# An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning

Philip Kin-Wah Chan

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences.

Columbia University
1996

# ABSTRACT

## An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning

Philip Kin-Wah Chan

Much of the research in inductive learning concentrates on problems with relatively small amounts of data. With the coming age of ubiquitous 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, especially for applications in data mining. 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. Moreover, data can be inherently distributed across multiple sites on the network and merging all the data in one location can be expensive or prohibitive.

In this thesis we propose, investigate, and evaluate a *meta-learning* approach to integrating the results of multiple learning processes. Our approach utilizes machine learning to guide the integration. We identified two main meta-learning strategies: *combiner* and *arbiter*. Both strategies are independent to the learning algorithms used in generating the classifiers. The combiner strategy attempts to reveal relationships among the learned classifiers' prediction patterns. The arbiter strategy tries to determine the correct prediction when the classifiers have different opinions. Various schemes under these two strategies have been developed. Empirical results show that our schemes can obtain accurate classifiers from inaccurate classifiers trained from data subsets. We also implemented and analyzed the schemes in a parallel and distributed environment to demonstrate their scalability.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

Unquestionably, I am grateful for the continual support and encouragement from my advisor, Prof. Sal Stolfo. Without Sal, this thesis might never finish. Life has its twists on me and Sal has given me the much needed trust and confidence. I came to Columbia to work with Sal and the decision was undoubtedly the right one. Moreover, he gave me the freedom to pursue my research interests. Although he had other projects in progress, he always had time to give me advice and ideas.

Through Sal, grants from National Science Foundation, New York State Science and Technology Foundation, and Citibank provided support for this study.

Profs. Kathy McKeown, Mukesh Dalal, and Alex Tuxhilin (New York University), and Dr. Foster Provost (NYNEX Science and Technology) dutifully served on my thesis committee and gave valuable comments on this document. Discussions with Dr. Dave Wolpert at IBM Almaden Research Center (previously at Santa Fe Institute) yielded improved ideas in this study.

The Department of Computer Science and Columbia University provided many resources for my work. Ashutosh Dutta, Chris Maio, Charles Thayer, and the CS Central Research Facilities staff answered many questions and solved many problems. Alice Cueba, Mel Francis, Germaine L'Eveque, Martha Peres, Renate Valencia, and the CS administrative staff tirelessly helped me with the necessary paperwork. The libraries on campus rarely disappointed and usually amazed me. Columbia never ceased edifying me and feeding my growing curiosity.

Florida Institute of Technology, my current employer, has been understanding and provides the time and resources for me to finish this thesis. Prof. Bill Shoaff, the computer science chair, has been very supportive and understanding.

I had an early interest in machine learning and Prof. Doug Fisher at Vanderbilt

University was responsible for teaching me what the field is all about. He was my first graduate advisor and I was his first graduate student. Prof. Larry Dowdy at Vanderbilt gave me my first research opportunities.

Summer internships at Citibank, GTE Laboratories, and Siemens Corporate Research provided me with valuable research experience. Dr. Schutzer introduced me to the suit-and-tie financial industry. Drs. Robert Weihmayer, Gregory Piatetsky-Shapiro, Chris Matheus, and Shri Goyal generously allowed me to work with them for two summers. If it was not for Drs. George Drastal and Pat Langley, I wouldn't have set foot at the Institute for Advanced Study or driven on Einstein Lane.

I benefited from technical discussions with Hasanat Dewan, Dave Espinosa, Dave Fan, Jason Glazier, Mauricio Hernandaz, Dave Ohsie, Lee Woodbury, and the late Russell Mills. Dave Fan generated some of the experimental results. Sabah Al-Binali, Simon Baker, Shu-Wei Chen, Susil DaSilva, Sam Fenster, Blair MacIntyre, and Bill Yoshimi injected quite a bit of social spirit in my departmental life. My kind roommates Katie Jacobs and George Panagos involved me with many activities. I was also the beneficiary of Katie's culinary talents.

A few memorable friendships fostered from the Barnard-Columbia Education Project. Lori Robertson, Liz Sallinger, and Katy Terry's dedication and hard work continue to encourage me to give more to the community. It was a pleasure working with them. Theresa Knapek sets a respectable and honorable example, which originally inspired and constantly reminds me this important part of my life-long journey.

Andrea Alexander, Ben Bushman, Heather Evans, Brett Helquist, the late Roxanna Glass, Jenny Helvey, Joelle Jugant, Greg Lunt, Ron Nelson, Jeff Sweat, Allison Stander, Mike Thomas, Ned Thomas, Brent Walker, Larry Wampler, Mike Whiting, Heather Willoughby, Julie Wilson, and many others from church gave me friendships that will not be easily forgotten. Bishop Brent Belnap is an inspiration. Our Heavenly Father has been very gracious.

Lap-Ki Chan, a good friend for more than eighteen years, is a constant companion for philosophical discussions. We seem to be running on the same road toward the same place. Joe Cheong, Joe Frisbie, Salina Fung, Edwin Lee provided friendships throughout these years.

The Chu and Lee families in Texas have been generous and kind. They provided much care and help during my college years, for which I will always be grateful.

The Cheung family is my second family in New York City. Mr. Jimmy Cheung, a long time family friend, Mrs. Cheung and their five children, Peter, Teresa, Charlie, Ting-Ting, and Bobby have been giving me much hospitality throughout my stay in New York. I would be on the street if they didn't open their home to me when I first arrived in the city. The weekend stay-overs, trips, and free meals will be missed. The family also introduced me to Pete Lee, who is now a good friend and constantly keeps me amused. Occasional free meals at the Bladwin, Cross, Cutshaw, and Stansifer families in Melbourne are always appreciated.

Lastly, I would like to thank my parents and my two brothers, George and Francis. I wouldn't be who I am if my parents didn't teach me the importance of values and can never repay what they have given me. My brothers tolerated me while we were going up. Although we are thousands of miles apart, our childhood will always be remembered. Thanks also go to Ginnie, my sister-in-law, for her kindness and, with George, for making me a soon-to-be uncle.

Many more remain unnamed, but my memory is lacking and it'll certainly take a book...

*To Peace*

*and*

*To My Parents.*

# Chapter 1

# Introduction

The key to intelligence is the ability to learn. Research in the field of *machine learning* (Carbonell, 1989) attempts to endow computers with this intrinsic capability that exists in all higher-order organisms to one degree or another. Learning can be loosely defined as a process that improves performance of an agent by acquiring knowledge through interactions with a changing environment.

In this thesis research we concentrate on a particular type of learning called *inductive learning* (Michalski, 1983). Given some examples (data) obtained from the environment, inductive learning aims to discover patterns in the examples and form *concepts* that describe the examples. For instance, given some examples of chairs and tables, one can form a concept that suggests that the surface of tables is usually hard, whereas the surface of chairs is usually soft.

There are many desirable characteristics of a learning process. Probably the most important is that it composes concepts that reflect reality. In other words, the concepts should be accurate and predictive. Given an instance that has not been encountered before, the learned concept should be able to correctly identify it. This has been the central issue for most inductive learning research efforts.

Another desirable characteristic is how fast a concept can be learned. It is important that a learning system process examples efficiently. Because of the advancement of computer technology, enormous amounts of data can easily be generated, and with "high-capacity" and "high-speed" networks, these data can be made available widely and quickly. For instance, the Human Genome Project (DeLisi, 1988), initiated by the National Institutes of Health (NIH) and Department of Energy (DOE), aims to map the entire human genome and will inevitably generate orders of magnitude more sequence data than exist today. The HPCC Grand Challenges (Wah, 1993) research efforts will generate more data and faster than ever before. Also, 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. However, much of the research in inductive learning concentrates on problems with relatively small amounts of data. The algorithms developed so far are generally not scalable to large databases as envisaged by the Genome Project. The complexity of typical machine learning algorithms renders their use infeasible in problems with massive amounts of data (Chan & Stolfo, 1993d). A more concrete testimony of the efficiency problem is from Catlett (1991), who projects that ID3 (Quinlan, 1986) (a popular inductive learning algorithm) on modern machines will take several months to learn from a million records in the flight data set obtained from NASA, which is clearly unacceptable.

Moreover, typical learning algorithms like ID3 rely on a monolithic memory to fit all of its training data. However, it is clear that main memory can easily be exceeded with massive amounts of data. 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. Therefore, to efficiently process huge databases, learning algorithms need to be *scalable*. We refer *scalability* as the ability to efficiently process increasing amounts of information, given that a machine has a limited amount of resources. (A more formal definition is in Section 9.2.3) On a single machine, its limited resources can get completely saturated by a learning algorithm when it is presented with large amounts of data, which results in intolerable performance or inability of the algorithm to execute. More importantly, machine learning is central to *knowledge discovery in databases / data mining* (KDD/DM) (Piatesky-Shapiro & Frawley, 1991; Matheus *et al.*, 1993) systems. In most cases research in this area is faced with massive databases. That is, learning systems are facing vast amounts of information and scaling them up is a critical issue facing machine learning research.

In the next section we explore the relationship between inductive learning and other related areas where scalability is an important issue.

## 1.1 Inductive Learning, Knowledge-Based Systems and Data Mining

*Inductive learning* is the task of identifying regularities in some given set of examples with little or no knowledge about the domain from which the examples are drawn. Inductive learning systems process examples that include class labels and generate *concepts* which accurately describe the classes present in the examples.

The learned concepts can also be used as knowledge in knowledge-based systems; learning provides a means for these systems to evolve over time and adapt to changing environments. For instance, in a rule-based expert system each rule consists of an antecedent, which is a pattern matching expression in some symbolic formalism, and a consequent, which specifies the actions to be taken if the antecedent is matched. Hence, each rule can be learned from examples by treating it as a concept to be learned, where the antecedent is the pattern describing the concept and the consequent is its classification. Inductive learning in this context can be viewed as automated knowledge acquisition for building knowledge-based systems. Much as knowledge engineering via human is the bottleneck in knowledge acquisition (Boose, 1986), inefficient machine learning is the bottleneck in automated knowledge acquisition.

Machine learning can be a continual process, as in people, for revising outdated theories in knowledge-based systems (Ourston & Mooney, 1990; Towell & Shavlik, 1993; Brunk & Pazzani, 1995).

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

One means of acquiring new knowledge from databases is to apply various machine learning algorithms that compute descriptive representations of the data as well as patterns that may be exhibited in the data. The field of machine learning has made substantial progress over the years and a number of algorithms have been popularized and applied to a host of applications in diverse fields (Langley & Simon, 1995; Bratko & Muggleton, 1995; Fayyad *et al.*, 1993; Craven & Shavlik, 1994). Thus, we may simply apply the current generation of learning algorithms to very large databases and wait for a response! However, the question is how long might we wait? Indeed, do the current generation of machine learning algorithms **scale** from tasks common today that include thousands of data items to new learning tasks encompassing as much as two orders of magnitude or more of data that is physically distributed? Furthermore, many existing learning algorithms require all the data to be resident in main memory, which is clearly untenable in many realistic databases. In certain cases, data is inherently distributed and cannot be localized on any one machine for a variety of practical reasons. In such situations it is infeasible to inspect all of the data at one processing site to compute one primary "global" classifier. We call the problem of learning useful new knowledge from large inherently distributed databases the *scaling problem for machine learning*.

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

In a *federated* or *integrated multi-database* context, a similar data mining task may be defined over the *universal* relation embodying the constituent databases. Here, however, the problem is more daunting. We presume all component databases of a federated system share common (or at least "provably equivalent") attributes with values drawn from a common domain of values. However, each component relation may include attributes that are unique to that database. In such situations, a data mining or machine learning task applied to the universal relation defined by the constituents would necessarily include *null values* in some tuples. The problem of logically forming the universal relation in preparation for a data mining process is itself a difficult problem studied by a large research community (e.g. (Hernandez & Stolfo, 1995)). For this study, we make the simplifying assumption that the universal relation is available over a distributed set of processing sites. Our focus is on various means that seek to integrate the entirety of distributed data to learn one "global classifier" able to predict unknown values of some desired attribute, or to classify data into semantically meaningful abstractions. Such a capability is useful in systems that aim to provide Intelligent Integration of Information. Here, *mediator services* may include data mining as a means of providing value-added services to learn concepts or to organize information in some reasoned fashion.

There are many useful applications of inductively learned classifiers computed over databases that support other useful query and transaction processing functions. For example, many large business 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. In such applications, a variety of data about a customer are integrated and merged together into a single structured database to which learning programs are applied. Credit bureau data is frequently merged with magazine subscription data as well as a company's own customer data to compose one universal relation for a data mining task. Similarly, defense and intelligence operations utilize similar methodologies on vast information sources to predict a wide range of conditions in various contexts (location of the enemy, conditions for political uprisings, the appearance of bioluminescence in the oceans, and so forth). Many organizations seeking similar added value from their data are already dealing with overwhelming amounts of global information that in time will likely grow in size faster than available improvements in machine resources.

## 1.2 Problem Statement and Our Approach

The central problem we study in this thesis is succinctly stated as:

> *We seek a means to improve the efficiency and accuracy of inductive learning systems applied to very large amounts of data that can be distributed among remote sites.*

*Meta-learning* (Chan & Stolfo, 1993b) is proposed as one such approach. This approach encompasses the use of learning algorithms to learn how to integrate results from multiple learning systems.

The accuracy and efficiency issues can be, and have been, approached separately, but is there a unified approach that can address both of them simultaneously or separately? One advantage of such a unified approach is the generality of applying the same method to each issue. Another is the cohesiveness of the combined solution for both issues. Thus, the central question we pose in this thesis is: "Is there a unified machine learning approach that can achieve high accuracy and efficiency when applied to massive databases of examples?" The research described here is an attempt to provide such a unified approach that we call *meta-learning*, and demonstrate its use for learning concepts accurately and efficiently. Meta-learning is a process that learns how to combine separate and distinct learning systems. Our 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. High accuracy is achieved by intelligently combining separately learned concepts to derive a final learned concept that explains a large data base more accurately than any of the individual learners.

That is, we are trying to build learners that can learn from massive amounts of data efficiently in processing time and in memory usage. Furthermore, data can be inherently partitioned and cannot be brought together at a single location. One example is that different data sets are owned by diverse parties and data sharing is prohibited.

One approach to speed up a learning algorithm is to parallelize the algorithm. However, this approach requires optimizing the code of a particular algorithm for a specific architecture. That is, for each algorithm-architecture combination, the optimized code is probably, if not always, different. With the growing number of learning algorithms and architectures, this requires substantial amount of optimization work to be performed for the desirable algorithm-architecture combination for a learning task.

The efficiency of classifying instances by the resulting learned system is a related and important issue. Our strategies generally produce components that can be executed concurrently, but detail schemes for parallelizing the classification process are beyond the scope of this thesis research.

Since different algorithms have different representations and search heuristics, different search spaces may be explored and hence potentially diverse results can be obtained from different algorithms. Mitchell (1980) refers to this phenomenon as *inductive bias*. That is, the outcome of running an algorithm is biased in a certain direction. Furthermore, different data sets have different characteristics and the performance of different algorithms on these data sets might differ.

Our proposed approach to improve accuracy is to combine different learning systems in a loose fashion by essentially meta-learning a new system that is taught 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.

We note with interest that this general meta-learning approach is independent of the underlying learning algorithms that may be employed. Furthermore, it is independent of the computing platform used. Thus, our meta-learning approach is intended to be *scalable* as well as *portable* and *extensible*.

However, we may not be able to guarantee the accuracy of the final result to be as good as an individual learning algorithm applied to the entire data set since a considerable amount of information may not be accessible to each of the separate learning processes. It is one of the primary issues we study in this thesis.

As our investigation progressed, the space of possible variations of our approach rapidly increased. To limit the scope of this thesis, we focus on the utility of meta-learning as a general and unified approach for scalable and accurate inductive learning in diverse situations. The more important, in our opinion, ideas were explored, but the less important ones were not fully investigated and were left as pointers for further studies.

## 1.3 Brief Summary of Results and Contributions

We proposed *meta-learning* as an unified approach to improving the efficiency and accuracy of inductive learning systems applied to massive amounts of data that can be distributed among remote sites. Here we briefly summarize the results and contributions:

- Several meta-learning strategies have been identified and specific schemes have been developed. A substantial number of systematic empirical evaluations with different permutations of learning algorithms and tasks have been performed.
- The meta-learning strategies do show a consistent improvement in classification accuracy over any of the base classifiers trained on a subsets of available training data. Our studies show that classifiers trained individually from random subsets of a large data set are not as accurate as integrating a collection of separately learned classifiers.
- The meta-learning strategies can outperform the other more common *one-level* voting-based or Bayesian techniques. In the learning tasks and domains we studied, the one-level meta-learning schemes do not consistently maintain high accuracy as the number of subsets increases (and the amount of available data thus decreases). However, the results show that the hierarchical meta-learning approach is able to sustain the same level of accuracy as a global classifier trained on the entire data set distributed among a number of sites.

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

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

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

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

- In many cases, replication buys nothing, meaning that learning over fully distributed disjoint training data with an appropriate distribution of class information is as effective as learning from distributed partially replicated data. This suggests that the various meta-learning strategies indeed do an effective job of sharing knowledge distributed among a set of independently trained classifiers.

- Local meta-learning can improve a classifier at a site by integrating imported remote classifiers. The remote classifiers are treated as "black boxes" and data at remote sites are not shared.

- Deeper analysis of our empirical results show that increase in accuracy improvement can be attributed to greater diversity and fewer correlated errors among more accurate base classifiers.

- The five learning algorithms used in this thesis were analyzed for time complexity. Three out of five algorithms exhibit linear complexity with respect to the number of training examples. However, none of them were linear in practice when very large data sets were used.

- Our parallel implementation demonstrates that our schemes are beneficial to some learning algorithms in terms of speed and others in terms of scalability.

- Meta-learning with multiple learning algorithms and whole data sets achieved as least the accuracy of the most accurate underlying learning algorithm. Since the best learner is not known apriori, meta-learning provides a mechanism to at least match the best.

- Combiner and stacked generalization were comparable in terms of accuracy as well as the resultant concept. However, stacked generalization is computationally more expensive.

Lastly, much of this thesis work has been published at various forums; their citations appear throughout this document.

## 1.4 Organization of the Thesis

In Chapter 2 we overview the machine learning area of inductive learning. Techniques in the literature for improving the accuracy and efficiency of learning algorithms are discussed.

In Chapter 3 we describe our meta-learning approach. The *combiner*, *arbiter*, and *hybrid* strategies are identified and the different specific schemes under these strategies are detailed.

To evaluate our proposed schemes and techniques, we performed a substantial number of experiments across different learning algorithms and tasks. the apparatus and methodology used in our experiments are described in Chapter 4.

In Chapter 5 we present how meta-learning is applied to integrating classifiers that are trained from partitioned data in disjoint subsets and empirically compares meta-learning to techniques found in the literature. Our techniques are also evaluated on data subsets with partially replicated data.

Techniques explored in Chapter 5 can be characterized as one-level techniques. The *combiner tree* and *arbiter tree* strategies are hierarchical techniques and are discussed in Chapter 6 to demonstrate that they can further improve the one-level meta-learning strategies.

In Chapter 7 we investigate how meta-learning can be used to improve a local classifier by integrating it with imported classifiers from remote sites. We assume no "raw data" can be shared among different sites and only the learned classifiers can be exchanged.

In Chapter 8 we formulate several metrics that can be used to analyze the integration of multiple classifiers. These metrics are then used to analyze our empirical results.

In Chapter 9 we perform a formal analysis on the time complexity of the various learning algorithms used in this thesis and the potential speed-up and degree of scalability of using meta-learning techniques. Our parallel and distributed implementation is discussed and evaluated.

Up to this point, this thesis has been discussing results from integrating classifiers trained by a single learning algorithm. In Chapter 10 we explore the integration of classifiers generated by different learning algorithms.

We conclude, in Chapter 11, by discussing the contributions and research directions of this work.

# Chapter 2

# Inductive Learning and Related Work

*Inductive learning* (Michalski, 1983) is the task of identifying regularities in some given set of training examples with little or no knowledge about the domain from which the examples are drawn. Given a set of training data, each interpreted as a set of feature vectors, $x$, and a class label $y$ associated with each vector, the task is to compute a classifier that correctly labels any feature vector drawn from the same source as the training set. It is common to call the body of knowledge that classifies data with the label $y$ as the *concept* $y$. Figure 2.1 depicts the inductive learning process.

For examples, Table 2.1 displays a tiny fraction of the congressional voting record

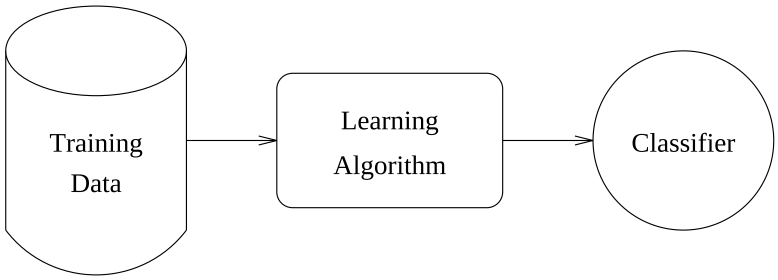


Figure 2.1: Inductive Learning.

| Party | Mx-missile | Edu-spending | Crime |
|---|---|---|---|
| REP | n | y | y |
| REP | n | y | y |
| DEM | n | n | y |
| DEM | n | n | n |

Table 2.1: A data set on congressional voting record.

data set obtained from the Machine Learning Database at the University of California, Irvine (Merz & Murphy, 1996). The data set records how lawmakers vote in the House on different legislative issues. One might want to determine if a lawmaker is a republican or a democrat by observing how he/she votes. From this data set and using party affiliation as a class label, a learning algorithm might produce this concept/classifier (in rule representation):

```
(Edu-spending = y) => REP
(Edu-spending = n) => DEM
```

denoting if a lawmaker votes yes on education spending, he/she is a republican, otherwise, the lawmaker is a democrat. This learned concept can then provide an "educated guess" for the following question (what is Smith's party affiliation?):

```
(Name = smith, Mx-missile = n, Edu-spending = y, Crime = n)
  =>
Party = ?
```

Inductive learning can be *supervised* or *unsupervised*. In supervised inductive learning, the class labels of training examples are supplied and the learned concepts describe these class labels (or *learning from examples* (Dietterich & Michalski, 1983)). However, in unsupervised inductive learning (or *learning from observations* (Michalski & Stepp, 1983)), the class labels are not supplied or known. The learning algorithm induces clusters, which can later be identified as individual concepts. CLUSTER/2 (Michalski & Stepp, 1983), COBWEB (Fisher, 1987), and AUTOCLASS (Chesseman *et al.*, 1988) are such conceptual clustering algorithms. Our work focuses on supervised inductive learning.

Inductive learning can be performed in two modes: *non-incremental* or *incremental*. In non-incremental learning all of the examples are presented to the learning algorithm as an aggregation (for example, ID3 (Quinlan, 1986)) . However, in incremental learning training examples are assimilated one at a time and the learning algorithms do not have control over the order of presentation (for example, ID5 (Utgoff, 1989), an incremental version of ID3). This research concentrates on supervised inductive learning in non-incremental mode.

In inductive learning, examples are usually presented as attribute-value pairs with the corresponding class labels (or *classifications*). Here, a concept (or *classifier*) is loosely defined as a description (pattern) and a conclusion (classification). That is, a concept can be used to draw a conclusion (classifying an instance) based on a matching description. Concepts generated by the learning algorithms can be used in classifying instances that have not been seen before. In other words, given a set of unseen and unclassified instances, the learned concepts merely predict the instances' classification.

Some of the common representations used for the generated classifiers are decision trees, rules, version spaces, neural networks, distance functions, and probability distributions. In general, these representations are associated with different types of algorithms that extract different information from the database and provide alternative capabilities besides the common ability to classify unknown exemplars drawn from some domain. For example, decision trees are declarative and thus more comprehensible to humans than weights computed within a neural network architecture. However, both are capable of classifying data in meaningful ways.

Decision trees are used in ID3 (Quinlan, 1986) and CART (Breiman *et al.*, 1984), where each concept is represented as a conjunction of terms on a path from the root of a tree to a leaf. Rules in AQ (Michalski *et al.*, 1986), Decision Lists (Rivest, 1987), CN2 (Clark & Niblett, 1989), and ITRULE (Goodman & Smyth, 1989) are if-then expressions, where the antecedent is a pattern expression and the consequent is a class label. Each version space learned in the Candidate Elimination algorithm (Mitchell, 1982) defines the most general and specific description boundaries of a concept using a restricted version of first order formulae. Neural networks compute a weighted network to classify data (Fahlman & Hinton, 1987; Lippmann, 1987; Hinton, 1989). The learned distance functions in exemplar-based learning algorithms (or nearest neighbor algorithms) define a similarity or "closeness" measure between two instances (Stanfill & Waltz, 1986; Aha *et al.*, 1991; Cost & Salzberg, 1993). Conditional probability distributions used by Bayesian classifiers are derived from the frequency distributions of attribute values and reflect the likelihood of a certain instance belonging to a particular classification (Duda & Hart, 1973; Langley *et al.*, 1992; Langley & Sage, 1994). Implicit decision rules classify according to maximal probabilities. Chromosomes composed of three-valued pattern vectors constitute a population (classifier) in genetic algorithms (DeJong, 1988; Booker *et al.*, 1989). Learning involves evolving the chromosomes to maximize a fitness function.

