-level training instances with the correct classification and the predictions; i.e., $T = \{(class(x), C_1(x), C_2(x), C_3(x)) \mid x \in E\}$. This scheme was also used by Wolpert (1992). (For further reference, this scheme is denoted as *meta-class*.)
2. Return meta-level training instances similar to those in the first (*meta-class*) scheme with the addition of the original attribute vectors in the training examples; i.e., $T = \{(class(x), C_1(x), C_2(x), C_3(x), attr\text{-}vec(x)) \mid x \in E\}$. (Henceforth, this scheme is denoted as *meta-class-attribute*.)
3. Return meta-level training instances similar to those in the *meta-class* scheme except that each prediction, $C_i(x)$, has m binary predictions, $C_{i_1}(x), \ldots, C_{i_m}(x)$, where m is the number of classes. Each prediction, $C_{i_j}(x)$, is produced from a binary classifier, which is trained on examples that are labeled with classes j and $\neg j$. In other words, we are using more specialized base classifiers and attempting to learn the correlation between the binary predictions and the correct prediction. For concreteness, $T = \{(class(x), C_{1_1}(x), \ldots, C_{1_m}(x), C_{2_1}(x), \ldots, C_{2_m}(x), C_{3_1}(x), \ldots, C_{3_m}(x)) \mid x \in E\}$. (Henceforth, this scheme is denoted as *meta-class-binary*.)

These three schemes for the composition rule are defined in the context of forming a training set for the combiner. These composition rules are also used in a similar manner during classification after a combiner has been computed. Given a test instance whose classification is sought, we first compute the classifications predicted by each of the base classifiers. The composition rule is then applied to generate a single meta-level test instance, which is then classified by the combiner to produce the final predicted class of the original test instance.

Arbiter strategy

In the *arbiter* strategy, the training set for the meta-learner is a subset of the training set for the base learners. That is, the meta-level training instances are a particular distribution of the raw training set E. The predictions of the learned base classifiers for the training set and a *selection rule*, which varies in different schemes, determines which subset will constitute the meta-learner's training set. (This contrasts with the *combiner* strategy which has the same number of examples for the base classifier as for the combiner. Also, the meta-level instances of the combiner strategy incorporate additional information than just the raw training data.) Based on this training set, the meta-learner generates a meta-classifier, in this case called an *arbiter*. In classifying an instance, the base classifiers first generate their predictions. These predictions, together with the arbiter and a corresponding *arbitration rule*, generate the final predictions. In this strategy one learns to arbitrate among the potentially different predictions from the base classifiers, instead of learning to coalesce the predictions as in the combiner strategy. We first describe the schemes for the selection rule and then those for the arbitration rule.

We experimented with two schemes for the *selection rule*, which chooses training examples for an arbiter. In essence the schemes select examples that are confusing to the three base classifiers, from which an arbiter is learned. Based on three predictions, $C_1(x)$, $C_2(x)$, and $C_3(x)$, for each example x in a set of training examples, E, each scheme generates a set of training examples, T ($\subseteq E$), for the arbiter. The two versions of this selection rule implemented and reported here include:

1. Return instances with predictions that disagree; i.e., $T = T_d = \{x \in E \mid (C_1(x) \neq C_2(x)) \vee (C_2(x) \neq C_3(x))\}$. Thus, instances with conflicting predictions are used to train the arbiter. However, instances with predictions that agree but are incorrect are not included. (We refer to this scheme as *meta-different*.)
2. Return instances with predictions that disagree, T_d, as in the first case (*meta-different*), but also instances with predictions that agree but are incorrect; i.e, $T = T_d \cup T_i$, where $T_i = \{x \in E \mid (C_1(x) =$

$C_2(x) = C_3(x)) \wedge (class(x) \neq C_1(x))\}$. Note that we compose both cases of instances that are incorrectly classified or are in disagreement. (Henceforth, we refer to this case as *meta-different-incorrect*.)

The arbiters are trained by some learning algorithm on the particular distinguished distributions of training data and are used in generating predictions. During the classification of an instance, y, the learned arbiter, A, and the corresponding *arbitration rule*, produce a final prediction based on the three predictions, $C_1(y)$, $C_2(y)$, and $C_3(y)$, from the three base classifiers and the arbiter's own prediction, $A(y)$. The following arbitration rule applies to both schemes for the selection rule described above.