These algorithms and their variants have been put to practical use in a wide range of data mining activities. In this thesis we do not seek new learning algorithms to add to this broad list. Rather, we propose meta-learning as an approach whereby any of these algorithm can be used in a "plug-and-play fashion." We utilize inductive learning algorithms to not only explain databases of information drawn from some arbitrary domain, but also we apply these same algorithms to a distributed database of predictions generated by a set of underlying classifiers! This means, we apply inductive learning to the task of "learning how distributed classifiers correlate with each other" to improve the accuracy of the desired classification process. This is the essence of what we mean by meta-learning. Our ultimate goal is to provide scalable inductive learning and classification capabilities in wide area computing networks to be able to learn globally what is only partially learned locally.

## 2.1 Improving Accuracy

Machine learning researchers clearly desire more accurate learning algorithms. One direction is to generate diverse classifiers by some method using a single learning algorithm and the classifiers are combined via some mechanism. Another approach has focussed on integrating by some means multiple strategies or multiple algorithms. Here we summarize a few of the most relevant efforts.

### 2.1.1 Single learning algorithm and diverse classifiers

Some research has concentrated on methods to improve an existing algorithm by using the algorithm itself to generate purposely biased distributions of training data. The most notable work in this area is due to Schapire (1990), which he refers to as *hypothesis boosting*. Based on an initial learned hypothesis for some concept derived from a random distribution of training data, Schapire's scheme iteratively generates two additional distributions of examples. The first newly derived distribution includes randomly chosen training examples that are equally likely to be correctly or incorrectly classified by the first learned classifier. A new classifier is formed from this distribution. The second distribution is formed from the training examples on which both of the first two classifiers disagree. A third classifier is computed from this distribution. The predictions of the three learned classifiers are combined using a simple voting rule. Schapire proves that the overall accuracy is higher than the one achieved by simply applying the learning algorithm to the initial distribution under the PAC (*Probabilistic Approximately Correct*) learning model (Valiant, 1984). In fact, he shows that arbitrarily high accuracy can be achieved by recursively applying the same procedure. Although his approach is limited to the PAC model of learning, some success was achieved in the domain of character recognition, using neural networks (Drucker *et al.*, 1993). Freund (1992) has a similar approach, but with potentially many more *sequentially* generated distributions involved.

Hansen and Salamon (1990) integrate an ensemble of neural networks by simple voting. The different networks in an ensemble are generated by randomized parameters. Kwok and Carter (1990) generate different decision trees by choosing different tests at the root and combine their predictions by simple voting. Breiman's (1994) *bagging* utilizes bootstrapping to generate many different training distributions and voting to combine the predictions. Optiz and Shavlik (1996) perturb neural networks using genetic algorithms and combine the networks using weighted voting.

Dietterich and Bakiri (1991; 1995) augment the output representation of a multi-class problem using error-correcting codes. Generated classifiers are integrated by choosing the code with the fewest errors in the code book.

Other work in this direction includes Qian and Sejnowski's (1988) cascaded neural networks, where the output of one neural network is fed into another to learn higher-level correlations. Kohavi and John (1995) search for the best algorithm parameters using extensive cross validation runs. Naik and Mammone (1992) apply learning to selecting parameters for neural networks. Pomerleau (1992) uses a rule-based approach to combining multiple driving experts for guiding a vehicle in a variety of circumstances. The driving experts are trained neural networks.

### 2.1.2 Integrating multiple learning algorithms

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

There are mainly two strategies that we may consider in integrating different learning strategies. One strategy is to increase the amount of knowledge in the learning system. For example, some work has been reported on integrating inductive and explanation-based learning (Flann & Dietterich, 1989; Danyluk, 1991). Explanation-based techniques are integrated to provide the appropriate domain knowledge that complements inductive learning, which is knowledge poor. This approach requires a complicated new algorithm that implements both strategies to learning in a single system. A less knowledge-intensive direction uses heuristics to combine multiple classifiers in a tree structure (Tcheng *et al.*, 1989; Brodley, 1995). The space of training examples is recursively partitioned into subspaces, from each of which a classifier is generated using a heuristically selected learning algorithm. (Further details on these methods are provided in Section 6.3).

Another strategy is to loosely integrate a number of different inductive learning algorithms by integrating their collective output concepts in some fashion. Some of these techniques are described below and later evaluated from our empirical results. For example, Silver et al.'s (1990) work on using a coordinator to gather votes from three different classifiers and Holder's (1991) work on selecting learning strategies based on their relative utility.

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

Littlestone and Warmuth (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 $\beta$, 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 worst classifier makes at most $m$ mistakes, the *weighted majority* algorithms will make at most $O(log(number\_of\_classifiers) + m)$ mistakes. 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 integrating predictions from multiple classifiers based on the Bayesian formalism. The belief function they derived (Equation

32) is simplified as:

$$bel(class_i, x) \approx \prod_{k}^{classifiers} P(class_i \mid classifier_k(x)),$$

where $x$ is an instance and $classifier_k(x)$ is the classification of instance $x$ predicted by $classifier_k$. The final prediction is $class_j$ where $bel(class_j, x)$ is the largest among all classes. In our experiments reported below, we estimate the conditional probabilities from the frequencies generated from the validation set. Xu et al. (1992) also developed integrating methods based on the Dempster-Shafer theory.

A more interesting approach to loosely combine learning programs is to learn how to combine independently learned concepts. Stolfo et al. (1989) propose learning rules by training weighted voting schemes, for merging different phoneme output representations from multiple trained speech recognizers. Wolpert (1992) presents a theory of *stacked generalization* to combine several classifiers. (Indeed, this work is closest to what we mean by meta-learning as we will describe later.) Several (level 0) classifiers are first learned from the same training set. The predictions made by these classifiers on the training set and the correct classifications form the training set of the next level (level 1) classifier. When an instance is being classified, the level 0 classifiers first make their predictions on the instance. The predictions are then presented to the level 1 classifier, which makes the final prediction. Zhang et al.'s (1992) work utilizes a similar approach to learn a *combiner* based on the predictions made by three different classifiers. Breiman (1996b) applied the stacking idea to regression. The techniques are closest to our meta-learning approach proposed here.

## 2.2 Improving Efficiency

Quinlan (1979) approached the problem of efficiently applying learning systems to data that are substantially larger than available main memory with a windowing technique. A learning algorithm is applied to a small subset of training data, called a *window*, and the learned concept is tested on the remaining training data. This is repeated on a new window of the same size with some of the incorrectly classified data replacing some of the data in the old window until all the data are correctly classified. Wirth and Catlett (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. Catlett (1991) demonstrates that larger amounts of data improves accuracy, but he 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. Using data reduction techniques, Domingos (1996) significantly improves the efficiency of RISE (a specific-to-general rule induction algorithm) (Domingos, 1995).

Catlett (1991) proposes some improvements to the ID3 algorithm particularly for handling attributes with real numbers. For each real-numbered attribute, ID3 sorts the attribute values present in the examples, considers a two-way split of the values (a threshold) between each pair of adjacent values, and selects the most effective split according to an objective function. That is, $n-1$ splits are considered for $n$ values. Catlett devised a scheme to skip some of the splits that are considered statistically not likely to be picked (Catlett, 1992). This scheme applies only to real-numbered attributes and the processing time can still be prohibitive due to ID3's non-linear complexity (Chan & Stolfo, 1993d).

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.

Other researchers use a more coarse-grain parallel/distributed approach. Classifiers are trained from data subsets and are combined via some mechanism. Provost et al. (Provost & Aronis, 1996; Provost & Hennessy, 1996) exchange and evaluate rules to provably and optimally combine rule sets learned by RL (Clearwater & Provost, 1990). Our meta-learning approach is similar, however, it is not restricted to a particular learning algorithm.

## 2.3 Incremental Learning

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.

## 2.4 Our Approach

Again, our approach for improving efficiency and accuracy for learning algorithms focuses on data reduction and meta-learning techniques. The meta-learning techniques attempt to learn correlations among the classifiers trained on multiple data sets. That is, they try to learn how to effectively integrate learned classifiers to achieve an accuracy higher than any of the individual classifiers. Data reduction techniques reduce and limit the amount of data inspected by any individual learning process. Unlike many related techniques, our meta-learning approach is scalable (by data reduction partitioning), extensible (algorithm-independent), and portable (architecture-independent). In the next chapter our meta-learning approach is discussed in detail.

## 2.5 Community

Because of the growing interest and importance in the area of integrating diverse learning systems, Prof. Sal Stolfo, Dr. Dave Wolpert, and I organized a workshop at the Fourteen National Conference on Artificial Intelligence (AAAI-96) in Portland, Oregon. The workshop was entitled "Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms" (Chan *et al.*, 1996) and was held on August 4th and 5th. 33 paper submissions were received from around the world. Because of the unexpected relatively large number of submissions, reviewers other than the organizers were enlisted. After evaluating two reviews for each submission, we accepted 24 papers for presentation. About 80 researchers and developers expressed interest in attending the two-day workshop; around 50 of them came and participated.

# Chapter 3

# Meta-Learning

*Meta-learning* (Chan & Stolfo, 1993b) is loosely defined as learning of meta-knowledge about learned knowledge. In our work we concentrate on learning from the output of concept learning systems. In this case meta-learning means learning from the *predictions* of these classifiers on common *training data*. A classifier (or concept) is the output of a concept learning system and a prediction (or classification) is the predicted class generated by a classifier when an instance is supplied. Thus, we are interested in the output of the classifiers, not the internal structure and strategies of the learning algorithms themselves. Moreover, in several of the schemes we define, the training data presented to the learning algorithms initially are also available to the *meta-learner* under certain circumstances.

Figure 3.1 depicts the different stages in a *simplified* meta-learning scenario:

1. The classifiers (*base classifiers*) are trained from the initial (base-level) training sets.

2. Predictions are generated by the learned classifiers on the training sets.

3. A meta-level training set is composed from the predictions generated by the classifiers.

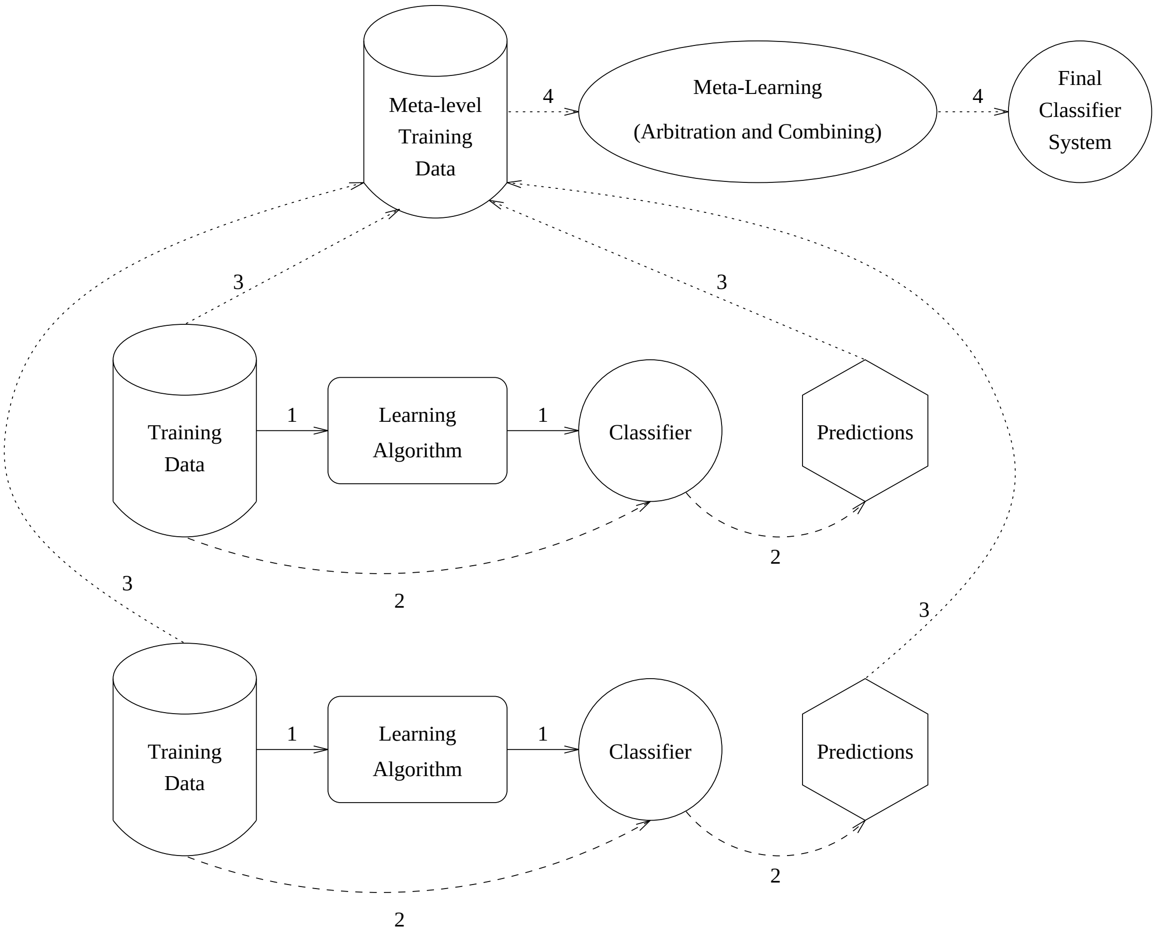


Figure 3.1: Meta-learning.

4. The final classifier *meta-classifier* is trained from the meta-level training set.

In meta-learning a learning algorithm is used to learn how to integrate the learned classifiers. That is, rather than having a predetermined and fixed integration rule (for example, voting), the integration rule is learned based on the behavior of the trained classifiers.

Sections 3.1 and 3.2 discuss how the base classifiers can be generated and how they can be integrated. Section 3.4 details our meta-learning strategies. Section 3.3 summarizes our methods by contrasting them with others in the literature.

## 3.1 Computing Initial Base Classifiers

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

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

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

Random selection of the partitioned data sets with a uniform distribution of classes is perhaps the most sensible solution. Here we may attempt to maintain the same frequency distribution over the "class attribute" so that each partition represents a good but smaller model of the entire training set. Otherwise, a totally random selection strategy may result in the absence of some classes we wish to discriminate among in some of the training subsets. Several experiments have been conducted and are reported below to explore these issues. Unfortunately, there is no strong theory to guide us on how to optimally solve this problem.

An alternative to partitioning data into disjoint training subsets is to apply "partial data reduction," meaning that we may allow for some amount of replication in the partitioned data sets. In this way, the separately learned base classifiers have some hope of analyzing common data, some of which may include "near-misses." However, this strategy implies that we are not making maximal use of our parallel resources, since a considerable amount of the original training database is being replicated at various distributed computing sites. However, the extreme bias in the data that may be derived from purely disjoint data partitioning can be relaxed to some degree with partial replication.

Several experiments have been conducted and are reported in later chapters detailing the surprising outcome of these two strategies. Disjoint partitioning of training data versus partially replicating information among the base training data sets is compared over two learning tasks, varying the amount of replication in a series of tests. The results show that partial replication essentially buys nothing: no improvement, nor any reduction, in accuracy is seen! Thus, meta-learning over disjoint sets of training data is effective provided the distribution of training data is not highly skewed or severely biased.

## 3.2 Integrating Base Classifiers

Since different learning algorithms employ different knowledge representations and search heuristics, different search spaces may be explored by each and hence potentially diverse results can be obtained. Mitchell (1980) refers to this phenomenon as *inductive bias*; the outcome of running an algorithm is biased towards a certain outcome. Furthermore, different partitions of a data set have different statistical characteristics and the performance of any single learning algorithm might differ substantially over these partitions. These observations imply that great care must be taken in designing an appropriate distributed meta-learning architecture. A number of these issues are explored in this section.

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

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

$P(C_i)$ is the probability that $C_i$ predicts correctly, (i.e., the probability it is the true model), while $P(y \mid C_i, x)$ is the probability that $x$ is of class $y$ given by $C_i$. Of course, this makes sense only when the probabilities are indeed known, and our classifiers are probabilistic and not categorical. The best we can do to estimate $P(C_i)$ is to calculate the appropriate statistics from observing the behavior of each classifier on the training set as an approximation to the actual probabilities (which may be quite inaccurate.) (Furthermore, Bayes theorem would be optimal if we knew all possible classifiers, not just those that we happen to compute.) This information, however, provides only statistics about each classifier's behavior with respect to the training set, and no information about how the classifiers are related to each other. For example, learning that two classifiers rarely agree with each other when predicting a class label $y$ (meaning that when one classifier predicts $y$, the other does not) might have much more predictive value (eg. when combined with a third classifier) than merely knowing that the two classifiers predict $y$ with equal probability! We view the Bayesian approach as a baseline, and use methods derived from this approach, Bayesian Belief (Xu *et al.*, 1992), for comparative purposes in our experiments reported later. There are many other approaches we might imagine that are based upon learning relationships between classifiers. The manner in which we learn the relationship between classifiers is to *learn a new classifier* (a "meta-level classifier") *whose input is the set of predictions of two or more classifiers on common data*. It is this latter view that we call meta-learning.

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

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

- Can meta-learning over data partitions maintain or boost the accuracy of a single global classifier?
- How do voting and Bayesian techniques compare to meta-learning in accuracy?
- How do arbiters compare to combiners in accuracy?
- A meta-learned classifier may be treated as a base classifier. Thus, might hierarchically meta-learned classifiers perform better than a single layered meta-learned architecture?

- How much training data and of what distribution should an arbiter or combiner be provided in order to produce accurate results?
- Might meta-learned classifiers be improved by learning over partitions of partially replicated training data, or is disjoint training data sufficient to achieve high accuracy?

A substantial number of exploratory evaluations have been completed. Details of these results are in the following chapters.

We have discovered through experimentation three very interesting behaviors exhibited by various meta-learning strategies that warrant further elaboration. We demonstrate that under certain circumstances, a meta-learning architecture can learn effectively with a fraction of the total available information at any one site, that accuracy can be boosted over the global classifier trained from all available data, and that maximal parallelism can be effectively exploited by meta-learning over disjoint data partitions without a substantial loss of accuracy (Chan & Stolfo, 1996a). These results suggest strongly that a "field test" of these techniques over a real world network computing environment (eg. over database server sites on the web) is not only technically feasible, but also an important next step in the development of these ideas.

In the following sections we present some of the different strategies used in our meta-learning study.

## 3.3 Voting, Combining and Arbitration

We distinguish three distinct strategies for combining multiple predictions from separately learned classifiers. *Voting* generally is understood to mean that each classifier gets one *vote*, and the majority (or plurality) wins. *Weighted voting* provides preferential treatment to some voting classifiers, as may be predicted by observing performance on some common test set. The outcome of voting is simply to choose one of the predictions from one or more of the classifiers. The second major strategy is *arbitration*, which entails the use of an "objective" judge whose own prediction is selected if the participating classifiers cannot reach a consensus decision. Thus, the arbiter is itself a classifier, and may choose a final outcome based upon its own prediction but cognizant of the other classifiers' predictions. Finally, *combining* refers to the use of knowledge about how classifiers behave with respect to each other. Thus, if we learn, for example, that when two classifiers predict the same class they are always correct (relative to some test set), this simple fact may lead to a powerful predictive tool. Indeed, we may wish to ignore all other classifiers when they predict a common outcome. Figures 3.2 and 3.6 contrast the primary differences between combiners and arbiters, which are now detailed.

## 3.4 Meta-learning by Combining and Arbitration

We distinguish between *base classifiers* and *combiners/arbiters* as follows. A base classifier is the outcome of applying a learning algorithm directly to "raw" training data. The base classifier is a program that given a test datum provides a prediction of its unknown class. An *combiner* or *arbiter*, 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 combiners/arbiters in a hierarchical manner.

Before we detail the different strategies, for concreteness, we define the following notations. 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, $C_1$, $C_2$, ... $C_k$. Examples randomly drawn from the entire original training set constitute the *validation set*, $E$, which is used to generate the meta-level training set according to the following strategies. $class(x)$ and $attribute\_vector(x)$ denote the correct classification and attribute vector of example $x$ as specified in the validation set, $E$.

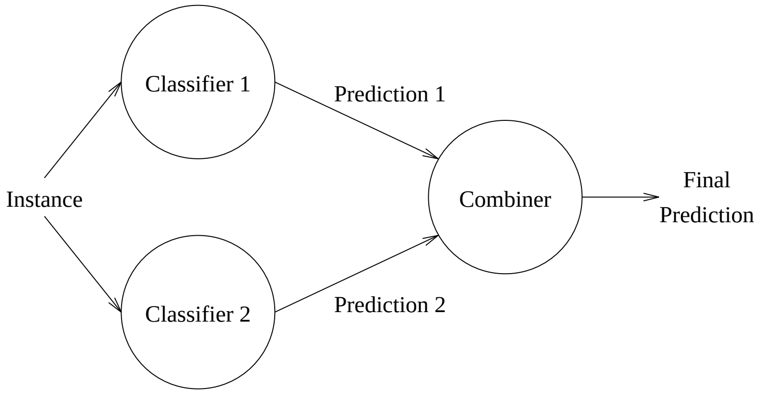


Figure 3.2: A combiner with two classifiers.

### 3.4.1 Combiner strategy

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 3.2). The aim of this strategy is to "coalesce" the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction. A combiner computes a prediction that may be entirely different from any proposed by a base classifier, whereas an arbiter chooses one of the predictions from the base classifiers and the arbiter itself.

We experimented with three 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:

| Example | Class | Attribute vector | Base classifiers' predictions | |
|---|---|---|---|---|
| $x$ | $class(x)$ | $attrvec(x)$ | $C_1(x)$ | $C_2(x)$ |
| $x_1$ | `table` | $attrvec_1$ | `table` | `table` |
| $x_2$ | `chair` | $attrvec_2$ | `table` | `chair` |
| $x_3$ | `table` | $attrvec_3$ | `chair` | `chair` |

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

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

Figure 3.3: Sample training sets generated by the *class-combiner* and *class-attribute-combiner* schemes with two base classifiers.

**class-combiner** 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). See Figure 3.3 for a sample training set.

**class-attribute-combiner** Return meta-level training instances as in *class-combiner* with the addition of the attribute vectors; i.e., $T = \{(class(x), C_1(x), C_2(x), ....C_k(x), attribute\_vector(x)) \mid x \in E\}$. See Figure 3.3 for a sample training set.

**binary-class-combiner** Return meta-level training instances similar to those in the *class-combiner* 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,

| Example | Class | Attribute vector | Base classifier1's predictions | | Base classifier2's predictions | |
|---|---|---|---|---|---|---|
| $x$ | $class(x)$ | $attrvec(x)$ | $C_{1_{table}}(x)$ | $C_{1_{chair}}(x)$ | $C_{2_{table}}(x)$ | $C_{2_{chair}}(x)$ |
| $x_1$ | `table` | $attrvec_1$ | yes | no | yes | no |
| $x_2$ | `chair` | $attrvec_2$ | yes | yes | no | yes |
| $x_3$ | `table` | $attrvec_3$ | no | yes | no | yes |

| Training set from the *binary-class-combiner* scheme | | |
|---|---|---|
| Instance | Class | Attribute vector |
| 1 | `table` | (yes, no, yes, no) |
| 2 | `chair` | (yes, yes, no, yes) |
| 3 | `table` | (no, yes, no, yes) |

Figure 3.4: Sample training set generated by the *binary-class-combiner* scheme with two base classifiers.

$C_{i_j}(x)$, 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, $T = \{(class(x), C_{1_1}(x), \ldots, C_{1_m}(x), C_{2_1}(x), \ldots, C_{2_m}(x), \ldots C_{k_1}(x), \ldots, C_{3_m}(x)) \mid x \in E\}$. See Figure 3.4 for a sample training set.

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 an 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 datum.

Figure 3.5 shows a combiner (in decision tree format) that is learned from 4 base classifiers in the RNA splice junction domain (Section 4.2.1). The combiner strategy discovers that `base-classifier-1` is much more relevant than the others that only

```
base-classifier-1 = EI:  EI
base-classifier-1 = IE:
|   p-3 = A:  N
|   p-3 = C:  IE
|   p-3 = G:  N
|   p-3 = T:  IE
base-classifier-1 = N:
|   p1 = A:  N
|   p1 = C:  N
|   p1 = G:
|   |   p5 = A:  N
|   |   p5 = C:  N
|   |   p5 = G:
|   |   |   p2 = A:  N
|   |   |   p2 = C:  N
|   |   |   p2 = G:  N
|   |   |   p2 = T:  EI
|   |   p5 = T:  N
|   p1 = T:  N
```

Figure 3.5: A sample combiner learned from 4 base classifiers. One classifier c1 survived.

`base-classifier-1` appears in the combiner.

### 3.4.2 Arbiter strategy

In the *arbiter* strategy, the training set for the meta-learner is a subset of the training set for the base learners; i.e. the meta-level training instances are a particular distribution of the raw training set. The predictions of the learned base classifiers for the training set and a *selection rule*, which varies in different schemes, determines

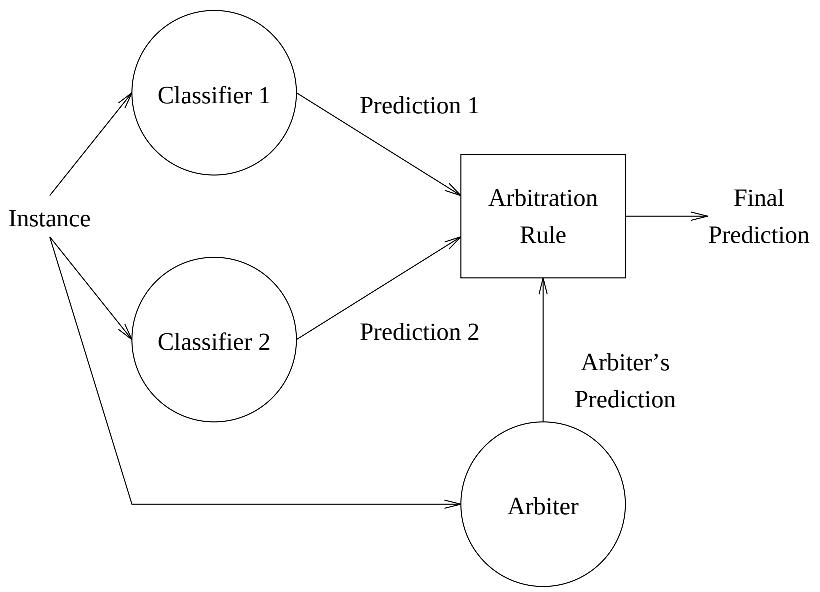


Figure 3.6: An arbiter with two classifiers.

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 training instances for a combiner 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's prediction and a corresponding *arbitration rule*, generate the final prediction (see Figure 3.6). 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 three schemes for the *selection rule*, which chooses training examples for an arbiter. In essence the schemes select examples that are confusing to the base classifiers, from which 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 selected from $E$ is dictated by a *selection rule*, that purposefully biases the arbiter training data. The three versions of this selection rule implemented and reported here include:

**different-arbiter** Select an instance from $E$ if none of the classes in the $k$ base predictions gathers a majority classification ($> k/2$ votes); i.e., $T = T_d = \{x \in E \mid no\_majority(C_1(x), C_2(x), ...C_k(x))\}$. The purpose of this rule is to choose data that are in some sense "confusing"; i.e., the majority of classifiers do not agree on how the data should be classified (*different* opinions). As we will show later, this rule has comparable performance to the other more complex rules, hence, when the specification of a selection rule is absent, this rule is implied. That is, this is the **default arbiter strategy**. For further reference, this scheme is denoted as *arbiter* or *different-arbiter*.

**different-incorrect-arbiter** Select instances with predictions that does not gather a majority, $T_d$, as in the first case, but also instances with predictions that have a majority but are incorrect; i.e, $T = D \cup I$, where $I = T_i = \{x \in E \mid majority(C_1(x), C_2(x), ..., C_k(x)) \neq class(x)\}$. Note that we lump together both cases of data that are incorrectly classified or in disagreement (no majority).

**different-incorrect-correct-arbiter** Return a set of three training sets: $T_d$ and $T_i$, as defined above, and $T_c$ with examples that have the same correct predictions; i.e., $T = \{T_d, T_i, T_c\}$, where $T_c = \{x \in E \mid majority(C_1(x), C_2(x), ..., C_k(x)) = class(x)\}$. Here we attempt to separate the data into three cases and distinguish each case by learning a separate "subarbiter." $T_d$, $T_i$, and $T_c$ generate $A_d$, $A_i$, and $A_c$, respectively. The first arbiter is like the one computed in the first case to arbitrate disagreements. The second and third arbiters attempt to distinguish the cases when the predictions have a majority but are either incorrect or correct.

Figure 3.7 depicts sample training sets for the three arbiter schemes. Note the difference in training data for the arbitration and combining. Arbiters are computed from a distinguished and biased subset of data selected from the input database

| Example | Class | Attribute vector | Base classifiers' predictions | |
|---|---|---|---|---|
| $x$ | $class(x)$ | $attrvec(x)$ | $C_1(x)$ | $C_2(x)$ |
| $x_1$ | `table` | $attrvec_1$ | `table` | `table` |
| $x_2$ | `chair` | $attrvec_2$ | `table` | `chair` |
| $x_3$ | `table` | $attrvec_3$ | `chair` | `chair` |

| Training set from the *different-arbiter* scheme | | | |
|---|---|---|---|
| Instance | Example | Class | Attribute vector |
| 1 | $x_2$ | `chair` | $attrvec_2$ |

| Training set from the *different-incorrect-arbiter* scheme | | | |
|---|---|---|---|
| Instance | Example | Class | Attribute vector |
| 1 | $x_2$ | `chair` | $attrvec_2$ |
| 2 | $x_3$ | `table` | $attrvec_3$ |

| Training sets from the *different-incorrect-correct-arbiter* scheme | | | | |
|---|---|---|---|---|
| Set | Instance | Example | Class | Attribute vector |
| Different ($T_d$) | 1 | $x_2$ | `chair` | $attrvec_2$ |
| Incorrect ($T_i$) | 1 | $x_3$ | `table` | $attrvec_3$ |
| Correct ($T_c$) | 1 | $x_1$ | `table` | $attrvec_1$ |

Figure 3.7: Sample training sets generated by the three arbiter schemes with two base classifiers.

used to train the base classifiers. Combiners, however, are trained on the predicted classifications of that data generated by the base classifiers, as well as the data itself.

The learned 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, $y$, an *arbitration rule* and the learned arbiter, $A$, produce a final prediction based on the $k$ predictions, $C_1(y)$, $C_2(y)$ ... $C_k(y)$, from the $k$ base classifiers and the arbiter's own prediction, $A(y)$.

Two versions of the *arbitration rule* have been implemented. The first version corresponds to the first two *selection* strategies, while the second version corresponds to the third strategy.

**different-arbiter or different-incorrect-arbiter** Return the class with a plurality of occurrences in $C_1(y)$, $C_2(y)$, ... $C_k(y)$, and $A(y)$, with preference given to the arbiter's choice in case of a tie. For example, if the three classifiers predict `table`, `chair`, and `table` and the arbiter predicts `chair` (i.e., a tie), the final prediction is `chair`.

**different-incorrect-correct-arbiter** Return a class label according to:
if $no\_majority(C_1(y), C_2(y), ...C_k(y))$

return $A_d(y)$

else if $majority(C_1(y), C_2(y), ...C_k(y)) = A_c(y)$

return $A_c(y)$

else

return $A_i(y)$,

where $A = \{A_d, A_i, A_c\}$.

This rule tries to differentiate the three different circumstances so that the three specialized "subarbiters" can be utilized.

We described the combiner and arbiter strategies for meta-learning. It is important to note the difference between the combiner and arbiter strategies. The combiner strategy tries to find relationships among the predictions generated by the classifiers and the correct predictions. A combiner is a "learned function" that determines the final prediction given a set of predictions. For example, given an unlabeled instance $x$, the combiner may learn a rule stating that if classifier $C_1$ predicts `table`, and $C_2$ predicts `chair`, then the combiner predicts `lamp` (i.e., possibly a completely different prediction from either classifier). However, the arbiter strategy attempts to arbitrate among the conflicting predictions and an arbiter is just another classifier, but trained on a biased distribution of the original examples. Here, for example, when

$C_1$, and $C_2$'s predictions disagree, the arbiter makes its own prediction, which could be completely different from the two base predictions, and a vote determines the final prediction.

We next discuss hybrid schemes that merge some of ideas from the arbiter and combiner strategies.

### 3.4.3 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 data in disagreement by the base classifiers. A learning algorithm then generates a meta-classifier, effectively a *combiner*, 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. The hybrid strategy thus attempts to improve the arbiter strategy by coalescing predictions instead of purely arbitrating among them.

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

**different-class-attribute-hybrid** Select examples that have different predictions from the base classifiers and the predictions, together with the correct classes and

| Example | Class | Attribute vector | Base classifiers' predictions | |
|---|---|---|---|---|
| $x$ | $class(x)$ | $attrvec(x)$ | $C_1(x)$ | $C_2(x)$ |
| $x_1$ | `table` | $attrvec_1$ | `table` | `table` |
| $x_2$ | `chair` | $attrvec_2$ | `table` | `chair` |
| $x_3$ | `table` | $attrvec_3$ | `chair` | `chair` |

| Training set from the *different-class-attribute-hybrid* scheme | | |
|---|---|---|
| Instance | Class | Attribute vector |
| 1 | `chair` | (`table`, `chair`, $attrvec_2$) |

| Training set from the *different-incorrect-class-attribute-hybrid* scheme | | |
|---|---|---|
| Instance | Class | Attribute vector |
| 1 | `chair` | (`table`, `chair`, $attrvec_2$) |
| 2 | `table` | (`chair`, `chair`, $attrvec_3$) |

Figure 3.8: Sample training sets generated by the *hybrid* schemes.

attribute vectors form the training set for the meta-learner. This integrates the *different-arbiter* and *class-attribute-combiner* schemes.

**different-incorrect-class-attribute-hybrid** 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 *different-incorrect-arbiter* and *class-attribute-combiner* schemes.

Sample training sets for these two hybrid schemes are displayed in Figure 3.8.

Much of our investigation in this thesis focuses on the *class-combiner*, *class-attribute-combiner*, and *different-arbiter* schemes. The other more complex schemes were not examined as much because preliminary experiments indicate that they do not gain much upon the simpler schemes. The next chapter details our experimental apparatus and methodology.

# Chapter 4

# Experimental Apparatus and Methodology

To evaluate our meta-learning approach and other techniques in the literature, we performed a substantial numbers of experiments using a variety of learning algorithms and tasks. We first discuss the apparatus and then the methodology for our experiments.

## 4.1 Learning Algorithms

Five inductive learning algorithms were used in our experiments. Implementations of these algorithms are "off-the-shelf" and were not modified. The variety of algorithms provide some generality for our empirical results.

We obtained ID3 (Quinlan, 1986) and CART (Breiman *et al.*, 1984) as part of the IND package (Buntine & Caruana, 1991) from NASA Ames Research Center; both algorithms compute decision trees. CN2 (Clark & Niblett, 1989) is a rule learning algorithm and was obtained from Dr. Clark (Boswell, 1990). WPEBLS is the weighted version of PEBLS (Cost & Salzberg, 1993), which is a nearest-neighbor learning algorithm. BAYES is a naive Bayesian learning algorithm that is based on computing conditional probabilities as described in (Clark & Niblett, 1989). The last two algorithms were reimplemented in C++.

## 4.2 Learning Tasks

Various machine learning techniques have been applied to different molecular biology sequence analysis tasks (Chan, 1991; Craven & Shavlik, 1994). For our study, we chose three sequence analysis tasks obtained from the Machine Learning Database Repository at University of California, Irvine (Merz & Murphy, 1996). Moreover, we also used an artificial data set that can be generated at random.

### 4.2.1 Molecular biology sequence analysis data

Molecular biologists in genetics have been focusing on analyzing sequences obtained from proteins, DNA (DeoxyriboNucleic Acid), and RNA (RiboNucleic Acid). These sequences are divided into two groups: amino acid sequences and nucleotide sequences. These two groups are briefly discussed in the following two sections. Further readings can be found in (Schleif, 1986; Lewin, 1987; Hunter, 1993).

The basic building blocks of genetics are nucleotides. DNA consists of two chemically linked sequences of nucleotides (or polynucleotide chains) in double helix form whereas RNA consists of only one polynucleotide chain. In DNA the nucleotide sequence of one strand complements the other strand in such a way that one strand can be used to synthesize the other. There are four different kinds of nucleotides in DNA (adenine, cytosine, guanine, and thymine) and RNA (adenine, cytosine, guanine, and uracil). That is, a DNA or RNA segment can be represented as a sequence of symbols, where each symbol denotes one of the four nucleotides. For example, GGGACGGUCC is a segment of ten nucleotides of the U1 RNA (Nakata *et al.*, 1985).

Although a DNA molecule consists of two nucleotide strands, one strand is the complement of the other, which is more or less redundant in terms of encoding genetic information. If one constructs the sequence of all the DNA molecules in an organism's genome, one can represent the organism as a long sequence of just four letters. The length of the human DNA sequence is estimated to be $3 \times 10^9$.

Characteristics and functions of organisms are controlled by proteins, whose production is regulated by the information encoded in the nucleotide sequences of DNA. A *gene* is basically a DNA fragment that carries the information representing a particular protein. This genetic information is primarily the order of nucleotides in the DNA. Proteins are not directly produced from the information on DNA. Instead, information in a gene is copied to another type of nucleotide sequence, called RNA, whose nucleotide order is used to produce proteins. Specifically, a sequence of three nucleotides, a *codon*, encodes one amino acid. Of the 64 possible codons, 61 represent amino acids and the remaining three signify the end of the encoded proteins. Since there are only 20 amino acids, all except two amino acids are represented by multiple codons.

As in analyzing amino acid sequences, we can gain more structural and functional information about DNA and RNA from studying their nucleotide sequences. For example, the decoding process of producing a protein always starts at a certain location in a DNA segment called the *promoter*. Molecular biologists try to identify the initiation region in a given DNA sequence so that they can understand more about the interactions between DNA and protein production.

Proteins are fundamental and instrumental in every aspect of biological function even though they are relatively simple in their sequence structure. Proteins consist of *polypeptide chains*. Each polypeptide chain is an unbranched sequence of amino acids linked by *peptide bonds*. There are a total of twenty different primary amino acids; others are derived from the primary ones. A protein segment can thus be represented as a sequence of symbols, where each symbol signifies a distinct amino acid. For example, PIVDTGSVAP is a segment of ten amino acids in Hemoglobin V (Qian & Sejnowski, 1988).

Due to the physical and chemical interactions among amino acids, proteins do not appear as linear ropes. Interacting segments create twists and turns (called *protein folding*). Scientists have identified structural patterns in proteins and classified four structural levels. The *primary structure* is the sequence of amino acids, a linear chain of specific acids. The main *conformational states* in the *secondary structure* are *$\alpha$-helix*, *$\beta$-sheet*, and *coil* (not *helix* or *sheet*), which are three-dimensional shapes formed from this linear chain. Features in the secondary structure induce the *tertiary structure*. For proteins that consist of multiple polypeptide chains, *multimeric proteins*, interactions among chains generate the *quaternary structure*.

The shape of a protein largely determines its functions. Various structural features provide sites for biochemical activities[1]. A lot of the enzymatic activities hinge on lock-and-key type reactions, which highly rely on structural properties.

The purpose of analyzing amino acid sequences is to gain information about proteins both structurally and functionally. Consider, for example, the *secondary structure* is important in determining the function of the protein. Scientists have been trying to find ways to predict the secondary structures from amino acid sequences so they can learn more about the functional properties of proteins (more in Sec. 4.2.1).

**RNA splice junction**

Protein synthesis begins with the construction of an mRNA molecule (messenger RNA (ribonucleic acid)) based on the nucleotide sequence of a DNA molecule. This process is called *transcription*. The composition of RNA is similar to that of DNA,

[1]Imagine a rope with a series of knots, each knot representing an amino acid. Our imaginary protein rope can be twisted and folded into a globule, the most frequent shape in protein, exposing some knots externally while hiding others internally. The exposed external knots generate various shapes, which largely determine the protein's biochemical behavior.

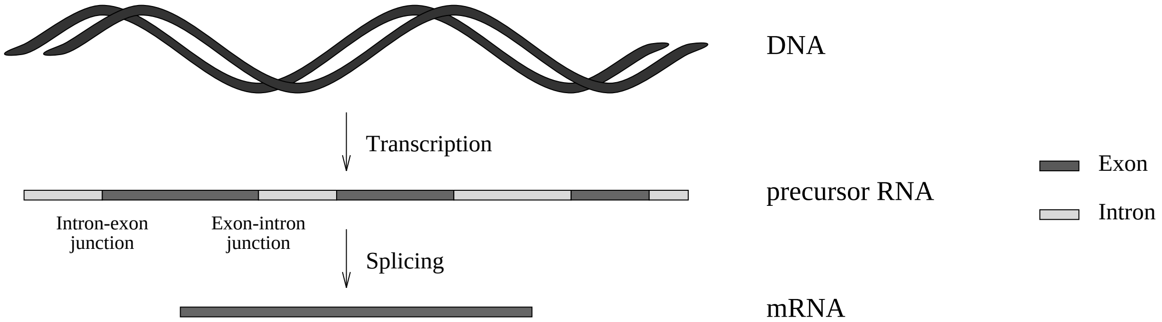


Figure 4.1: Splice junctions and mRNA.

except RNA is single-stranded, the ribose component replaces the deoxyribose one, and uracil (U) replaces thymine. The second process is *translation*, where each coding triplet of nucleotides on an mRNA molecule is mapped to an amino acid and a chain of amino acids forms a protein.

In eukaryotes' (organisms with cells that have nuclear membrane (for example, human)) DNA, there are *interrupted* genes. That is, some regions of a gene do not encode protein information. During *transcription*, these non-protein-encoding regions (called *introns*) are passed to the precursor RNA. Introns are sliced off before translation begins. The regions that encode protein information (called *exons*), are spliced together and the resultant intron-free mRNA is used in translation. Figure 4.1 schematically depicts the process of generating an mRNA molecule.

The RNA 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 learning task is to identify the type of splice junction with high accuracy.

### Protein coding region

As discussed previously, amino acids are encoded by nucleotide triplets (codons) in a DNA sequence. It is important to know how the nucleotides are grouped into codons (*reading frames*). Starting at position $x$ and position $x+1$ gives two different sequences of codons and hence two different sequences of amino acids. One amino acid sequence might constitute a known protein, the other might be a structure of unknown nature.

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. If the 15 nucleotides represent 5 codons which are part of a known protein, the sequence is labeled coding, otherwise, it is labeled non-coding. The PCR data set has 20,000 sequences. The learning task is to recognize coding regions accurately.

### Protein secondary structure

There have been quite a number of research attempts to use machine learning techniques to identify conformational states in the protein secondary structure from amino acid sequences. The task is to learn the rules governing the formation of, say an $\alpha$-helix, given a particular amino acid sequence. That is, given an amino acid sequence, we want to predict the conformational state around the mid point (usually) of that sequence. The *windowing* technique is commonly used for generating training sequences. Each training sequence consists of a fixed number of neighboring amino acids in sequence and a *window*, a fixed number of amino acids considered as a subsequence. The window slides over the protein sequence, one amino acid at a time, to generate different training sequences. The window size varies according to the method applied in different tasks.

The protein secondary structure (SS) data set (courtesy of Qian and Sejnowski

(1988)) contains sequences of amino acids and the secondary structures at the corresponding positions. There are three structures (*alpha-helix, beta-sheet, and coil*) and 20 amino acids (21 attributes, including 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 SS data set has 21,625 sequences.

These three data sets represent different degrees of difficulty in learning an accurate concept. Typical learning algorithms can achieve an accuracy of 90+% in the splice junction data set, 70+% in the protein coding region data set, and 50+% accuracy in the protein secondary structure data set. The learning task is to identify the different secondary structures.

### 4.2.2 Artificial data

A fourth data set was also employed. This artificial (ART) data set has 10,000 instances randomly generated from a disjunctive boolean expression that has 4 symbolic (26 values) and 4 numeric (1,000 values) variables. The expression is:

$$(x_0 < `N' \wedge x_1 < 500 \wedge x_2 < `N') \vee (x_1 > 500 \wedge x_2 > `N' \wedge x_3 > 500) \vee$$
$$(x_4 < `N' \wedge x_5 < 500 \wedge x_6 < `N') \vee (x_5 > 500 \wedge x_6 > `N' \wedge x_7 > 500),$$

where $x_0$, $x_2$, $x_4$, and $x_6$ are symbolic variables with values 'A' through 'Z', and $x_1$, $x_3$, $x_5$, and $x_7$ are numeric variables with values 0 through 999. A total of $4.6 \times 10^{17}$ instances are possible. Arbitrarily large data sets can be generated at will and are used in experiments destinated for measuring efficiency performance.

Although these data sets (except the artificial data sets used in efficiency experiments) are not very large data sets, they do provide us with an idea of how our strategies behave in practice. Since the data sets are sufficiently small, we are able to generate base line statistics on the accuracy of each learning algorithm we have chosen to use in this study. Otherwise, using a massive database would imply that we have unbounded resources and time in order to compute baseline statistics. As we have noted (as well as (Catlett, 1991)) this might take many years of computing. Furthermore, scaling studies are possible on these smaller sets simply by varying the number and size of the subsets formed in the initial data reduction schemes and extrapolating. However, larger data sets are being sought for use in this study which will be the focus of our work. Stated another way, *if we cannot display useful and interesting results of meta-learning on these small test cases, larger-scale studies are probably not warranted.*

## 4.3 Experimental Methodology

One of the more common techniques used in evaluating the accuracy of a learning program is *cross validation* (Breiman *et al.*, 1984). In an $n$-fold cross validation, the entire data set is divided into $n$ disjoint subsets and $n$ train-and-test runs are performed. In each run, two disjoint sets are formed: a training set and a test set. One of the $n$ subsets form the test set and the remaining $n-1$ subsets merged to form the training set. A classifier is generated by applying a learning algorithm to the training set and is evaluated on the test set. Note that the classifier is evaluated on data not used in training. That is, the accuracy obtained from the test set estimates the accuracy/predictiveness of the learned classifier. A different subset (out of $n$ subsets) is used as the test set (hence a different training set) in each of the $n$ runs. The accuracies of the $n$ different classifiers measured over the $n$ different test sets are averaged as the final prediction accuracy for the learning algorithm employed.

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

To simulate the multiple-site scenario, in our experiments, we divided the training set into equi-sized subsets (each subset representing a site) and varied the number of subsets (sites) from 2 to 64. *Base-classifiers* are learned from these subsets. In most experiments we also ensured that each subset 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). Prediction accuracy of meta-learned classifiers (or *meta-classifiers*) are compared to the accuracy of *global classifiers*, which are learned from the entire data set before partitioning. As the number of subsets increases, each of the subset becomes smaller, which result in significant loss of information. Therefore, it is particularly important that the meta-classifiers are at least as as accurate as the global classifiers.

## 4.4 Limitations in Experiments

Experiments in this thesis were performed at different time periods (not in chapter order) and the choice of different learning tasks and algorithms in different sets of experiments changed when more learning tasks and algorithms were acquired. WPE-BLS was used in earlier studies and was later excluded because it is significantly slower than the other algorithms and it needs much more space to store its classifiers since each of the classifiers stores the corresponding training examples. CN2 was included in our studies later when it became available. Earlier experiments focused on ID3 and CART because they are the more popular learning algorithms. In fact, they were used in almost every experiment.

Data for splice junctions and secondary structures were used in earlier experiments. Data for protein coding regions were later included when it became available. Since various learning algorithms are not suitable to generate accurate classifiers for the secondary structure data (as experienced by other researchers), the data set was not used in later experiments. The artificial data set was added to our study for large-scale studies when we could not obtain a real-world data set large enough for our scalability experiments.

Definitive results can only be obtained from very large-scale experiments, which are beyond the scope of this study. Our results are limited to the different learning algorithms and tasks employed in our experiments. However, each set of experiments was performed on quite a number of different *combinations* of learning tasks, learning algorithms, and other parameters. That is, our results do provide some degree of generality.

# Chapter 5

# One-level Meta-Learning on Partitioned Data

As we discussed previously, our approach to solve the scaling problem is to partition the data set into smaller subsets, apply learning algorithms to each subset, followed by a meta-learning phase that combines the learned results. In this study we focus on using the same learning algorithm on each of the subsets as well as for generating the integrated (meta-learned) structures. Figure 5.1 depicts this process with the same learning algorithm $L$ generating classifiers $C_1$ through $C_n$ from training data subsets $T_1$ through $T_n$. *Multistrategy meta-learning* (using multiple learning algorithms) will be discussed in Chapter 10. 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. 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

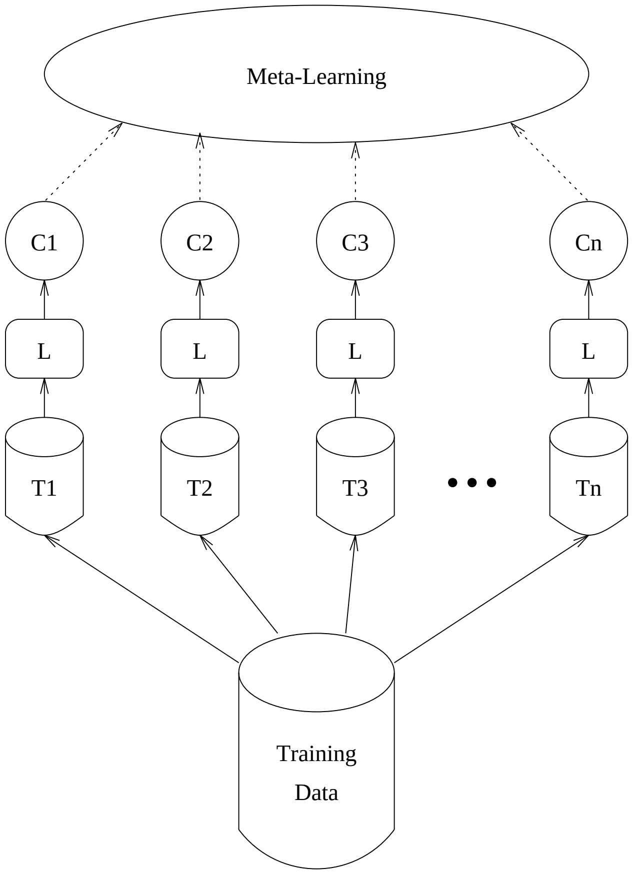


Figure 5.1: Meta-learning from partitioned data.