- Return the simple vote of the base and arbiter's predictions, breaking ties in favor of the arbiter's prediction; i.e., if there are no ties, return $vote(C_1(y), C_2(y), C_3(y), A(y))$, otherwise return $A(y)$.

Hybrid strategy

We integrate the *combiner* and *arbiter* strategies in the *hybrid* strategy. Given the predictions of the base classifiers on the original training set, a *selection rule* picks examples from the training set as in the arbiter strategy. However, the training set for the meta-learner is generated by a *composition rule* applied to the distribution of training data (a subset of E) as defined in the combiner strategy. Thus, the hybrid strategy attempts to improve the arbiter strategy by correcting the predictions of the "confused" examples. It does so by using the combiner strategy to coalesce the predicted classifications of instances in disagreement by the base classifiers, instead of purely arbitrating among them. A learning algorithm then generates a meta-classifier from this training set. When a test instance is classified, the base classifiers first generate their predictions. These predictions are then composed to form a meta-level instance for the learned meta-classifier using the same composition rule. The meta-classifier then produces the final prediction.

We experimented with two combinations of composition and selection rules, though any combination of the rules is possible:

1. Select examples that have different predictions from the base classifiers and the predictions, together with the correct classes and attribute vectors from the training set for the meta-learner. This integrates the *meta-different* and *meta-class-attribute* schemes. (Henceforth, we refer to this scheme as *meta-different-class-attribute*.)
2. Select examples that have different or incorrect predictions from the base classifiers and the predictions, together with the correct classes and attribute vectors form the training set for the meta-learner. This integrates the *meta-different-incorrect* and *meta-class-attribute* schemes. (Henceforth, denoted as *meta-different-incorrect-class-attribute*.)

We discussed three general meta-learning strategies (*combiner*, *arbiter*, and *hybrid*) for *multistrategy parallel learning*. The combiner strategy aims at coalescing predictions from the constituent classifiers, whereas the arbiter strategy arbitrates among them. In addition, the training set for the combiner strategy includes examples derived from the entire original training set, whereas the one for the arbiter includes only examples chosen by a selection rule from the original set. That is, the training set for the arbiter strategy is usually smaller than the one for the combiner strategy and hence contains less information. The hybrid strategy is intended to augment this deficiency in the arbiter strategy by coalescing the predictions from the selected examples. We postulate that the combiner strategy would still be the most effective one due to the larger amount of information available and the coalescing process.

Among the combiner schemes, the *meta-class-attribute* scheme provides more information (the addition of attribute vectors) than the *meta-class* scheme in the combiner training set. The *meta-class-binary* scheme provides more precise information for the meta-learner because more specialized base classifiers are used. Among the arbiter schemes, the *meta-different-incorrect* scheme includes more examples in the arbiter training set than the *meta-different* scheme. Similar observations can be made for corresponding schemes in the hybrid strategy.

Experiments

Four inductive learning algorithms were used in our experiments. We obtained ID3 (Quinlan 1986) and CART (Breiman et al. 1984) as part of the IND package from NASA Ames Research Center; both algorithms compute decision trees. WPEBLS is the weighted version of PEBLS (Cost & Salzberg 1993), which is a nearest-neighbor learning algorithm. BAYES is a Bayesian classifier that is based on computing conditional probabilities (frequency distributions), which is described in (Clark & Niblett 1987). The latter two algorithms were reimplemented in C.

The base-learners are first trained on the data subsets and the whole training set is then classified by the learned base-classifiers. Since the base-learners are trained on only part of the whole training set, classifying the rest of the set mimics the behavior of the learned classifiers when unseen instances are classified. That is, the meta-learner is partially trained on predictions of unseen instances in the training set. The base-classifiers' predictions on the training set are used in constructing the training set for the meta-learner. To focus our attention on the effectiveness of meta-learning on disjoint subsets, we ran the meta-learner and base learners serially. That is, no arbiter trees were generated.