(sometimes called "boosting" (Schapire, 1990)).

In this chapter we study different techniques for integrating predictions generated by a set of *base classifiers*, each of which is computed by a learning algorithm applied to a distinct partitioned data subset. Common voting and statistical techniques are evaluated. Techniques described in Section 2.1.2 that are included in this study are:

- Voting
- Weighted Voting
- Weighted Majority (WM) (Littlestone & Warmuth, 1989)
- Weighted Majority-Limit (WML) (Littlestone & Warmuth, 1989)
- Weighted Majority-Random (WMR) (Littlestone & Warmuth, 1989)

- Bayesian Belief (Xu *et al.*, 1992)

These familiar techniques are empirically compared to our proposed meta-learning techniques (*arbiter, class-combiner, and class-attribute-combiner*) described in Chapter 3. All the meta-learning strategies discussed so far have only one level of meta-learning to create the integrating structures. Hence, we characterize these strategies as *one-level* meta-learning methods (Chan & Stolfo, 1995a).

## 5.1 Issues

Before we compare the other techniques in the literature with ours, several issues have to be addressed and are discussed as follows.

**Number and size of training subsets:** The number of initially partitioned training data subsets largely depends on the number of processors available, the inherent distribution of data across multiple platforms (some possibly mobile and periodically disconnected), the total size of the available training set, and the complexity of the learning algorithms. The available resources at each processing sites naturally defines an upper bound on the size of each subset. If the number of subsets exceeds the number of processors available, each processor can simulate the work of multiple ones by serially executing the task of each processor. Another consideration is the desired accuracy we wish to achieve. As we will see in our experimental results, there may be a tradeoff between the number of subsets and the final accuracy of a meta-learning system. Moreover, the size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective base classifier in the initial stage of training.

**Distribution of examples, disjoint or replicated:** Since a totally random distribution of examples may result in the absence of one or more classes in the partitioned data subsets, the classifiers formed from those subsets will be ignorant about those classes. That is, more "disagreements" may occur between classifiers, which leads to larger arbiter training sets. Maintaining the class distribution in each subset as in the total available training set may alleviate this problem. The classifiers generated from these subsets may be closer in behavior to the global classifier produced from the entire training set than those trained on random class distributions. In addition, disjoint data subsets promote the maximum amount of parallelism and hence are more desirable. Yet partial replication (Chan & Stolfo, 1996a) may mitigate the problem of extreme bias potentially introduced by disjoint data.

**Strategies:** There are indeed many strategies for arbitration and combining as detailed here, each impacting the size of training data required to implement them effectively. Several experiments were run to determine the relative effectiveness of some of these strategies. They vary in the type of information or biased distributions of training data the arbiter is allowed to see. Thus far, the meta-learning strategies we discussed are applied solely to a single collection of base classifiers. (These are called "one-level" meta-learners.) We also studied building hierarchical structures in a recursive fashion, i.e., meta-learning arbiters and combiners from a collection of "lower level" arbiters and combiners. These hierarchical classifiers attempt to improve the prediction accuracy that may be achieved by one-level meta-learned classifiers and are described in Chapter 6.

## 5.2 Experiments and Results

Different combinations of two learning algorithms (ID3 and CART) and two data sets (Splice Junctions and Protein Coding Regions) were explored in our experiments. The experimental procedures in Section 4.3 were followed and results are summarized in the following sections.

### 5.2.1 Voting, statistical, and meta-learning techniques

We first consider whether meta-learning performs as well as the common voting and Bayesian techniques reported in the literature. In our experiments, we varied the number of equi-sized subsets of training data from 2 to 64 ensuring each was disjoint but with proportional distribution of examples of each class. The size of a validation set used for generating the *integrating structures* (weights/probabilities/arbiters/combiners) is twice the size of the underlying training set for a base classifier. (Since the arbiter approach selects a subset of the validation set, a larger validation set size than the base set size provides more training examples at the meta-level.) The prediction accuracy on a separate test set is our primary comparison measure. The different strategies were run on the two data sets with the two learning algorithms. The results from the splice junctions data set are plotted in Figure 5.2 and the protein coding regions data set in Figure 5.3. In each figure the first row of graphs depicts results from the different integrating techniques using ID3 and the second row using CART. The accuracy for the *global classifier* is plotted as "one subset," meaning the learning algorithm was applied to the entire training set to produce the baseline accuracy results for comparison. The average accuracy of the base classifiers for each number of subsets is also plotted, labeled as "avg-base." By way of comparison, the average accuracy of the most accurate base classifiers is plotted as "max-base." The plotted accuracy is the average of 10-fold cross-validation runs.

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

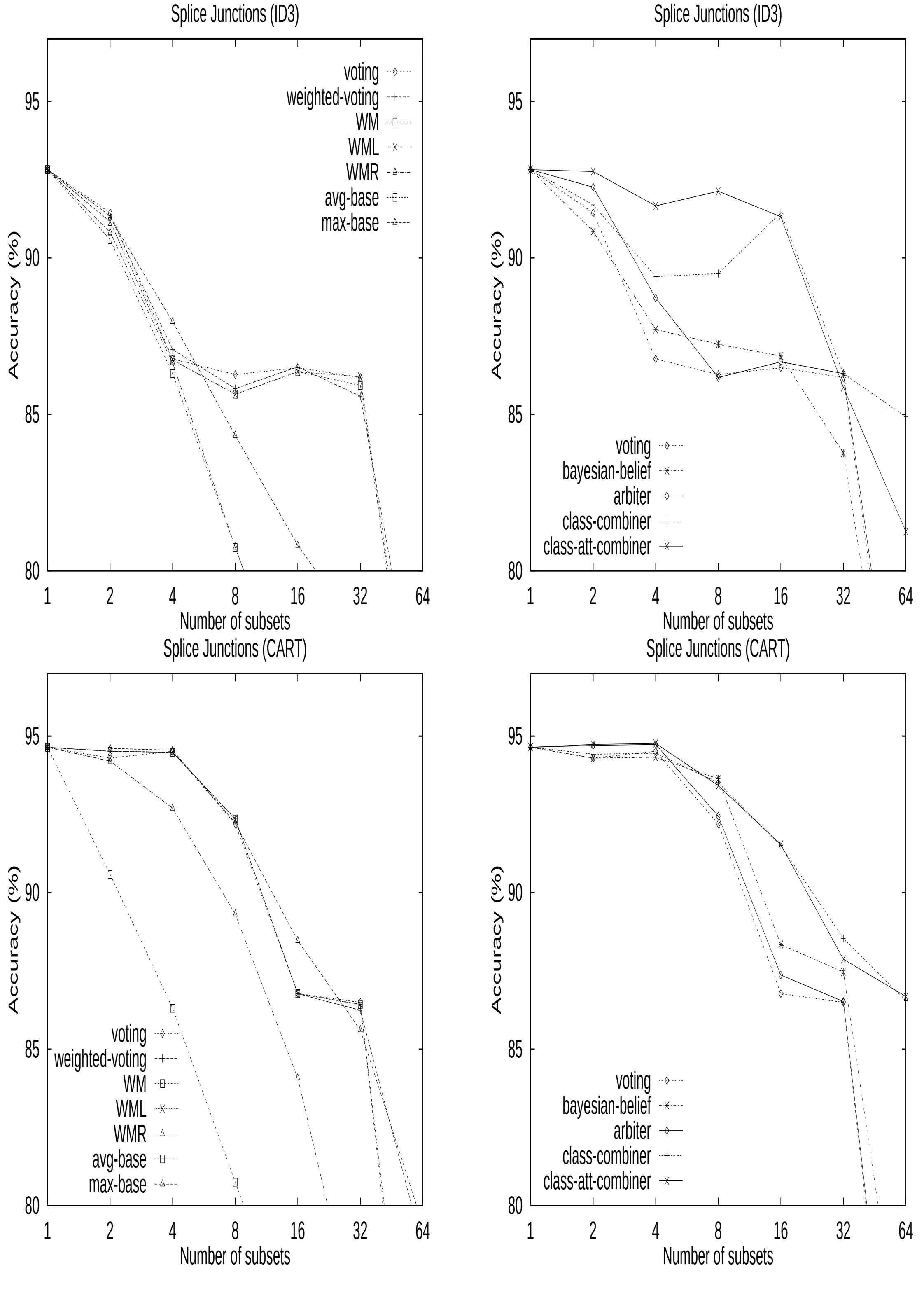


Figure 5.2: Accuracy for the one-level integrating techniques in the splice junctions domain.

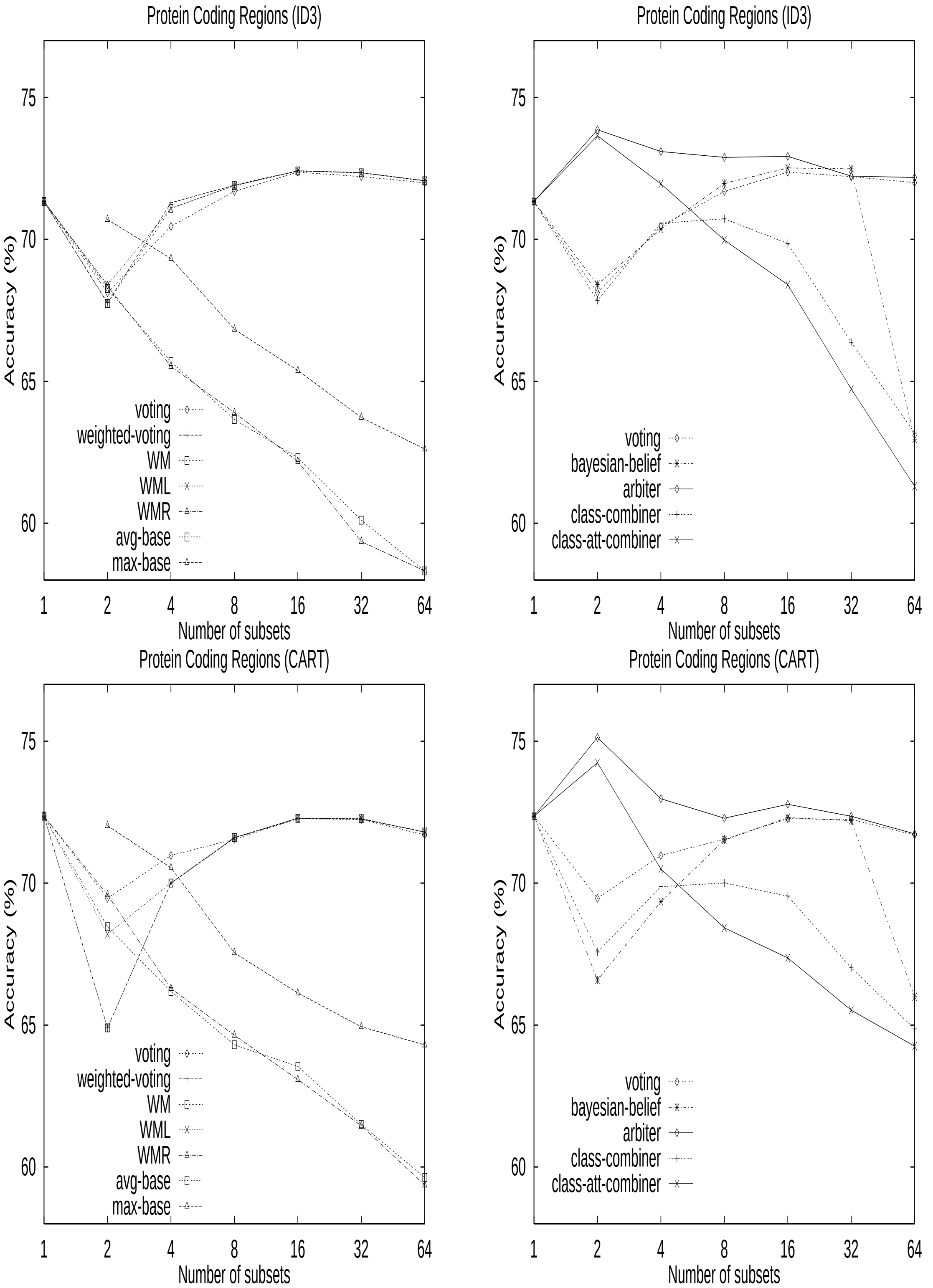


Figure 5.3: Accuracy for the one-level integrating techniques in the protein coding regions domain.

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

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

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

As we observe in the SJ domain, none of the schemes can maintain the baseline accuracy when the number of subsets increases. Next, we investigate the relaxation of the disjoint subset property.

### 5.2.2 Partitioned data with replication

In our previous set of experiments the accuracy level of the global classifier cannot always be achieved. One possible problem is the lack of sufficient data in each disjoint subset. To solve this problem, we allow each data partition to have some amount of replicated data from other partitions. We prepare each learning task by generating

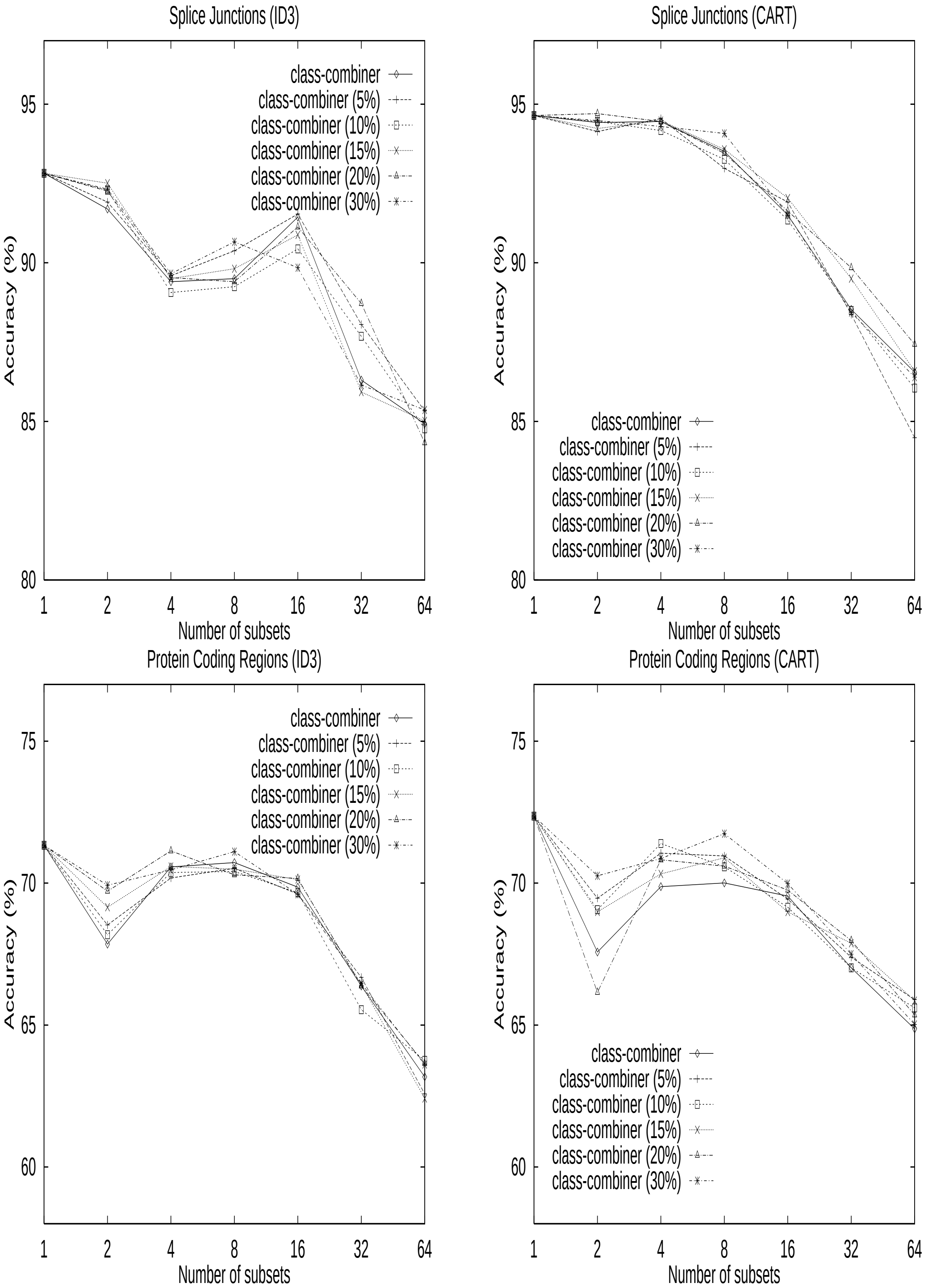


Figure 5.4: Accuracy for the *class-combiner* scheme trained over varying amounts of replicated data. $\Delta$ ranges from 0% to 30%.

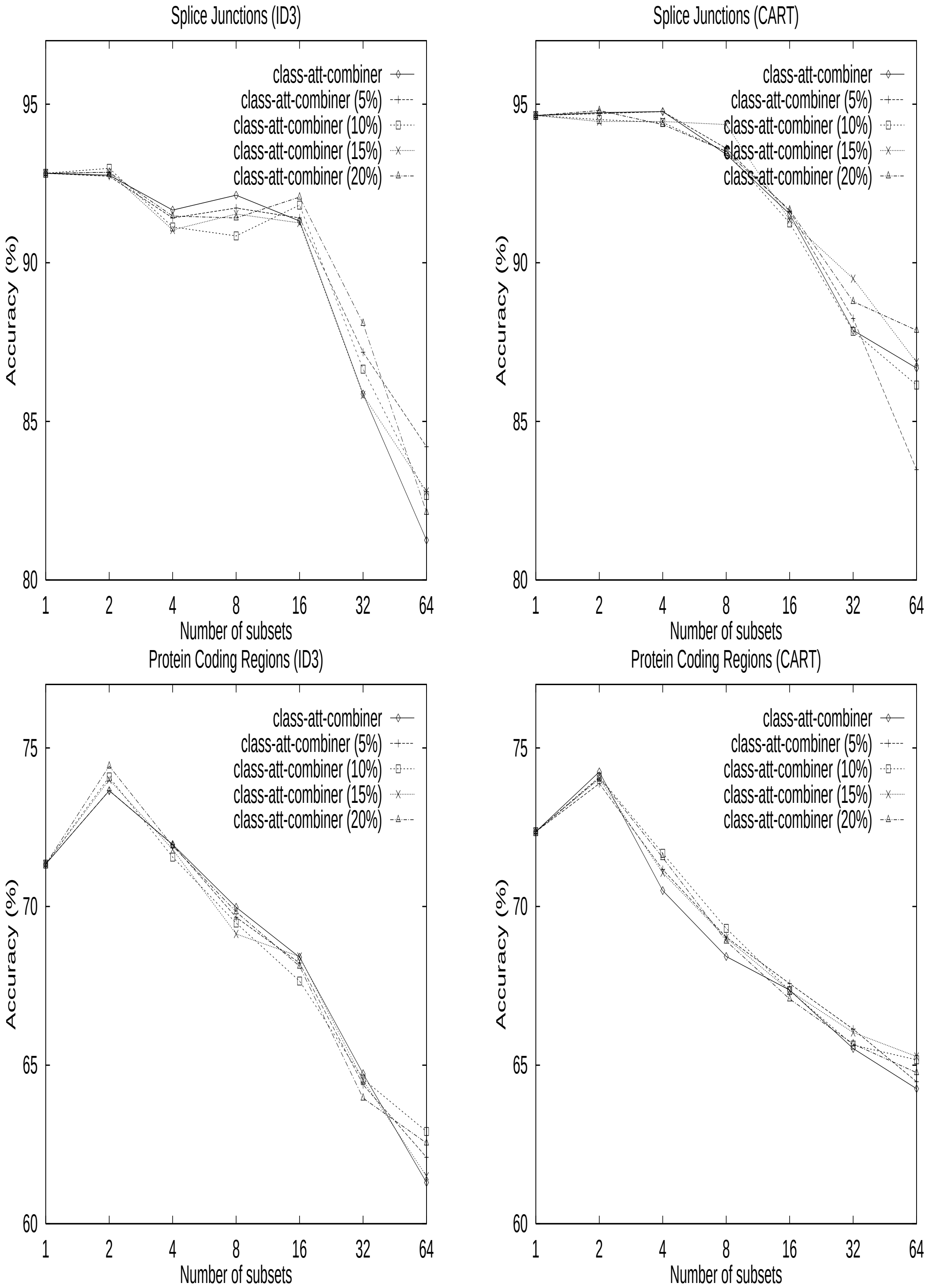


Figure 5.5: Accuracy for the *class-attribute-combiner* scheme trained over varying amounts of replicated data. $\Delta$ ranges from 0% to 30%.

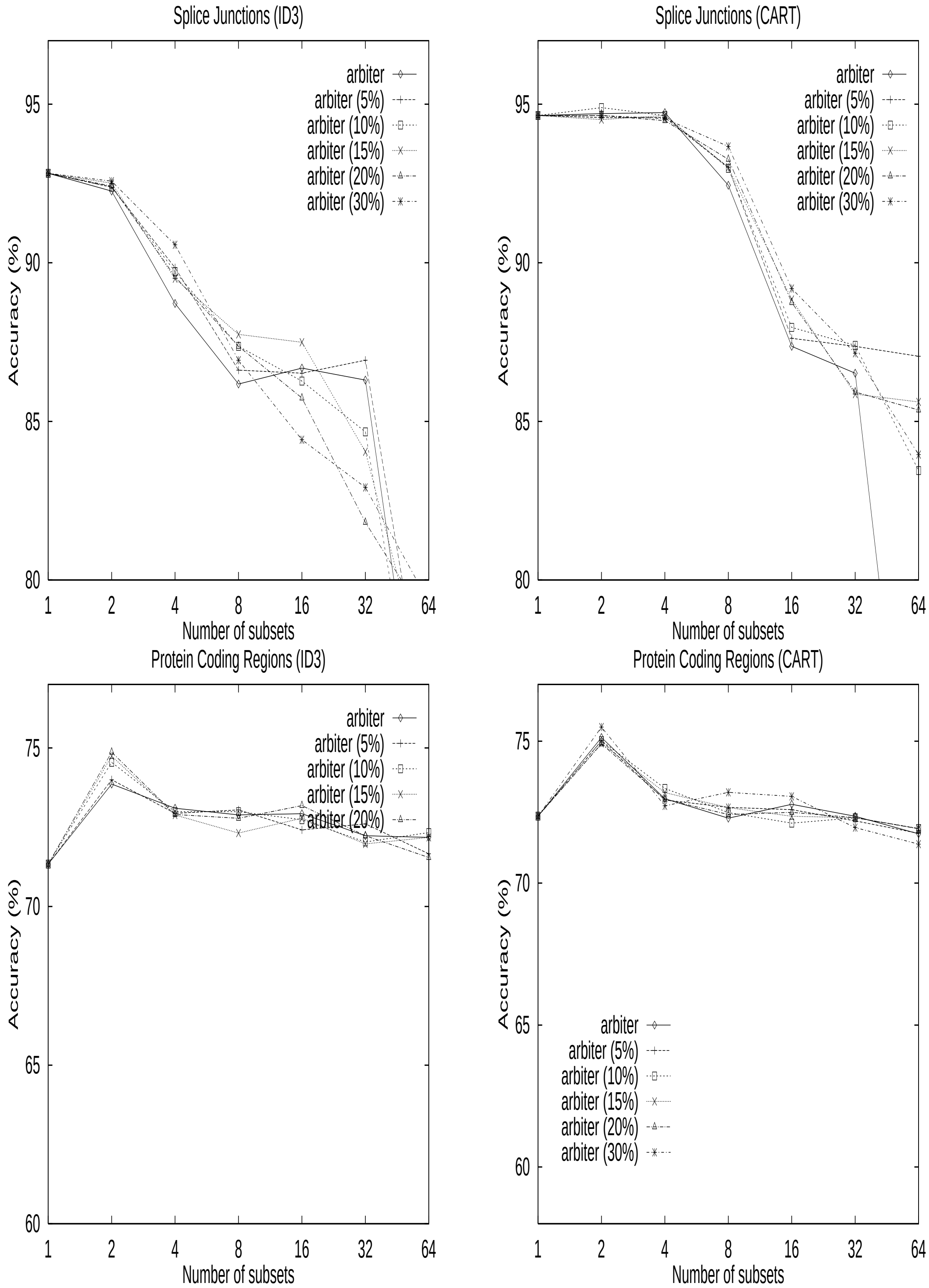


Figure 5.6: Accuracy for the *arbiter* scheme trained over varying amounts of replicated data. $\Delta$ ranges from 0% to 30%.

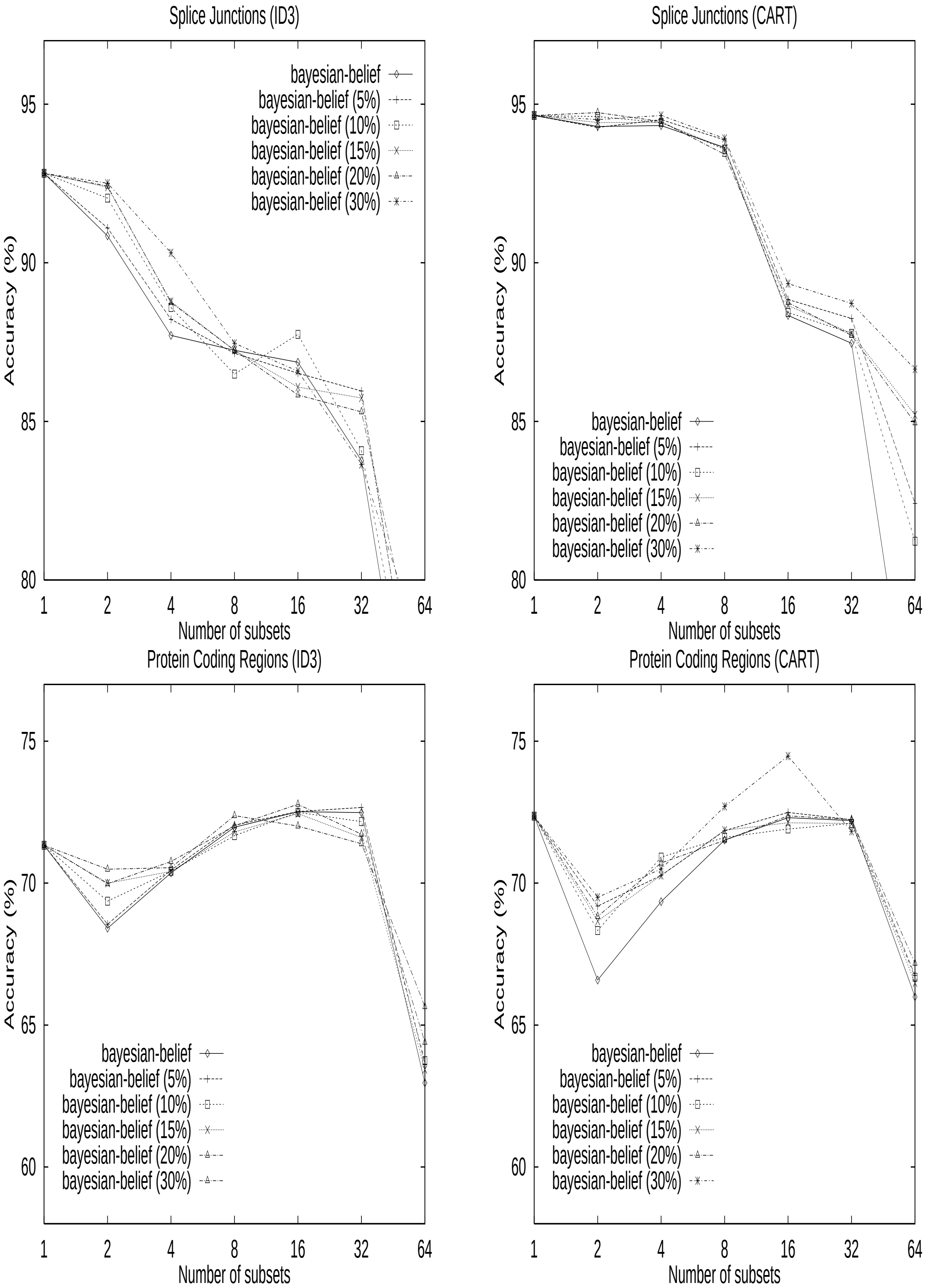


Figure 5.7: Accuracy for the *bayesian-belief* scheme trained over varying amounts of replicated data. $\Delta$ ranges from 0% to 30%.

subsets of training data for the base classifiers according to the following generative scheme (Chan & Stolfo, 1996a):

1. Starting with $N$ disjoint subsets, randomly choose from any of these sets one example $X$, distinct from any other previously chosen in a prior iteration.

2. Randomly choose a number $r$ from $1...(N-1)$, i.e. the number of times this example will be replicated.

3. Randomly choose $r$ subsets (not including the subset from which $X$ was drawn) and assign $X$ to those $r$ subsets.

4. Repeat this process until the size of the largest (replicated) subset is reached to some maximum (as a percentage, $\Delta$, of the original training subset size).

In the experiments reported here, $\Delta$ ranged from 0% to 30%. Again, we used different combinations of two learning algorithms (ID3 and CART) and two learning tasks (Splice Junctions and Protein Coding Regions). Each set of incremental experimental runs, however, chooses an entirely new distribution of replicated values. No attempt was made to maintain a prior distribution of training data when incrementing the amount of replication. This "shot gun" approach provides us with some sense of a "random learning problem" that we may be faced with in real world scenarios where replication of information is likely inevitable or purposefully orchestrated.

The graphs in Figures 5.4 through 5.7 plot the results for the *class-combiner*, class-attribute-combiner, *arbiter*, and *bayesian-belief* schemes. The results in all cases are conclusive: *replication essentially buys nothing!* In each case no measurable improvement in predictive accuracy is seen no matter which learning algorithm or combining scheme is used.

These negative results for replication are in fact positive from the perspective of computational performance! One may presume that applying a number of instances of a learning algorithm to disjoint training data results in a set of base classifiers each biased towards its own partition of data. Combining two or more such biased base classifiers by meta-learning attempts to share knowledge among the base classifiers and to reduce each individual's bias. Replication of training data is an alternative attempt to reduce this bias. Common or shared information replicated across subsets of training data at the onset of learning attempts to provide each learned base classifier with a "common view" of the learning task. The results here show that meta-learning from disjoint training data does an effective job of sharing knowledge among separate classifiers anyway. In fact, the overhead that may be attributed to replicated data (since the same data is being treated multiple times by separate learning processes) may be comfortably avoided, i.e. meta-learning on purely disjoint data seems to achieve good performance, at perhaps optimal speeds due to optimal data reduction.

These rather surprising results are of course limited to the learning algorithms and data sets used in this study. Further support for this behavior is found in experimental results reported in Section 7.3.

## 5.3 Summary

We systematically compare schemes reported in the literature to our proposed meta-learning techniques and demonstrate empirically that the arbiter scheme produces more accurate trained classifiers than the other schemes. However, we observe that our techniques (and others) cannot always maintain the baseline accuracy (of the global classifiers) when the number of data subsets increases. Moreover, partially replicating some of the data from other subsets in each subset does not alleviate the problem. As a result, from a multiprocessing perspective, each concurrent base learning process need not consume more data than necessary.

All the techniques presented so far are *one-level* methods. They only perform one level of processing to generate the integrating structures. In the next chapter we consider the behavior of *hierarchical meta-learning* structures.

# Chapter 6

# Hierarchical Meta-Learning on Partitioned Data

In this chapter 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 the previous chapter we demonstrate that our meta-learning techniques outperform the voting-based and statistical techniques in terms of prediction accuracy. However, the one-level techniques cannot always achieve the same level of accuracy as the global classifier. Here we describe our *hierarchical* (multi-level) meta-learning methods called *arbiter tree* and *combiner tree* (Chan & Stolfo, 1993d; Chan & Stolfo, 1995b). We empirically compare these 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.

We first present our hierarchical meta-learning methods in Sections 6.1 and 6.2. Section 6.3 examines some related work. Evaluation of our schemes based on experimental results are discussed in Sections 6.4 and 6.5.

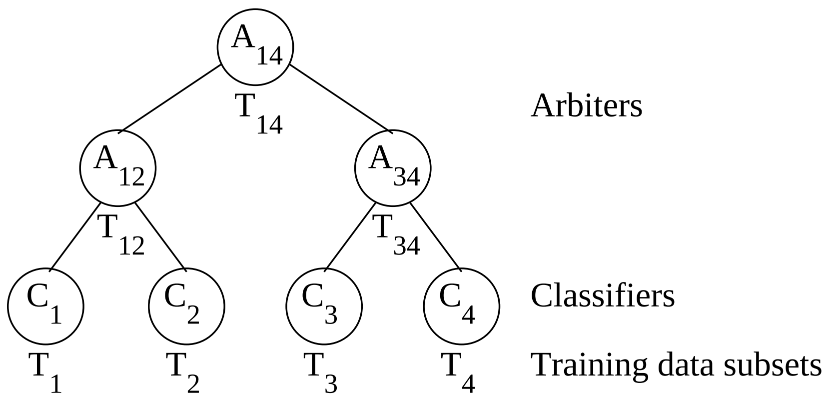


Figure 6.1: Sample arbiter tree.

## 6.1 Arbiter Tree

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

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

Suppose there are initially four training data subsets ($T_1 - T_4$), processed by some learning algorithm, $L$. First, four classifiers ($C_1 - C_4$) are generated from four instances of $L$ applied to $T_1 - T_4$. The union of the subsets $T_1$ and $T_2$, $U_{12}$, is then classified by $C_1$ and $C_2$, which generates two sets of predictions, $P_1$ and $P_2$. A *selection rule* as detailed earlier generates a training set ($T_{12}$) for the arbiter from the predictions $P_1$ and $P_2$, and the subset $U_{12}$. The arbiter ($A_{12}$) is then trained from the set $T_{12}$ by algorithm $L$. Similarly, arbiter $A_{34}$ is generated from $T_3$ and $T_4$ and hence all the first-level arbiters are produced. Then $U_{14}$ is formed by the union of subsets $T_1$ through $T_4$ and is classified by the arbiter trees rooted with $A_{12}$ and $A_{34}$. Similarly, $T_{14}$ and $A_{14}$ (root arbiter) are generated and the arbiter tree is complete. The resultant tree is depicted in Figure 6.1.

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

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

### 6.1.1 Discussion

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

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

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

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

## 6.2 Combiner Tree

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

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

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

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

## 6.3 Related Work

Brodley (1995) and Tcheng et al. (1989) also build trees with a classifier in each of the tree nodes. Their *top-down* tree building (root to leaves) approach is the same as the one used in common decision tree algorithms like ID3 (Quinlan, 1986). Like training the tree, classification of instances using the tree is also performed in a top-down fashion. On the contrary, our arbiter and combiner trees are built *bottom-up* (leaves to root) and classification of instances is performed bottom-up as well. Their approach in training is to recursively partition the space of examples and heuristically choose a learning algorithm to generate a classifier from each subspace. That is, the system has to initially process the entire set of examples to generate the classifier at the root node. Their goal is to build a hybrid classifier in a tree structure to improve accuracy (not training efficiency). On the other hand, our goal is to improve training efficiency and the tree structure is built to integrate base classifiers learned from data subsets at the leaf level. The entire set of examples is never used to generate a classifier that is part of the final tree structure—only subsets are used.

A number of experiments were performed on the arbiter tree and combiner tree strategies. The experiments and their results for the arbiter tree strategy are discussed next, followed by those for the combiner tree strategy.

## 6.4 Experimental Results for Arbiter Tree

We ran a series of experiments to test our strategies based on the splice junction prediction task described in Section 4.2.1. Four different learning algorithms (ID3, CART, WPEBLS, and BAYES) were used to show that our strategies are applicable to diverse algorithms. The prediction accuracy on the test set is our primary comparison measure. All the empirical results presented in this paper are averages from five-fold cross-validation runs (except in the experiments for random partitioning, which is further discussed in Section 6.4.4). That is, the entire training set is divided into five partitions, each partition takes turn in being the test set and the remaining partitions constitute the training set. We varied the number of subsets from 2 to 64 and the equal-size subsets were disjoint with proportional partitioning of classes. The accuracy for the global classifier as "one subset."

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

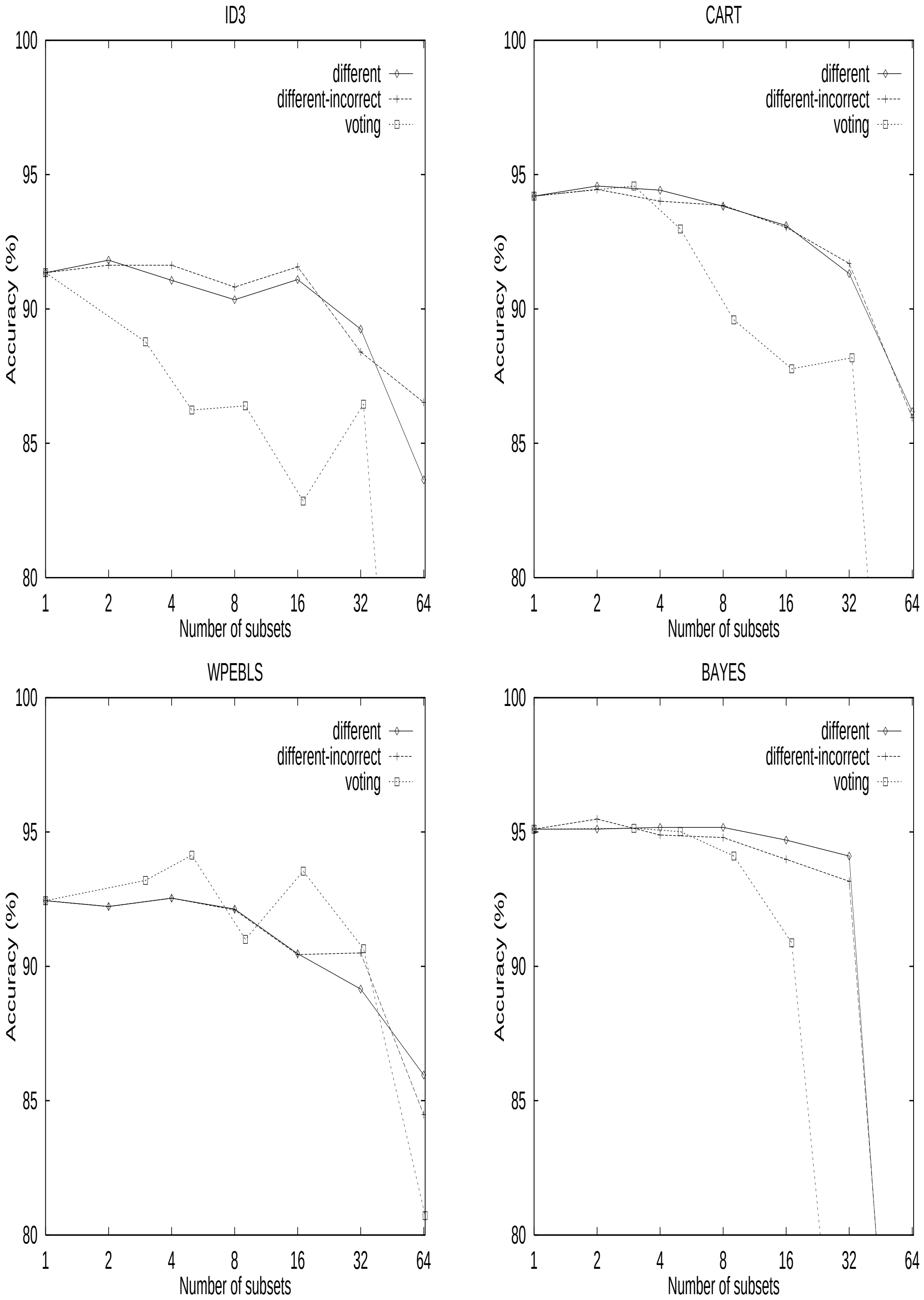


Figure 6.2: Results on different arbiter schemes.

### 6.4.1 Bounded arbiter training sets

Two arbiter schemes (*different-arbiter* and *different-incorrect-arbiter*) were run on the splice junction data set with the four learning algorithms. In addition, we applied a simple voting scheme on the leaf classifiers for comparison.

In Figure 6.2, for the two arbiter schemes, we observe that the accuracy slightly decreased when the number of subsets increased. With 64 subsets, most of the learners exhibited at most an 8% drop in accuracy, with the exception of BAYES. The sudden drop in accuracy in BAYES was likely due to the lack of information in the training data subsets. In the splice junction data set there are only $\sim$ 40 training examples in each of the 64 subsets. If we look at the case with 32 subsets ($\sim$ 80 examples each), all the learners sustained a drop in accuracy of at most 3%. This shows that the data subset size cannot be too small. The voting scheme performed poorly. Furthermore, the two arbiter schemes had comparable performance and since the *different-arbiter* scheme produces fewer examples in the arbiter training sets, it is the preferred scheme. This scheme is also our default scheme–when a particular arbiter scheme is not specified, the *different-arbiter* scheme is assumed.

### 6.4.2 Order of arbiter trees and training set size limit

We performed experiments on the splice junctions and protein coding regions data to evaluate the arbiter trees of different orders. Again, we varied the number of subsets from 2 to 64 and measured the prediction accuracy on a disjoint test set. The plotted results in Figure 6.3 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 the plots display a drop in accuracy when the number of subsets increases. Also, the higher order trees are generally less accurate than the lower ones. However, in the protein coding region data set experiments the accuracy is maintained, or exceeded in some circumstances, regardless of the order of the trees.

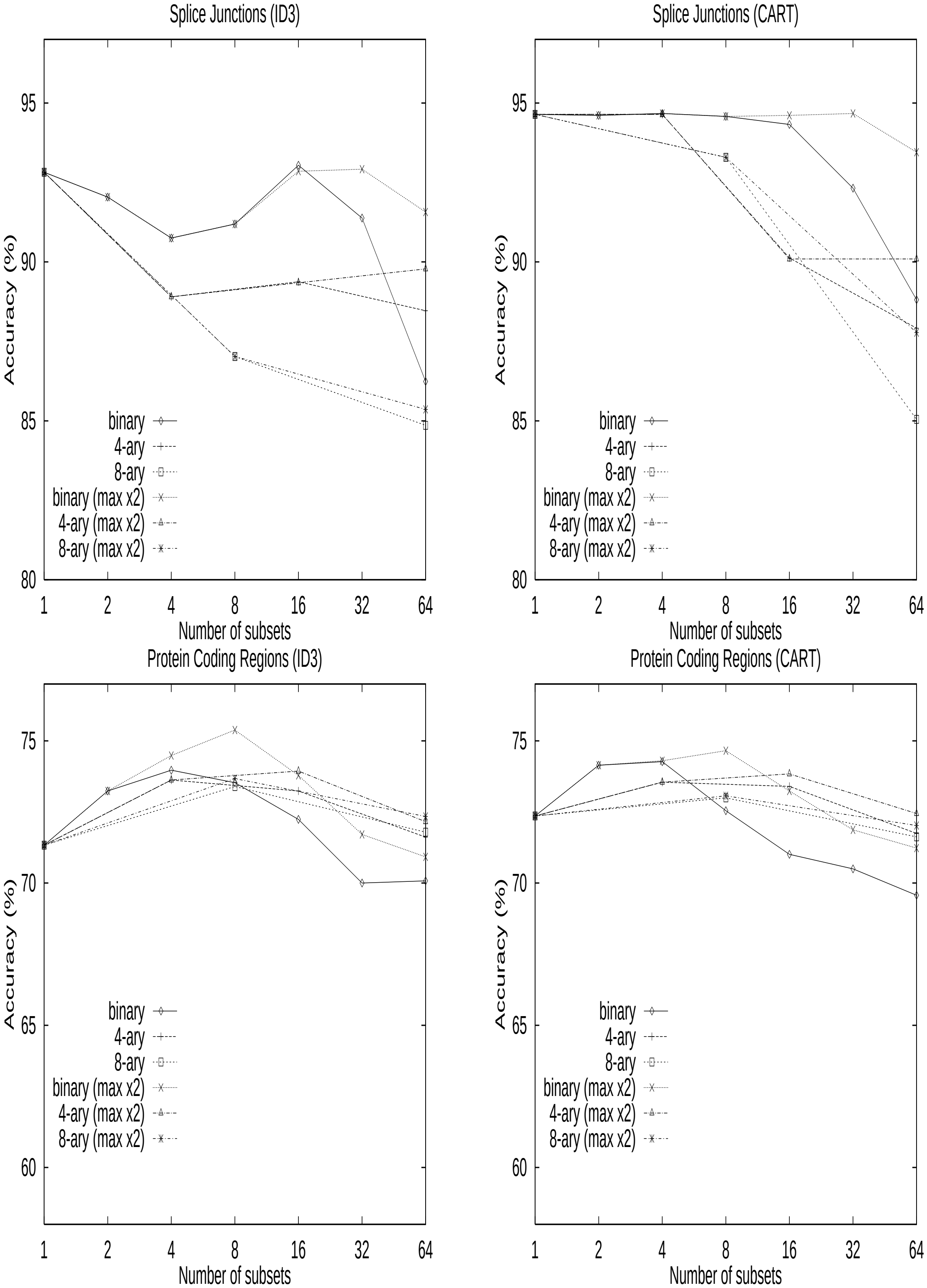


Figure 6.3: Accuracy for different orders of arbiter trees and limits for training set size.

Recall that at each tree level, the size of the arbiter training set is fixed to the size of a data subset used in training the base classifiers. If we relax the restriction on the size of the data set for training an arbiter, we might expect an improvement in accuracy at the expense in processing time. To test this hypothesis, a set of experiments was performed to double the maximum training set size for the arbiters. As we observe in Figure 6.3, 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. For 4-ary and 8-ary trees, the accuracy results show no significant improvement. However, this multi-level arbiter tree approach does demonstrate an accuracy improvement over the *one-level* techniques, which generally cannot maintain the accuracy obtained from the whole data set in our experiments.

### 6.4.3 Unbounded arbiter training sets

If we *further* relax the restriction on the size of the data set for training an arbiter, we might expect additional improvement in accuracy, but decline in execution speed. Again, the different sizes are constant multiples of the size of a data subset. We evaluated sizes that doubles and triples the subset size. In one set of experiments the size limit was lifted. The results plotted in Figure 6.4 were obtained from using the *different-arbiter* scheme on the splice junction data using four different learning algorithms.

As we expected, by increasing the maximum arbiter training set size, higher accuracy can be achieved. A significant improvement is observed when the maximum size is just two times the size of the original subsets. As discussed in the previous set of experiment, doubling the set size roughly maintains the accuracy of the global classifier (except in for the BAYES algorithm with 64 subsets). Further increase in size limit yields smaller improvement. When the maximum size is unlimited (i.e., allowing each arbiter to be trained on the entire union set), the accuracy is the highest. In fact, we observed an increase of 2% in accuracy for ID3 with 64 subsets.

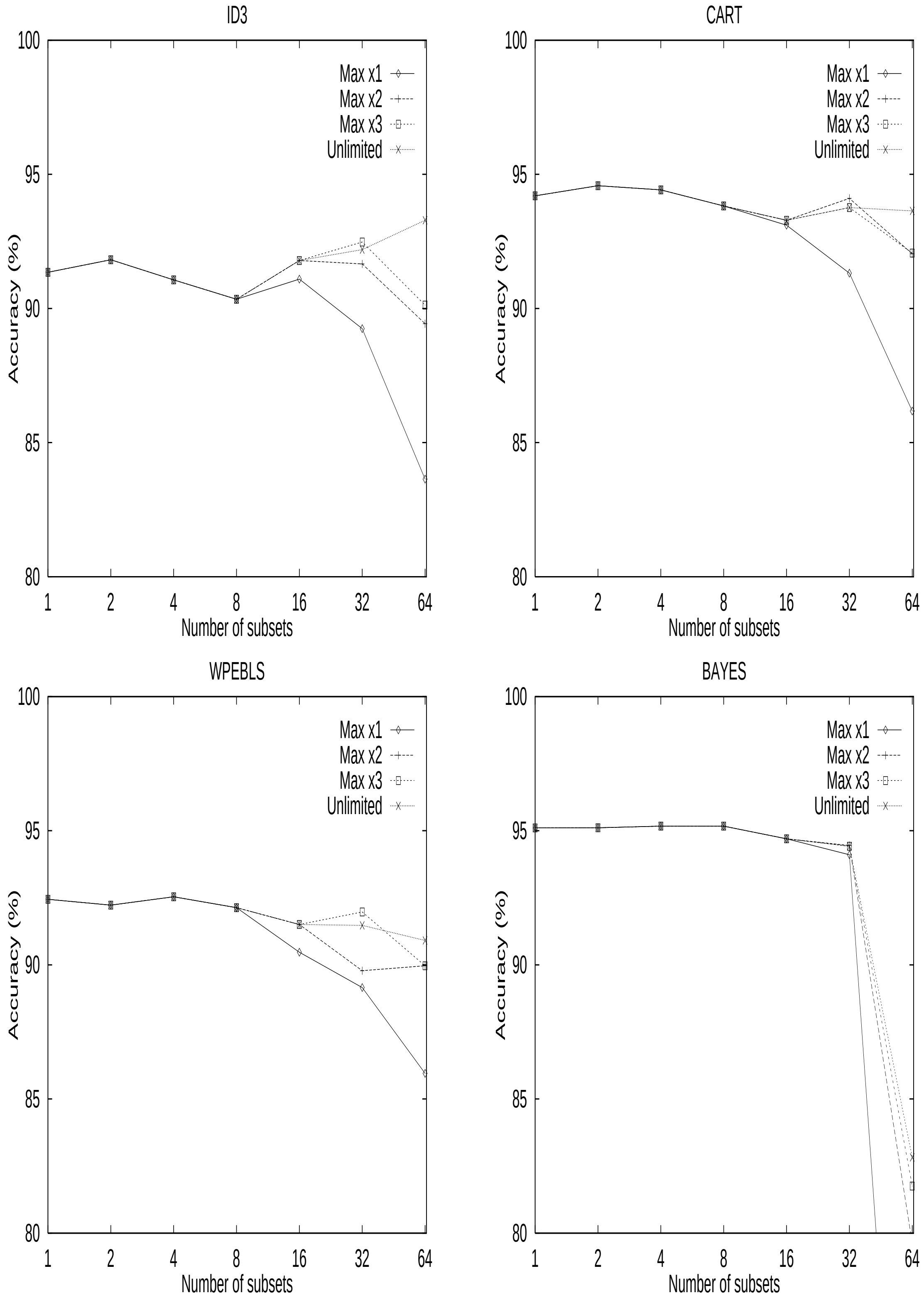


Figure 6.4: Results on different maximum arbiter training set sizes.