We performed experiments on the different schemes for the combiner, arbiter, and hybrid strategies. Dif-

Base learners: ID3, CART & WPEBLS

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	53.1	51.8	51.8	53.8
m-c-a	58.0+	50.8	48.5	49.1
m-c-b	52.8	52.4	52.2	52.2
m-d	55.2+	54.4+	54.1+	54.3+
m-d-i	55.2+	53.7	54.5+	54.1+
m-d-c-a	55.5+	54.9+	54.4+	54.3+
m-d-i-c-a	55.0	54.1+	54.1+	54.0+
m-c-w	56.5+	54.7+	51.8	53.4
vote	54.7+	+ better than the 3 base classifiers (subsets)		
freq	51.9	* better than all 4 classifiers (subsets)		

Base learners: BAYES, ID3 & CART

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	57.9	53.8	54.5	54.9
m-c-a	58.8	54.5	50.9	52.2
m-c-b	57.9	57.4	57.2	54.8
m-d	59.5	58.8	58.5	58.1
m-d-i	59.0	58.6	58.5	58.4
m-d-c-a	59.1	58.8	58.6	58.6
m-d-i-c-a	58.9	58.8	58.6	58.9
m-c-w	58.0	55.8	53.0	53.6
vote	58.5			
freq	57.1			

Base learners: BAYES, ID3 & WPEBLS

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	54.7	53.8	53.4	56.5
m-c-a	59.7	53.0	49.0	50.3
m-c-b	57.4	54.8	54.9	54.4
m-d	58.0	57.6	57.2	57.5
m-d-i	57.9	57.4	57.5	57.4
m-d-c-a	57.8	57.8	57.6	57.7
m-d-i-c-a	58.0	57.8	57.6	57.5
m-c-w	59.4	56.8	53.6	53.4
vote	57.8			
freq	54.0			

Base learners: BAYES, CART & WPEBLS

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	55.7	54.0	53.1	54.9
m-c-a	59.3	53.0	49.7	49.8
m-c-b	55.3	52.4	53.9	54.3
m-d	57.2	56.8	56.8	56.6
m-d-i	57.0	56.5	56.6	56.3
m-d-c-a	57.3	56.9	56.8	56.7
m-d-i-c-a	57.0	56.4	56.4	56.5
m-c-w	59.2	55.3	53.4	55.2
vote	57.2			
freq	54.0			

Table 1: Summary of Secondary Structure Prediction Accuracy (%)

ferent combinations of three base and one meta-learner were explored on the two data sets and the results are presented in Tables 1 and 2. Each table has four subtables and each subtable presents results from a different combination of base learners. The first column of a subtable denotes the different schemes and the next four columns denote the four different meta-learners. The names of meta-learning schemes are abbreviated in the tables: *m-c* represents *meta-class*, *m-c-a* represents *meta-class-attribute*, and so on. Results for the two data sets with single-strategy classifiers are displayed in Table 3. In addition, we experimented with a windowing scheme used in Zhang et al.'s (1992) work, which is specific to the SS data. This scheme is similar to the *meta-class* scheme described above. However, in addition to the three predictions present in one training example for the meta-learner, the guesses on either side of the three predictions in the sequence (*windows*) are also present in the example. We denote this scheme as *meta-class-window* (or *m-c-w* in the tables).

Furthermore, two non-meta-learning approaches were applied for comparison. *Vote* is a simple voting scheme applied to the predictions from the base classifiers. *Freq* predicts the most frequent correct class with respect to a combination of predictions in the training set[2]. That is, for a given combination of predictions (m^c combinations for m classes and c classifiers), *freq* predicts the most frequent correct class in the training data.

Results

There are three ways to analyze the results. First, we consider whether the employment of a meta-learner improves accuracy with respect to the underlying three base classifiers learned on a subset. (The presence of an improvement is denoted by a '+' in the tables.) For the SJ data, improvements were almost always achieved when the combinations of base learners are ID3-CART-WPEBLS (an improvement from 94.1% up to 96.2%) and BAYES-CART-WPEBLS (from 95.7% up to 97.2%), regardless of the meta-learners and strategies. For the SS data, when the combination of base-learners is ID3-CART-WPEBLS, more than half of the meta-learner/strategy combinations achieved higher accuracy than any of the base learners (an improvement from 53.9% up to 58.0%).

Second, we examine whether the use of meta-learning achieves higher accuracy than the most accurate classifier learned from a subset (BAYES in this case). (The presence of an improvement is denoted by a '*' in the tables.) For the SJ data, the *meta-class-attribute* strategy with BAYES as the meta-learner always attained higher accuracy (an improvement from 95.7% up to 97.2%), regardless of the base learners and strategies. For the SS data, all the results did not outperform BAYES as a single base learner.