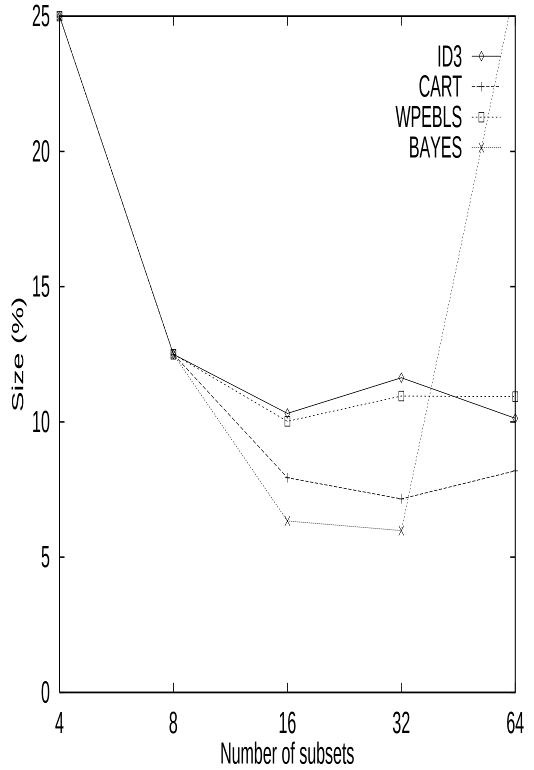


Figure 6.5: Largest set sizes with unlimited maximum arbiter training set size.

Next, we investigate the size and location of the largest arbiter training set in the entire arbiter tree. (Recall, an arbiter training set is produced by a selection rule.) This gives us a notion of the memory requirement at any processing site and the location of the main potential bottleneck during meta-learning. Our empirical results presented in Figure 6.5 indicate that the largest arbiter training set size was never significantly greater than 10% of the total training set (except for BAYES with 64 subsets) and always happened at the root level, independent of the number of subsets at the leaves (that was greater than four). (Note that when the number of subsets is two and four, the training set sizes are 50% and 25%, respectively, of the original set at the leaves and become the largest in the tree.) This implies that the bottleneck was in processing around 10% of the entire training data set at the root level. This also implies that our arbiter tree strategy required only around 10% of the memory used by the serial case at any single processing site. This has a significant impact on scalability. Suppose a single processor is limited in memory and able to solve a learning task of size $n$. Our experiments suggest that meta-learning allows that single processor to solve a problem of size $10n$. (Strategies for reducing the largest arbiter training set size even further are discussed in the next section.) Recall that the accuracy level of this strategy is roughly the same as the serial case. Thus, the arbiter tree strategy (with no restrictions on the arbiter training set size) can perform the same job as the serial case with less time and memory without parallelizing the learning algorithms. With restricted training set sizes, our strategies can theoretically scale to arbitrarily large problems by setting the size restriction to the memory capacity of a single processor and using more processors.

In summary, when the arbiter training set size is bounded to the size of each initial training data subset, a small degradation in prediction accuracy (at most 3%) was observed with 32 subsets. A further increase in the number of subsets (64 subsets) produced a much larger decline in accuracy. This indicates that each of the subsets cannot be too small in the training of the initial classifiers. Accuracy was preserved when the bound on the size of the arbiter training set was lifted. However, we observe that the size of the arbiter training sets was limited to about 10% of the entire training set in the splice junction domain. Recall that the arbiter training sets consist of "disagreed" instances, hence, classifiers with higher error rates and/or significantly diverse behaviors will have a size limit larger than 10%.

### 6.4.4 Reducing the largest arbiter training set size

As mentioned in the previous section, we discovered that our scheme required at most 10% of the entire training set at any processing site to maintain the same prediction accuracy as in the global classifier case for the splice junction data. However, the percentage is dependent on several factors: the prediction accuracy of the algorithm on the given data set, the partitioning of the data in the leaf subsets, and the *pairing* of learned classifiers and arbiters at each level.

### Class partitioning

If the prediction accuracy is high, the arbiter training sets will be small because the predictions will usually be correct and few disagreements will occur. In our earlier experiments reported in (Chan & Stolfo, 1993d), the partitioning of data in the subsets was random and later we discovered that half of the final arbiter tree was trained on examples with only two of the three classes. That is, half of the tree was not aware of the third class appearing in the entire training data. We postulate that if the class partitioning in the subsets is proportional, the leaf classifiers and arbiters in the arbiter tree will be more accurate and hence the training sets for the arbiter will be smaller. Indeed, results from experiments reported in here significantly lower the largest size observed from 30% to 10%. We ran additional experiments on training sets with a more randomized partitioning scheme. A randomly chosen training set is used in each run and the results averaged from five runs are presented in Figure 6.6. As one might expect, a "truly randomized" partitioning scheme approximates our *proportional partitioning* scheme and therefore the accuracy obtained using the two schemes should be roughly the same. Indeed the accuracy curves in Figure 6.6 are very close.

### Classifier Pairing

Some experiments were performed on the two pairing strategies applied only at the leaf level and the results are shown in Figure 6.7. All these experiments used the *different-arbiter* scheme for meta-learning arbiters. Different pairing schemes were used with proportional partitioning and "non-random" partitioning of classes. In non-random partitioning, examples are not proportionally partitioned according to their classes and each partitioned subset is usually dominated by examples of a single class. In addition, with the no (or "neighbor") pairing schemes, a class might be absent from half of the arbiter tree. The pairing schemes with proportional partitioning did not affect the arbiter training sets sizes significantly and are not shown here. However,

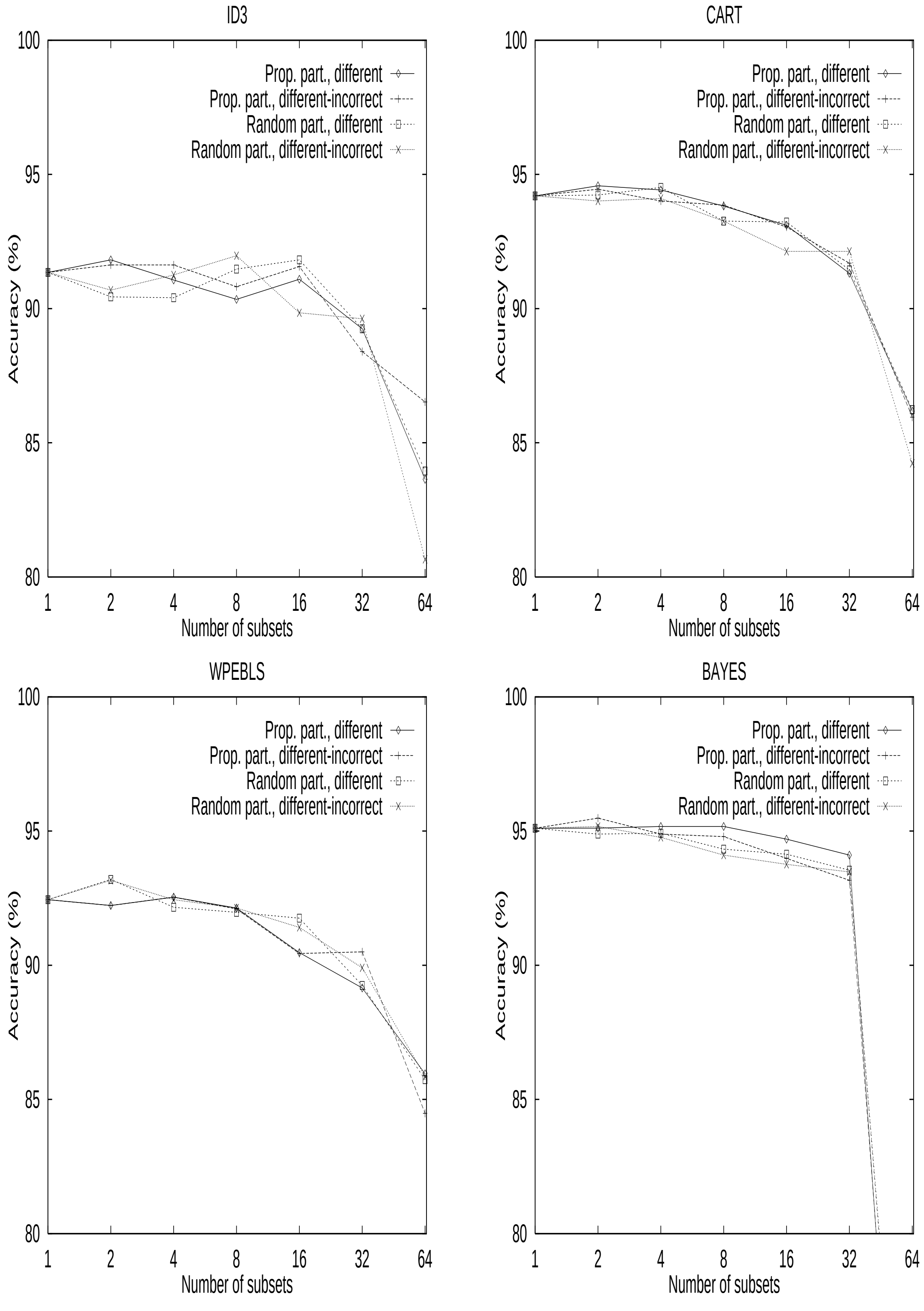


Figure 6.6: Accuracy with different class partitioning schemes.

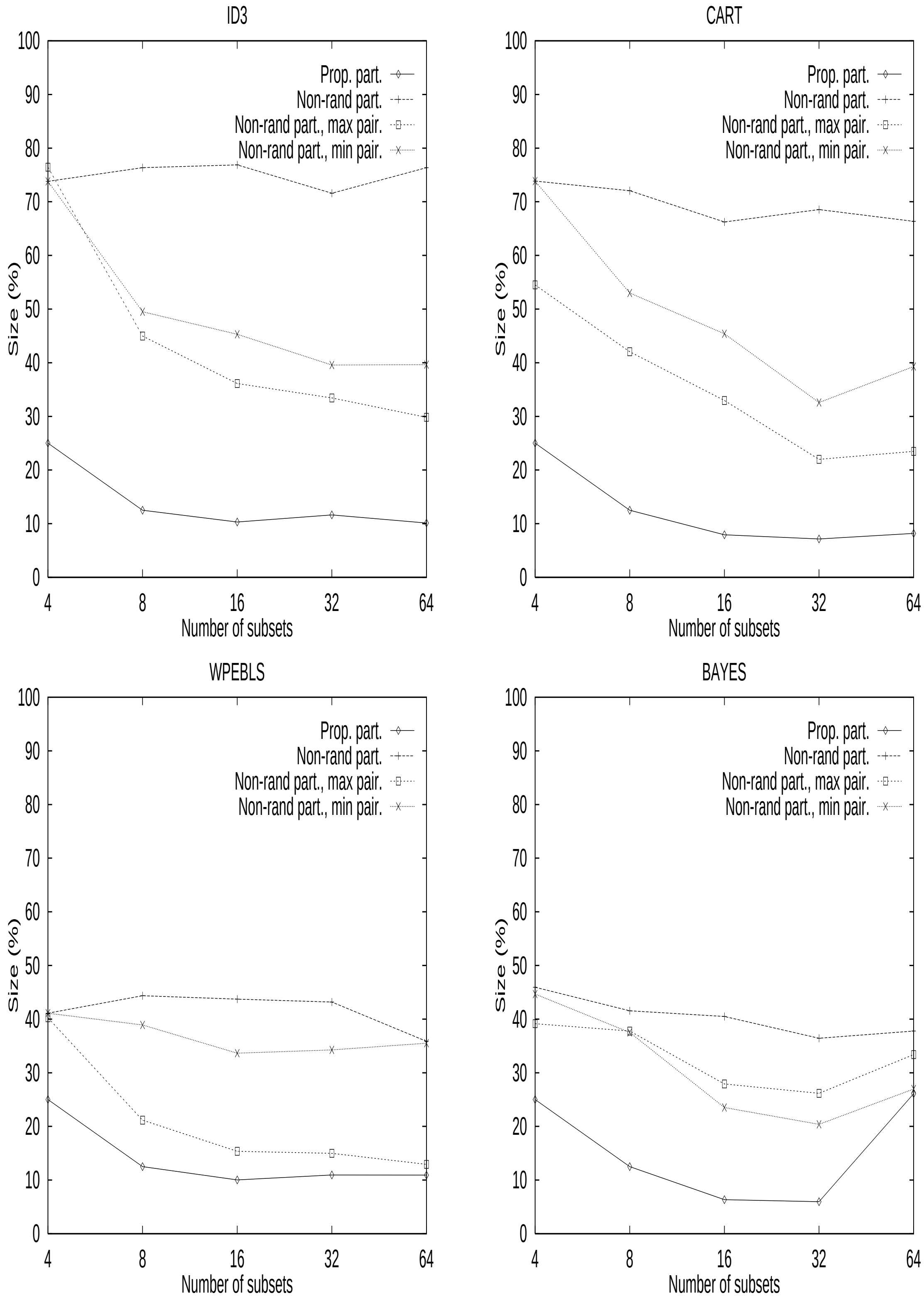


Figure 6.7: Arbiter training set size with different class partitioning and pairing strategies.

as shown in Figure 6.7, with non-random partitioning, both max-size and min-size pairing scheme significantly reduce the training set sizes in our experiments. Between the two schemes, max-size pairing empirically exhibited greater reduction in set sizes than min-size pairing. The largest arbiter training set sizes were around 10% of the original data when the number of subsets was larger than eight (except for BAYES with 64 subsets). (BAYES seemed to be not able to gather enough statistics on small subsets, which can also be observed from results presented earlier). Note that when the number of subsets is eight or fewer, the training sets for the leaf classifiers are larger than 10% of the original data set and become the largest in the arbiter tree. As mentioned before, the two pairing schemes did not affect the sizes of the arbiter training sets for the proportional partitioning. One possible explanation is that the proportional partitioning scheme produced the smallest training sets possible and the pairing schemes did not matter. In summary, proportional class partitioning tends to produce the smallest training sets and the *max-size* pairing scheme can reduce the set sizes in partitioning schemes that do not maintain the proportional partitioning of classes.

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

## 6.5 Experimental Results for Combiner Tree

Here we consider the accuracy of combiner trees. In our experiments, we varied the number of equi-sized subsets of training data from 2 to 64 ensuring each was disjoint but with proportional distribution of examples of each class. We also varied the

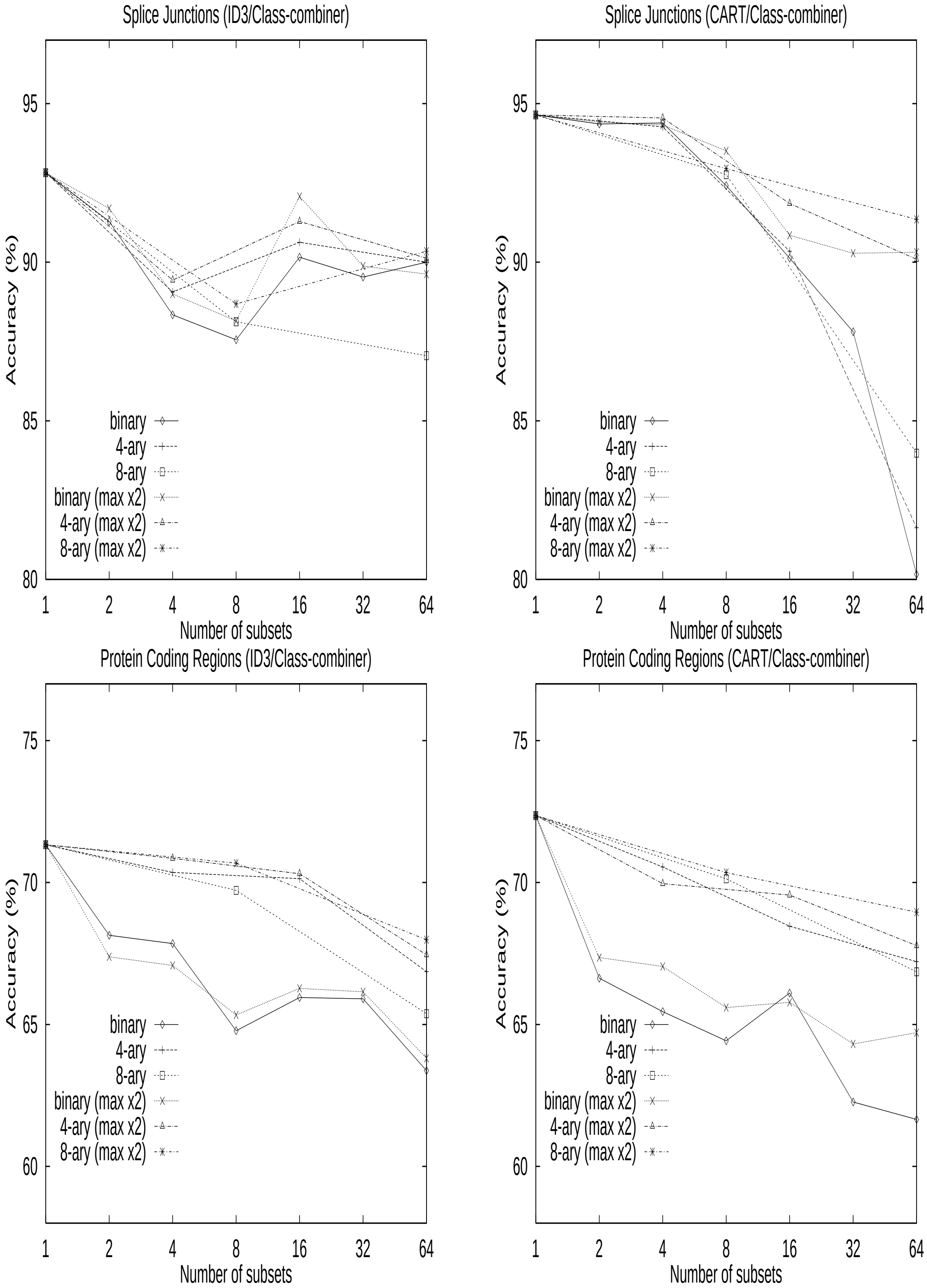


Figure 6.8: Accuracy for the *class-combiner* tree techniques.

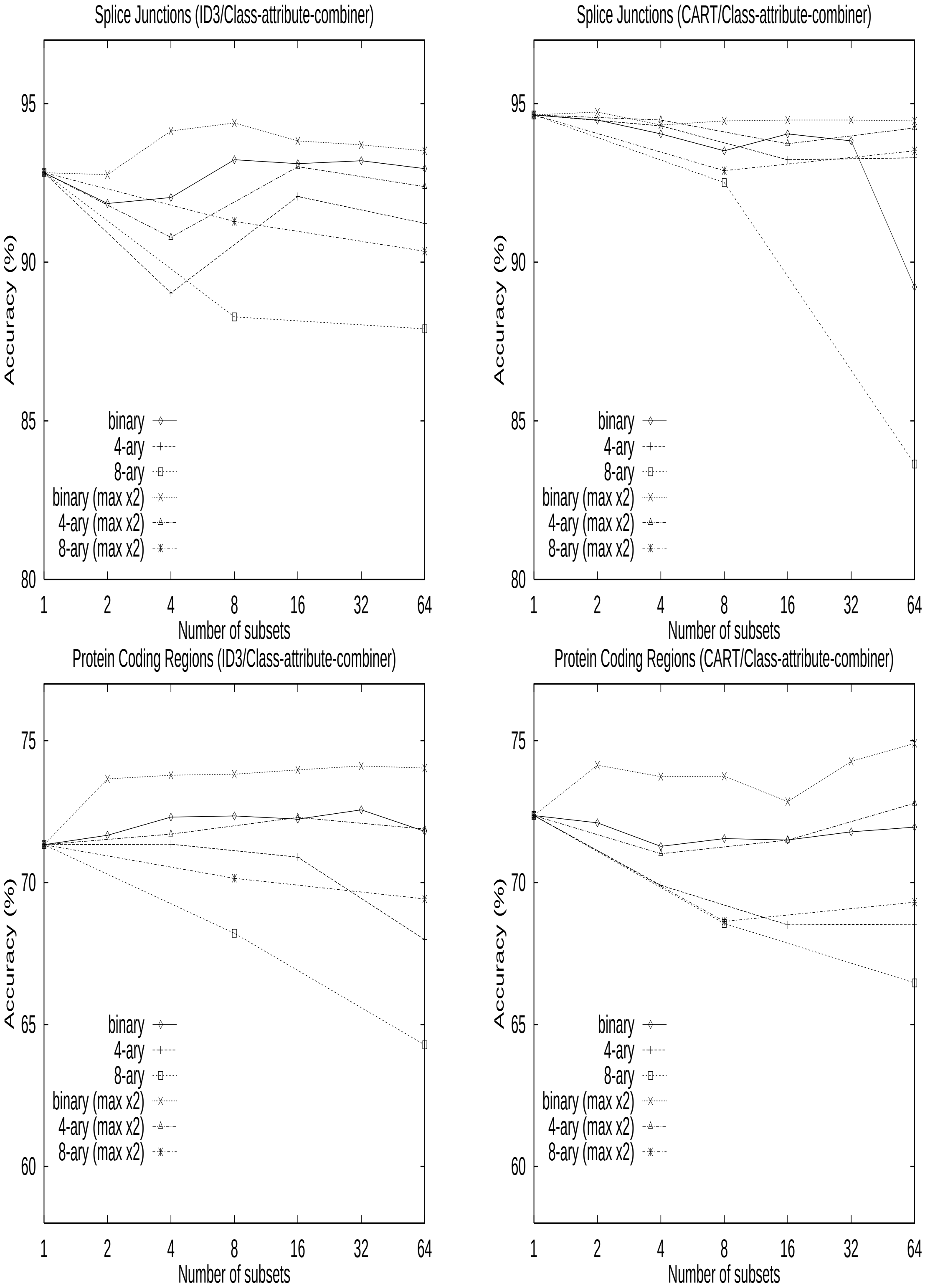


Figure 6.9: Accuracy for the *class-attribute-combiner* tree techniques.

order of the combiner trees from two to eight. Different combinations of two learning algorithms (ID3 and CART) and two learning tasks (Splice Junctions and Protein Coding Regions) were employed. The results of our experiments on the combiner trees (under two different training strategies) are displayed in Figure 6.8 and 6.9. The baseline accuracy for comparative evaluation is plotted as "one subset," meaning the learning algorithms were applied to the entire training set in-toto to produce the global classifier. The plots are derived from the average of 10-fold cross-validation runs.

Results from the *class-combiner* tree strategy displayed in Figure 6.8 show a drop in accuracy in both data sets in most cases, compared to the global classifier, when the number of subsets increases. The drop varies from 3% to 15%. (The percentage decrease in the amount of data in each training subset is far larger!) The binary combiner trees are less accurate than higher order trees in this case. This might be due to the lack of information for finding correlations among only two sets of predictions. As in the experiments for arbiter trees, we doubled the size of meta-level training sets. Statistically significant improvements were observed in the SJ data set with CART as the learner.

In another experiment using the *class-attribute-combiner* tree strategy, Figure 6.9 suggests that the binary trees appear to maintain the accuracy of the global classifier except in the splice junctions data set with CART as the learner. Higher-order trees were generally less accurate.

We note with interest that doubling the size of the training sets for combiners improved accuracy significantly. For the protein coding regions data set, the accuracy of the binary trees was consistently higher than that from the global classifier; i.e., *this meta-learning strategy has demonstrated a means of boosting accuracy of a single classifier* trained on the entire data set. The improvement is statistically significant. This is a particularly interesting finding since the information loss due to data partitioning was more than recovered by the combiner tree. Thus, this scheme demonstrates a means of integrating the collective knowledge distributed among the individual base classifiers.