Third, we study whether the use of meta-learning achieves higher accuracy than the most accurate classifier learned from the full training set (BAYES in this case). (The presence of an improvement is denoted by a '!' in the tables.) For the SJ data, *meta-class-attribute* strategy with BAYES as the meta-learner almost always attained higher accuracy (from 96.4% up to 97.2%), regardless of the base learners and strategies. For the SS data, all the results did not outperform BAYES.

In general, *meta-class-attribute* is the more effective scheme and BAYES is the more successful meta-learner. Therefore, it reinforces our conjecture that

[2] *Freq* was suggested by Wolpert (personal communication).

Base learners: ID3, CART & WPEBLS

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	95.1+	95.0+	95.0+	71.5
m-c-a	96.2+*	94.8+	94.8+	95.0+
m-c-b	95.8+*	96.1+*	95.6+	75.5
m-d	95.1+	95.0+	95.1+	95.1+
m-d-i	95.1+	95.1+	95.1+	95.1+
m-d-c-a	95.1+	95.0+	95.1+	95.1+
m-d-i-c-a	95.1+	95.1+	95.1+	95.1+
vote	95.1+*	+ better than the 3 base classifiers (subsets)		
freq	95.6+*	* better than all 4 classifiers (subsets)		
		! better than all 4 classifiers (full set)		

Base learners: BAYES, ID3 & CART

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	94.5	95.9+*	95.5	73.7
m-c-a	96.7+*!	96.1+*	95.5	95.1
m-c-b	95.5	95.6	95.0	76.0
m-d	95.0	95.0	95.0	95.0
m-d-i	95.0	94.5	94.5	95.0
m-d-c-a	95.0	95.0	95.0	94.8
m-d-i-c-a	95.0	94.8	94.8	94.8
vote	95.0			
freq	95.8+*			

Base learners: BAYES, ID3 & WPEBLS

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	95.8+*	95.5	95.1	72.7
m-c-a	97.0+*!	95.5	95.1	95.6
m-c-b	96.2+*	95.1	95.0	76.5
m-d	95.9+*	95.9+*	95.9+*	95.5
m-d-i	95.9+*	95.9+*	95.9+*	95.9+*
m-d-c-a	95.9+*	95.9+*	95.9+*	95.5+*
m-d-i-c-a	95.9+*	95.8+*	95.8+*	95.8+*
vote	95.9+*			
freq	95.3			

Base learners: BAYES, CART & WPEBLS

Scheme	Meta-learner BAYES	ID3	CART	WPEBLS
m-c	96.7+*!	95.9+*	96.1+*	72.4
m-c-a	97.2+*!	95.9+*	95.9+*	95.8+*
m-c-b	95.9+*	94.8	95.6	93.6
m-d	96.9+*!	96.9+*!	96.9+*!	96.7+*!
m-d-i	96.9+*!	96.9+*!	96.6+*!	96.7+*!
m-d-c-a	96.9+*!	96.9+*!	96.9+*!	96.7+*!
m-d-i-c-a	96.9+*!	96.7+*!	96.7+*!	96.7+*!
vote	96.9+*!			
freq	96.7+*!			

Table 2: Summary of Splice Junction Prediction Accuracy (%)

Data set/Algorithm	BAYES	ID3	CART	WPEBLS
Secondary Structure (SS) (full set)	62.1	55.4	50.8	48.1
Average of 3 subsets	60.2	53.9	49.5	47.2
Splice Junction (SJ) (full set)	96.4	93.9	94.8	94.4
Average of 3 subsets	95.7	88.9	94.1	93.4

Table 3: Single-strategy Prediction Accuracy (%)

coalescing results are more effective than arbitrating among them and predictions alone may not be enough for meta-learning. Compared to results obtained from the MSHB framework (Chan & Stolfo 1993a), smaller improvements were observed here. This is mainly due to the smaller amount of information presented to the base learners. Surprisingly, the hybrid schemes did not improve the arbiter strategies. Also, Zhang's (1992) *meta-class-window* strategy for the SS data did not improve accuracy with the base and meta-learners used here. He uses a neural net, and different Bayesian and exemplar-based learners.

Discussion and Concluding Remarks

The two data sets chosen 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 experiments indicate that some of our meta-learning strategies improve accuracy in the SJ data and are more effective than the non-meta-learning strategies. However, in the SS data, both meta-learning and non-meta-learning strategies are roughly the same. This can be attributed to the quality of predictions from the base classifiers for the two data sets as described in (Chan & Stolfo 1993a). The high percentage of having one or none correct out of three predictions in the SS data set might greatly hinder the ability of meta-learning to work. One possible solution is to increase the number of base classifiers to lower the percentage of having one or none correct predictions.

Unlike Wolpert (1992) and Zhang et al.'s (1992) reports, we present results from all the combinations of presented strategies, base learners, and meta-learners. We have shown that improvements can be achieved consistently with a combination of a meta-learner and collection of base learners across various strategies in both data sets. Similarly, better results were achieved for various combinations of different strategies and meta-learners across all base learners in the SJ data set. Improvements on the already high accuracy obtained from the base learners in the SJ data set reflects the viability of the meta-learning approach.

As mentioned in the previous section, the combiner schemes generally performed more effectively than the arbiter or hybrid schemes. This suggests that coalescing the results is more beneficial than arbitrating among them. In addition, the training set for the combiner strategy includes examples derived from the entire original training set, whereas the one for the arbiter or hybrid strategy includes only examples chosen by a selection rule from the original set. That is, the training set for the arbiter or hybrid strategy is usually smaller than the one for the combiner strategy and hence contains less information. (This lack of information may not be exhibited in larger learning tasks with massive amounts of data.)

Among the combiner schemes, the *meta-class-attribute* scheme performed more effectively than the others. This might be due to the additional information (attribute vectors) present in the training exam-

ples, suggesting that information from the predictions alone is not sufficient to achieve higher prediction accuracy. To our surprise, the *meta-class-binary* scheme did not perform more effectively than the *meta-class* scheme. We postulated that more specialized binary classifiers would provide more precise information for the meta-learner. However, this was not exhibited in our experimental results.

We also postulated that a probabilistic learner like BAYES would be a more effective meta-learner due to the relatively low regularity in the training data for meta-learners. Our empirical results indeed show that BAYES is a more effective meta-learner.

We ran the the meta-learner and base learners serially to concentrate our investigation on the viability of meta-learning over disjoint subsets. The prediction accuracy results indicate the effectiveness of our meta-learning approach. Although we have no relevant timing results of a parallel implementation at this point, it is easy to see that speed can be improved when the base learners are trained on the disjoint training data subsets in parallel. Since the training data are distributed in disjoint subsets, each parallel process (a learning algorithm) applied to each subset need not to communicate with any other process. There are no overheads to pay in communicating or synchronizing while the learning algorithms are computing. (However, communication and synchronization are still needed for each algorithm, which has multiple parallel learning processes.) Compared to meta-learning over the full data set using multiple strategies (Chan & Stolfo 1993a), meta-learning over subsets degraded accuracy only slightly. That is, we can increase learning speed, and pay only a modest decrease in accuracy.

In summary, we tested our approach of multistrategy parallel learning by meta-learning and preliminary results are encouraging. We empirically identified strategies and learning algorithms that are more effective at the meta-level. More experiments will be conducted to ensure our results obtained so far are statistically significant. We are also exploring new meta-learning strategies. The work presented here is a step toward our goal of building a parallel and distributed system that can learn from large amounts of data efficiently and accurately.

Acknowledgements

We thank David Wolpert for many useful and insightful discussions that substantially improved the ideas presented in this paper.

References

Boose, J. 1986. *Expertise Transfer for Expert System Design.* Elsevier, Amsterdam, Netherlands.

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

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

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

Chan, P. and Stolfo, S. 1993a. Experiments on multistrategy learning by meta-learning. Submitted to Intl. Conf. Info. Know. Manag.

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

Chan, P. and Stolfo, S. 1993c. Toward parallel and distributed learning by meta-learning. In *Proc. AAAI Work. on Knowledge Discovery in Databases.* To appear.

Chan, P. 1991. Machine learning in molecular biology sequence analysis. Technical Report CUCS-041-91, Department of Computer Science, Columbia University, New York, NY.

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

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

DeLisi, C. 1988. The human genome project. *American Scientist* 76:488–493.

Flann, N. and Dietterich, T. 1989. A study of explanation-based mehtods for inductive learning. *Machine Learning* 4:187–266.

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

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

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

Stolfo, S.; Galil, Z.; McKeown, K.; and Mills, R. 1989. Speech recognition in parallel. In *Proc. Speech Nat. Lang. Work.* DARPA. 353–373.

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

von Heijne, G. 1987. *Sequence Analysis in Molecular Biology.* Academic Press, San Diego, CA.

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

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