## 6.6 Summary

We detailed two hierarchical meta-learning strategies: *arbiter tree* and *combiner tree*. Empirical results from bounded arbiter training sets indicate that the strategies are viable in speeding up learning algorithms with a small degradation in prediction accuracy. In addition, the algorithms can scale to arbitrarily large problems by setting the size limit of distinct training data subsets to the memory capacity of an individual processor and increasing the number of processors. When the arbiter training sets are unbounded, the strategies can preserve prediction accuracy with less training time and required memory than the serial version. Schemes for reducing the size of arbiter training sets were also discussed. In particular, proportional partitioning of classes in the training subsets and a particular classifier pairing schemes have been empirically observed to reduce the size of arbiter training sets.

Also, the *class-combiner* tree scheme does not perform as well in maintaining or boosting accuracy as the *arbiter* or *class-attribute-combiner* tree scheme. 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, in the previous chapter, compared to the *hierarchical* (tree) strategies 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.

The reduced memory requirement and usage of multiple processors make our strategies scalable to much larger problems, which will inevitably arise from the Human Genome Project and many other efforts. Moreover, without the benefit of multiple processors, our strategies can still be used to handle problems larger than possible on a single processor. Thus, by using meta-learning techniques, main-memory based learning algorithms can scale to larger problems with or without the usage of multiple processors.

# Chapter 7

# Local Meta-Learning with Imported Remote Classifiers

Frequently, local databases represent only a partial view of the all the data available. For example, in detecting credit card fraud, a bank has information on its credit card transactions, from which it can learn fraud patterns. However, the patterns learned usually don't reflect all the fraud patterns found in transactions at other banks. That is, a bank might not know a fraud pattern that is prevalent at other banks.

One approach to solving this problem is to merge transactions from all databases into one database and locate all the fraud patterns. It is not uncommon that a bank has millions of credit card transactions; pooling transactions from all banks will create a database of astronomical dimension. Learning fraud patterns from millions of transactions already creates efficiency problems, processing transactions from all banks will probably be infeasible. In addition, transactions at one bank are usually proprietary because sharing them with other banks means giving away valuable customer purchasing information, which can be used to generate future profits. Exchanging transactions might also violate customers' privacy.

Another solution is to share the fraud patterns instead of the transaction data.

This approach benefits from a significant reduction of information needed to be merged and processed. Also, proprietary customer transaction information need not be shared. You might now ask that if the data are proprietary, the fraud patterns can also be proprietary. If the patterns are encoded in programs, the executables can be treated as "black boxes." That is, by sharing the black boxes, one doesn't have to worry about giving away valuable and proprietary information. The next question is how we can merge the black boxes.

In this chapter we explore the use of meta-learning in improving the accuracy performance of local learned models by merging them with ones imported from remote sites (Chan & Stolfo, 1996b). That is, at each site, learned models from other sites are also available. Furthermore, we investigate the effects on local accuracy when the local underlying training data overlap with those at remote sites. This situation arises in reality because, for example, the same person might be a customer at several banks and/or the same person can commit the same credit card fraud at different banks. We next discuss how meta-learning can improve local learning. Sections 7.2 and 7.3 evaluate local meta-learning and the effect of data replication.

## 7.1 Local Meta-Learning

In previous chapters we assume a certain degree of "raw data" sharing. As we discussed earlier, situations might arise when data sharing is not feasible, but sharing of "black-box" learned models is. In this scenario a local site can "import" classifiers learned at remote sites and use them to improve local learning. The problem we face is how we can take advantage of the imported "black-box" classifiers. Our approach is to treat it as an integration problem and use meta-learning techniques to integrate the classifiers.

Since only the local dataset, called $T_i$ at site $i$, is available at a site, we are limited to that dataset for meta-learning. A classifier, $C_i$, is trained from $T_i$ locally

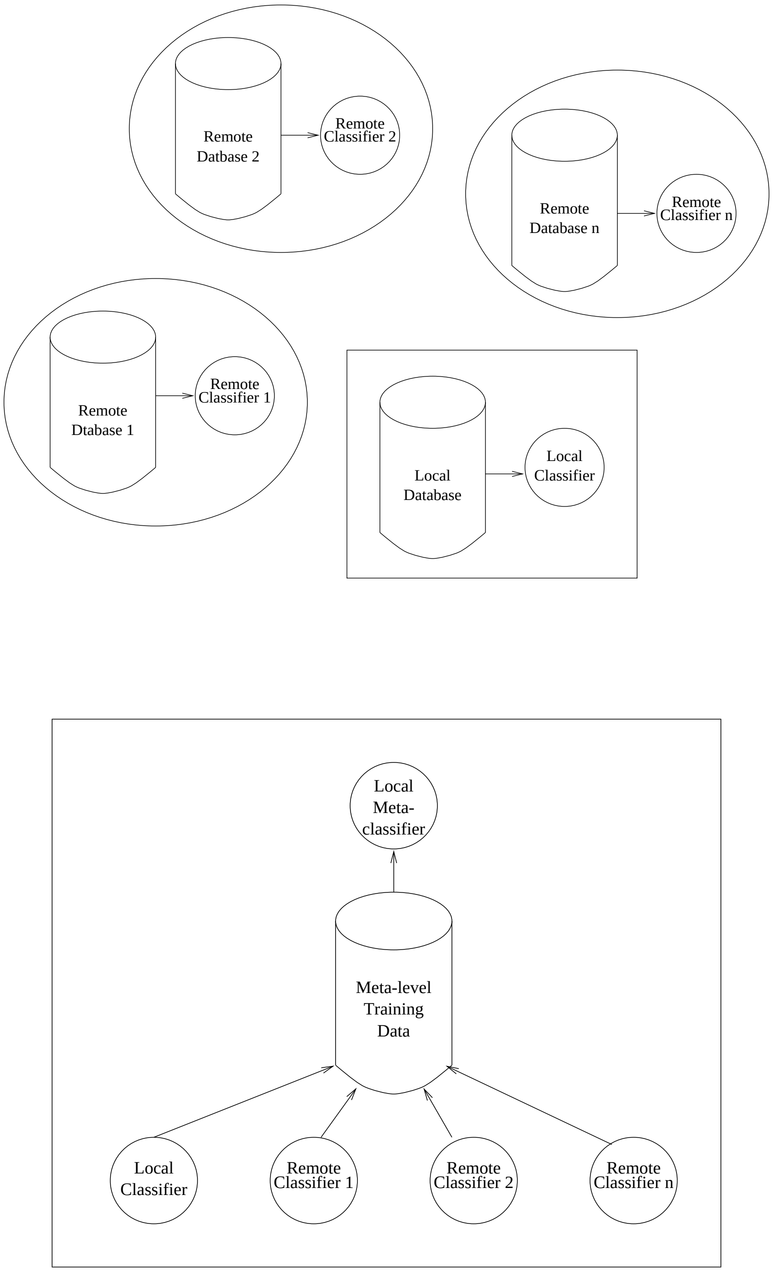


Figure 7.1: Local meta-learning at a site with three remote sites.

and a classifier, $C_j$ where $j \neq i$, is imported from each site $j$. Using $T_i$, each $C_j$ then generates predictions $P_{ij}$ and $C_i$ produces $P_{ii}$. $P_{ij}$ and $P_{ii}$ form the meta-level training set according to the strategies described in Chapter 3. That is, the local and remote classifiers are treated as base classifiers in our previous discussion. Once the meta-level training set is created, the corresponding meta-classifier is learned. Figure 7.1

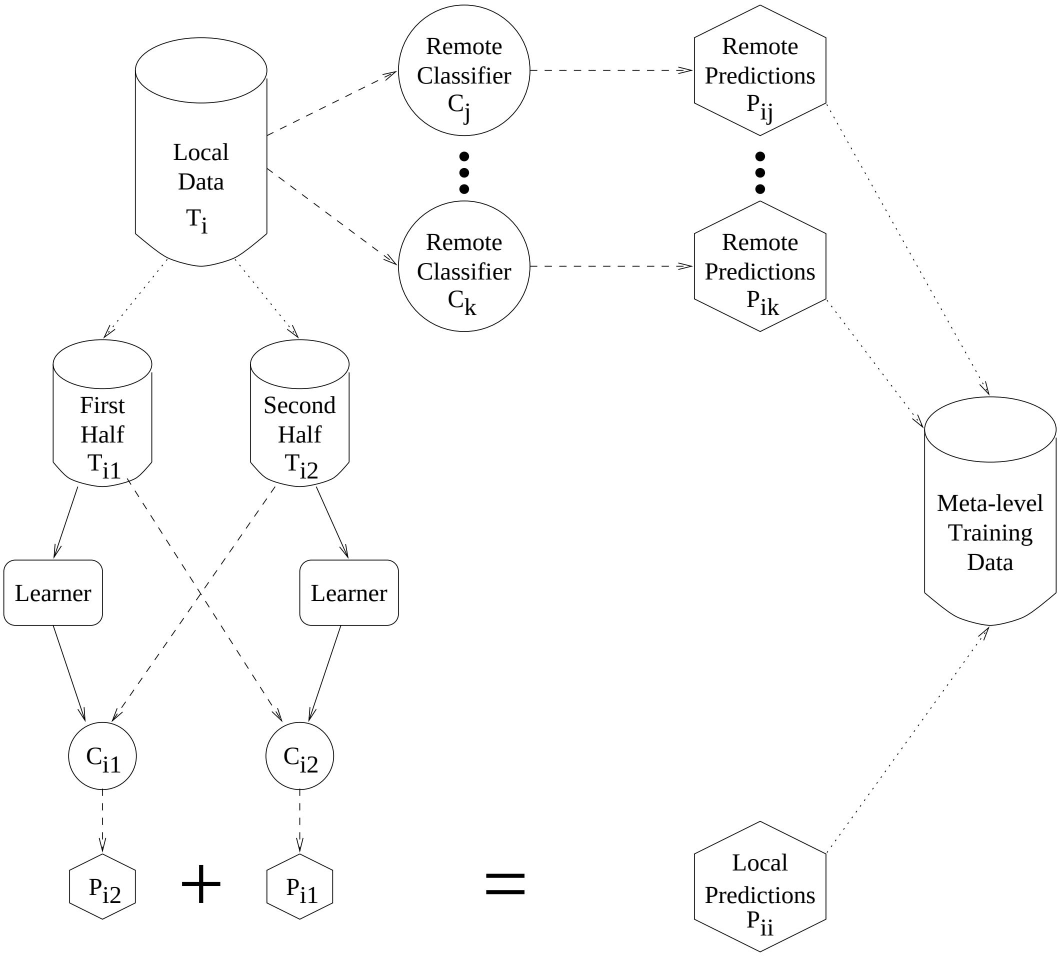


Figure 7.2: Generating local meta-level training data.

shows the relationship among various classifiers and sites during local meta-learning.

However, the predictions $P_{ii}$ of the local classifier $C_i$ on the local training set $T_i$ will be more correct than the predictions, $P_{ij}$, generated by the remote classifiers because $C_i$ was trained from $T_i$. As a result, during meta-learning, the trained meta-classifier will heavily bias toward the local classifier since the local classifier predicts much more accurately than the remote classifiers (recall that the remote classifiers were not trained on the local dataset $T_i$). For example, a local nearest-neighbor classifier can predict the local training set perfectly and the meta-learner will ignore all the remote classifiers. That is, we can't use the remote classifiers to improve local learning, which defeats the purpose of importing the remote classifiers initially.

To resolve this situation, at the local site, we partition $T_i$ into two sets, $T_{i1}$ and $T_{i2}$, from which classifiers $C_{i1}$ and $C_{i2}$ are trained. $C_{i1}$ then predicts on $T_{i2}$ and $C_{i2}$ on $T_{i1}$. The union of the two sets of predictions form the predictions for the local classifier ($P_{ii}$). This method, called 2-fold *cross-validation partitioning*, tries to approximate the behavior of $C_i$ on unseen data. The process of obtaining the predictions $P_{ij}$ from the remote classifiers remains unchanged. Figure 7.1 depicts this process of generating local meta-level training data. Now, during meta-learning, remote classifiers will not be automatically ignored since the local classifier is also judged on "unseen" data. The next section discusses our experimental evaluation of the local meta-learning approach.

## 7.2 Experimental Results

Different combinations of four inductive learning algorithms (ID3, CART, BAYES, and CN2) and four data sets (Splice Junctions, Protein Coding Regions, Protein Secondary Structures, and Artificial) were used in this set of experiments (as described in Section 4.3). To simulate the multiple-site scenario, we divided the training set into equi-sized subsets (each subset representing a site) and varied the number of subsets (sites) from 2 to 64. We also ensured that each subset 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). The *arbiter*, *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 above four data sets, each with the above four learning algorithms and the results are plotted in Figures 7.3 through 7.6. The plotted accuracy is the average accuracy of local meta-classifiers over 10-fold cross-validation runs. In each run, $m$ sites generate $m$ local classifiers and $m$ local meta-classifiers. In the figures, *avg-base* denotes the average accuracy of the local/base classifiers, which is our base line. Statistical significance was measured by using the one-sided t-test with a 90% confidence value.

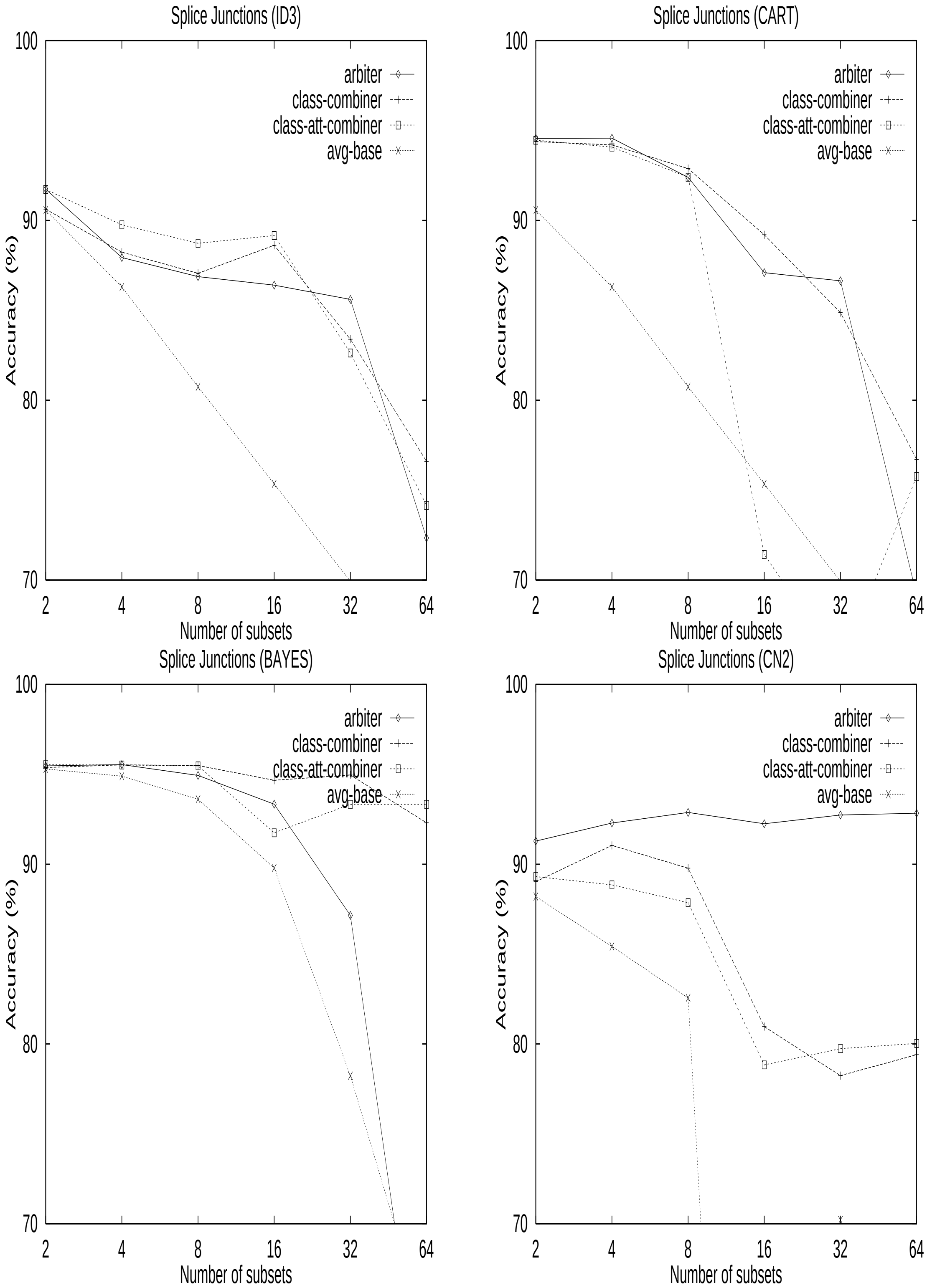


Figure 7.3: Accuracy for local meta-learning vs. number of subsets in the splice junction domain.

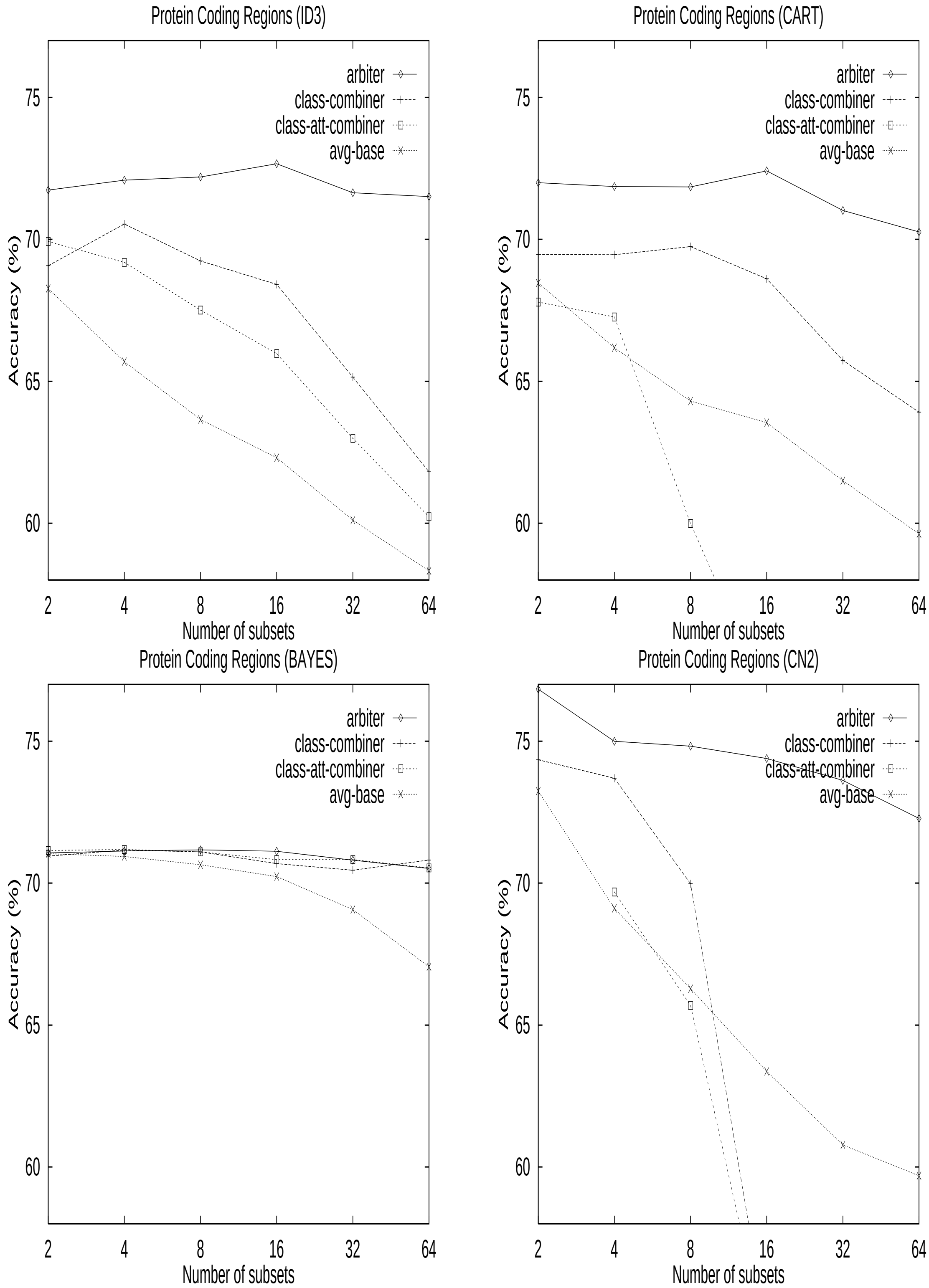


Figure 7.4: Accuracy for local meta-learning vs. number of subsets in the protein coding region domain.

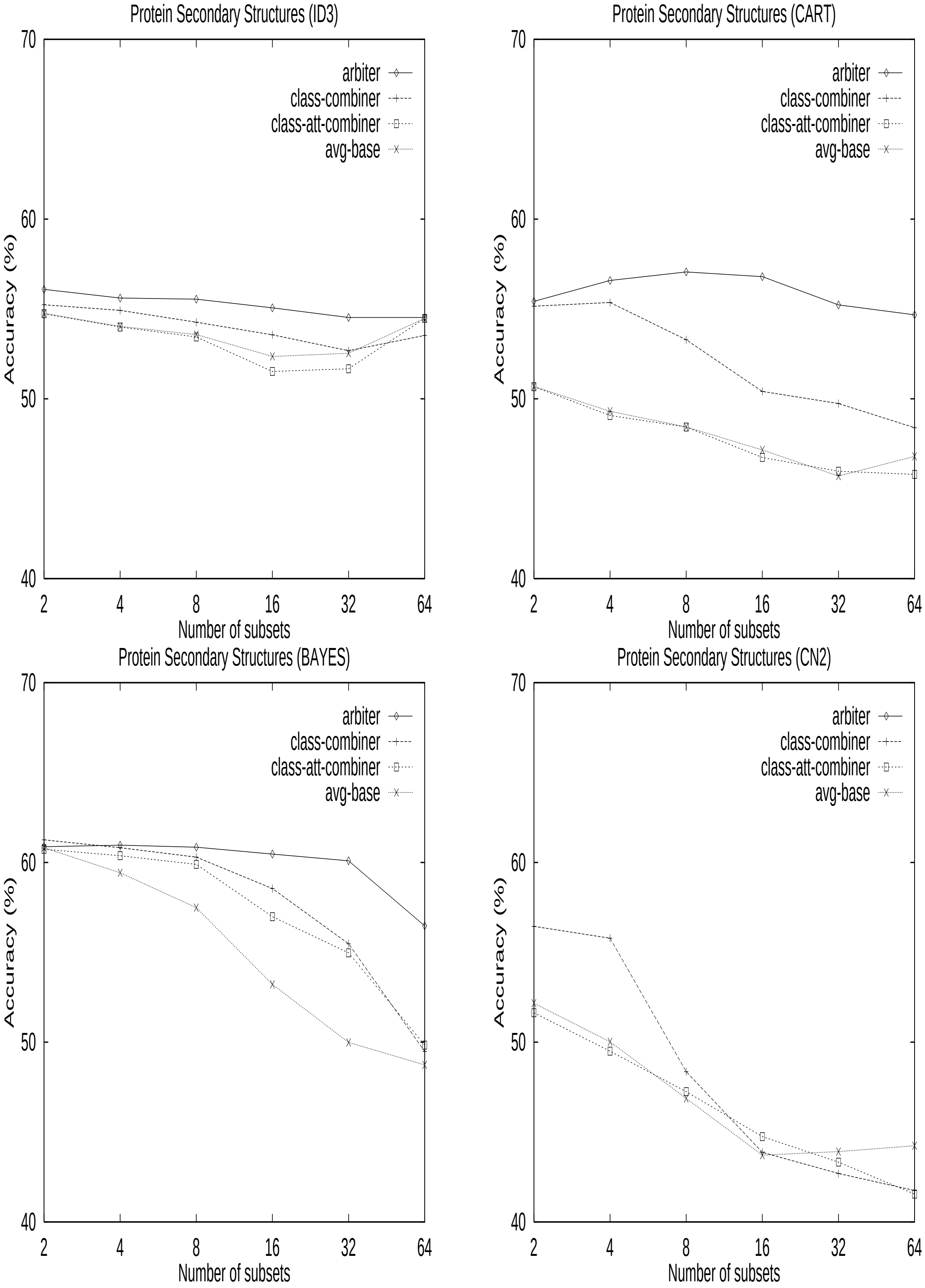


Figure 7.5: Accuracy for local meta-learning vs. number of subsets in the secondary structure domain.

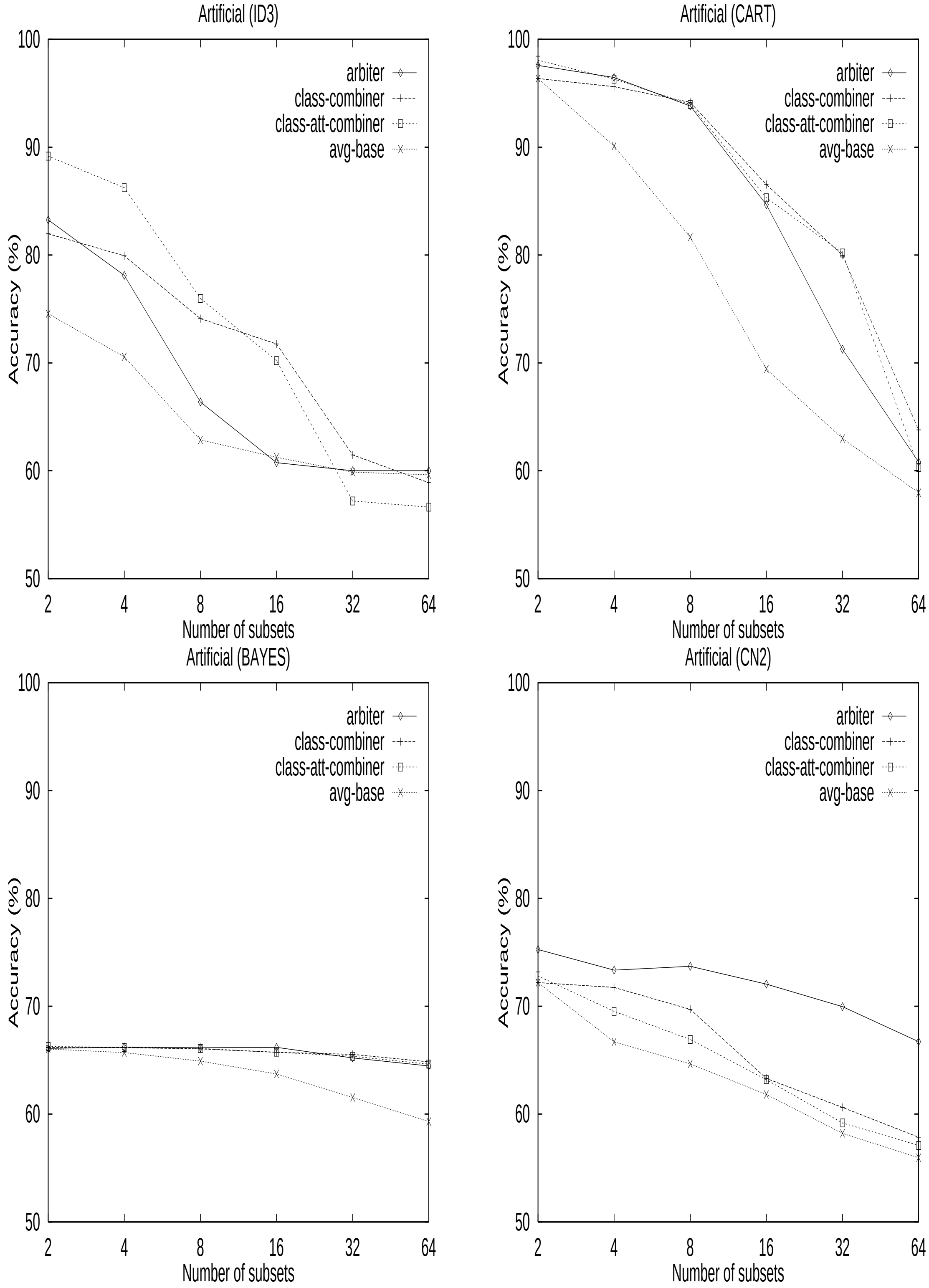


Figure 7.6: Accuracy for local meta-learning vs. number of subsets in the artificial domain.

When compared to the base accuracy, at least one of three local meta-learning strategies yields significantly higher accuracy in 13 out of the 16 cases (mostly at 4 or more subsets). Local meta-learning still has higher accuracy (not significantly) in 2 of the 3 remaining cases. Larger improvement usually occurs when the size of the local dataset is smaller (the number of subsets/sites are larger). In many cases the arbiter scheme improves accuracy more than the two combiner strategies.

While many of the base classifiers drop in accuracy when the dataset size gets smaller, some of the meta-learning strategies roughly maintain the same level of accuracy. One apparent example is the arbiter scheme using ID3 as the learner in the Coding Regions dataset (Figure 7.4). The arbiter scheme stays above 70% accuracy while the base accuracy drops to below 60%. The arbiter scheme maintains the accuracy in 8 out of 16 cases. For the Coding Regions dataset, the arbiter scheme improves local learning by a wide margin when the learners are ID3, CART, and CN2 (3 of the 4 learners).

The results obtained here are consistent with those from non-local meta-learning in previous chapters, where raw data can be shared among sites. Meta-learning improves accuracy in a distributed environment and the arbiter scheme is more effective than the two combiner techniques. Next, we investigate the effects on accuracy of local meta-learning when different sites possess some degree of common data.

## 7.3 Experimental Results on Data Replication

As we discussed previously, different sites might have some overlapping data. To simulate this phenomenon, we allow some amount of replication in each partition of data. We prepare each learning task by generating subsets of training data for the local/base classifiers according to the same generative scheme in Section 5.2.2.

In the experiments reported here, $\Delta$ ranged from 0% to 40%, with 10% increments. Each set of incremental experimental runs, however, chooses an entirely new

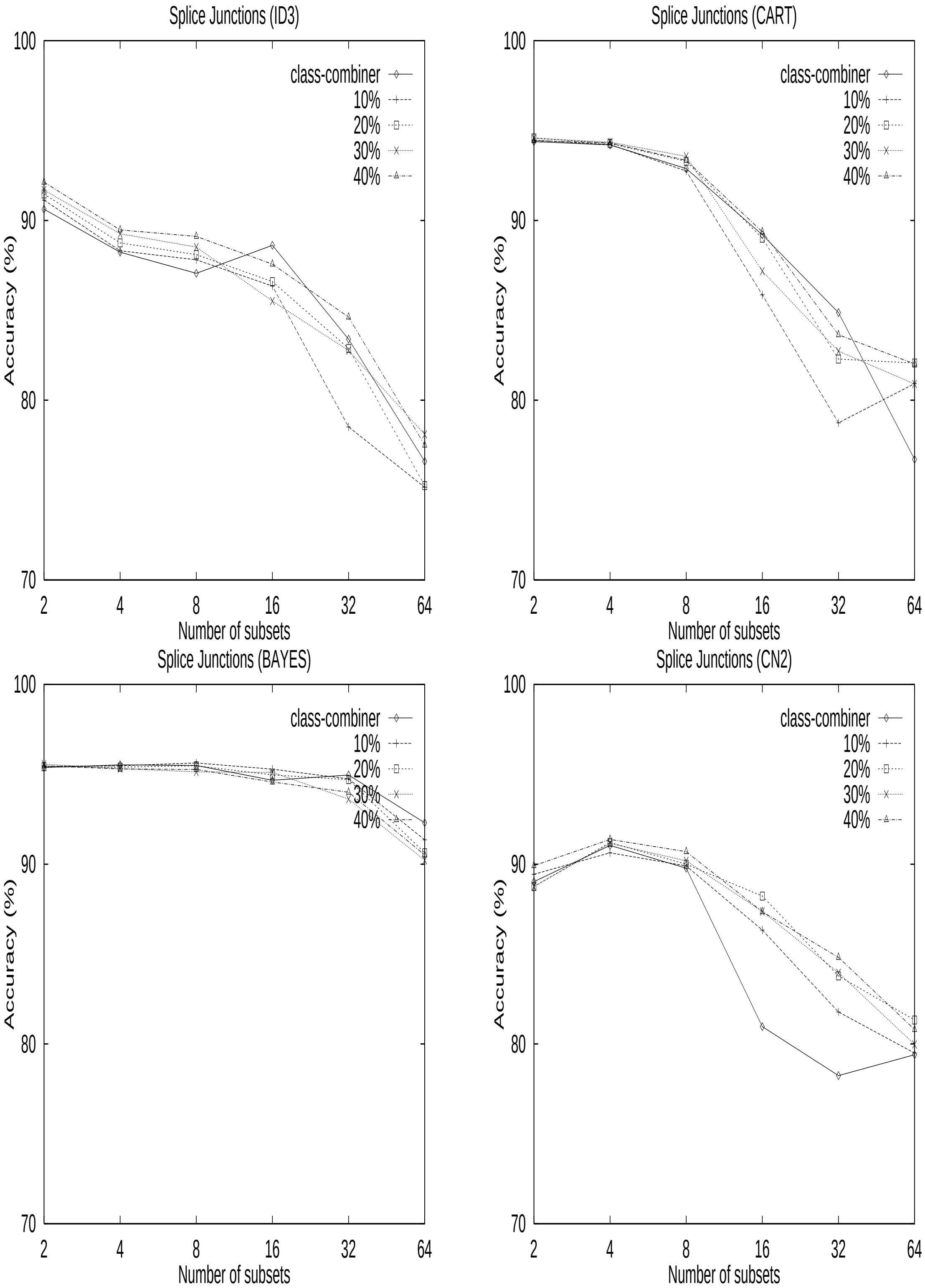


Figure 7.7: Accuracy for the *class-combiner* scheme trained over varying amounts of replicated splice junction data. $\Delta$ ranges from 0% to 40%.

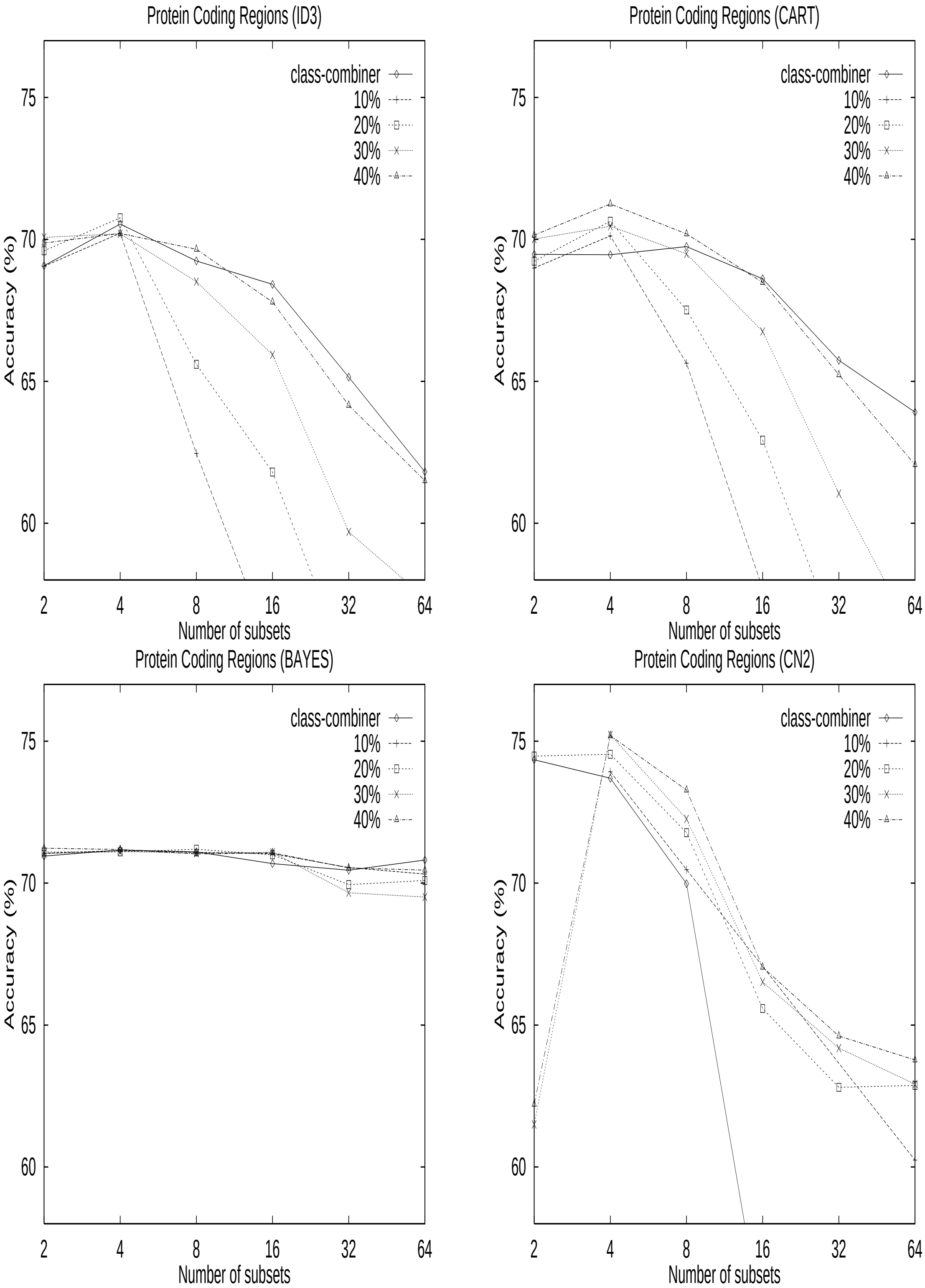


Figure 7.8: Accuracy for the *class-combiner* scheme trained over varying amounts of replicated protein coding region data. $\Delta$ ranges from 0% to 40%.

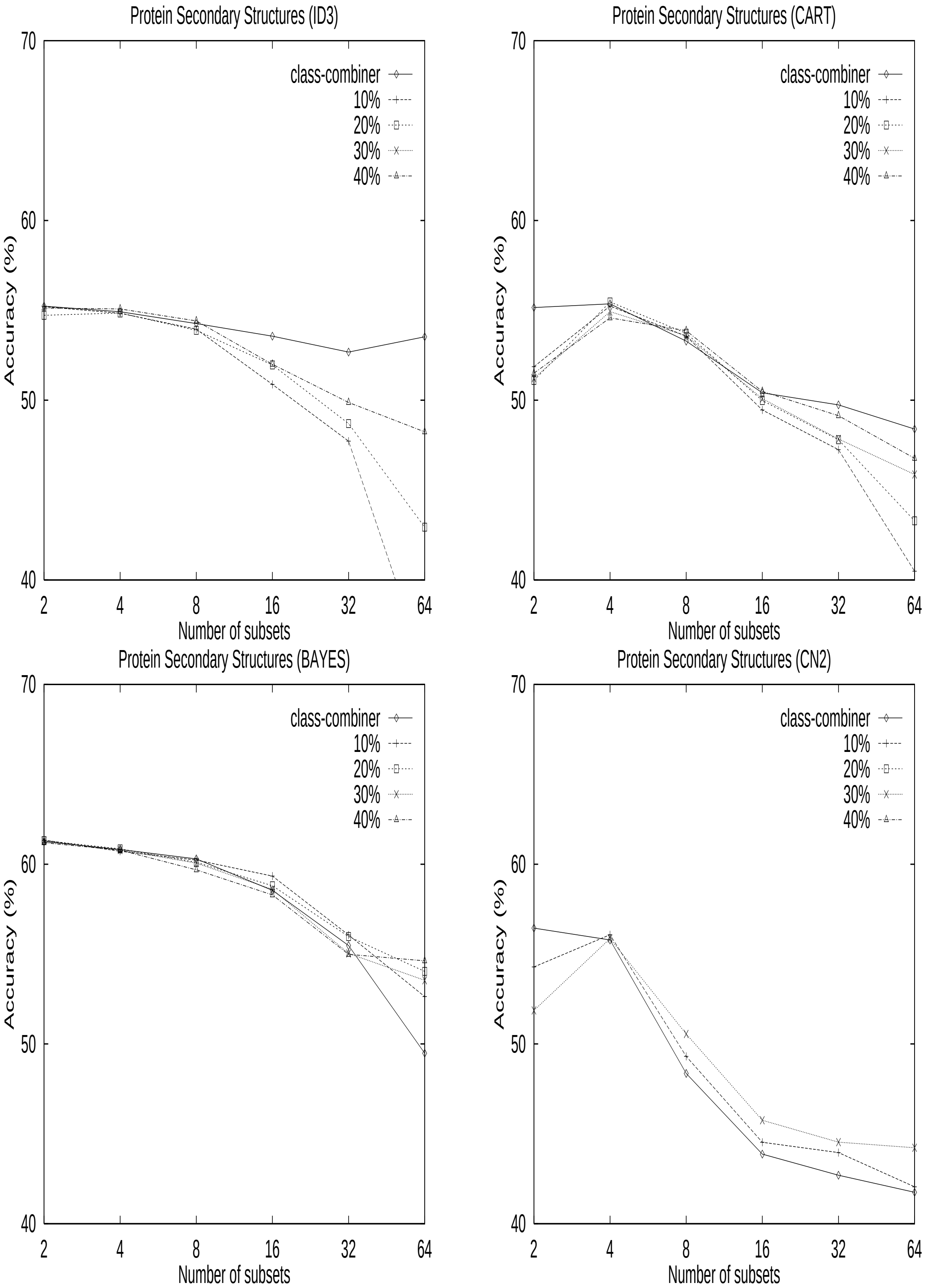


Figure 7.9: Accuracy for the *class-combiner* technique trained over varying amounts of replicated secondary structure data. $\Delta$ ranges from 0% to 40%.

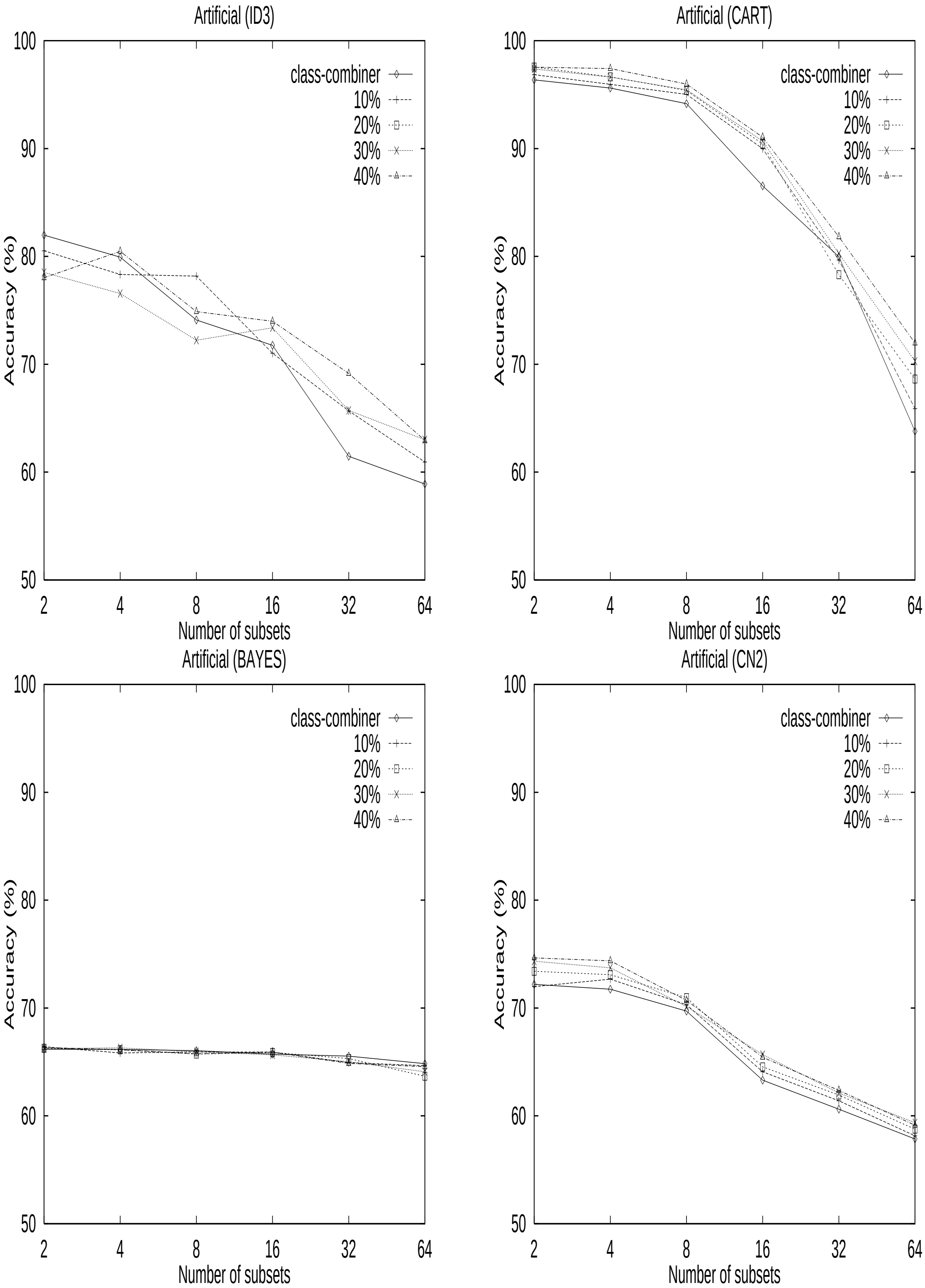


Figure 7.10: Accuracy for the *class-combiner* technique trained over varying amounts of replicated artificial data. $\Delta$ ranges from 0% to 40%.

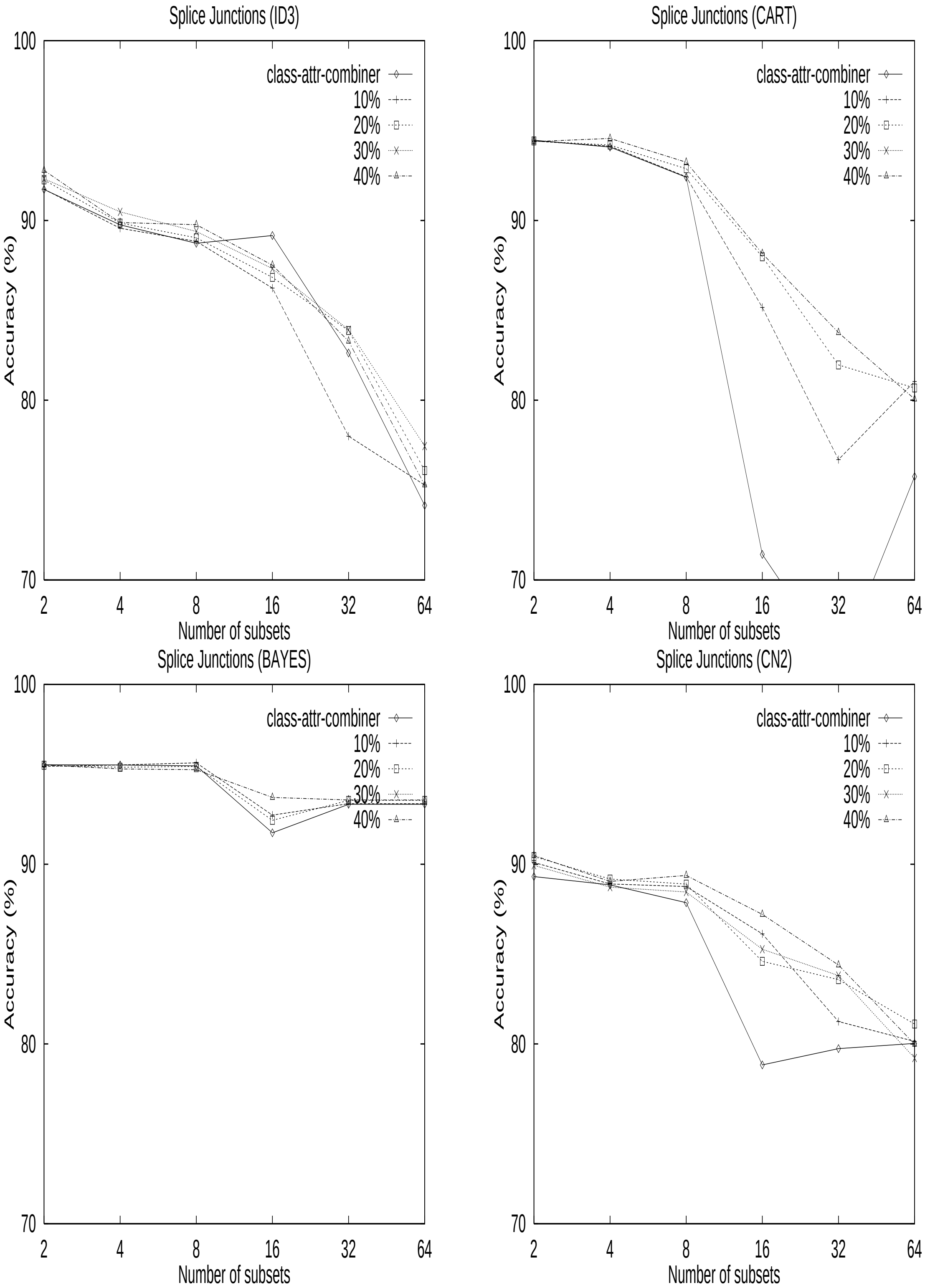


Figure 7.11: Accuracy for the *class-attribute-combiner* technique trained over varying amounts of replicated splice junction data. $\Delta$ ranges from 0% to 40%.

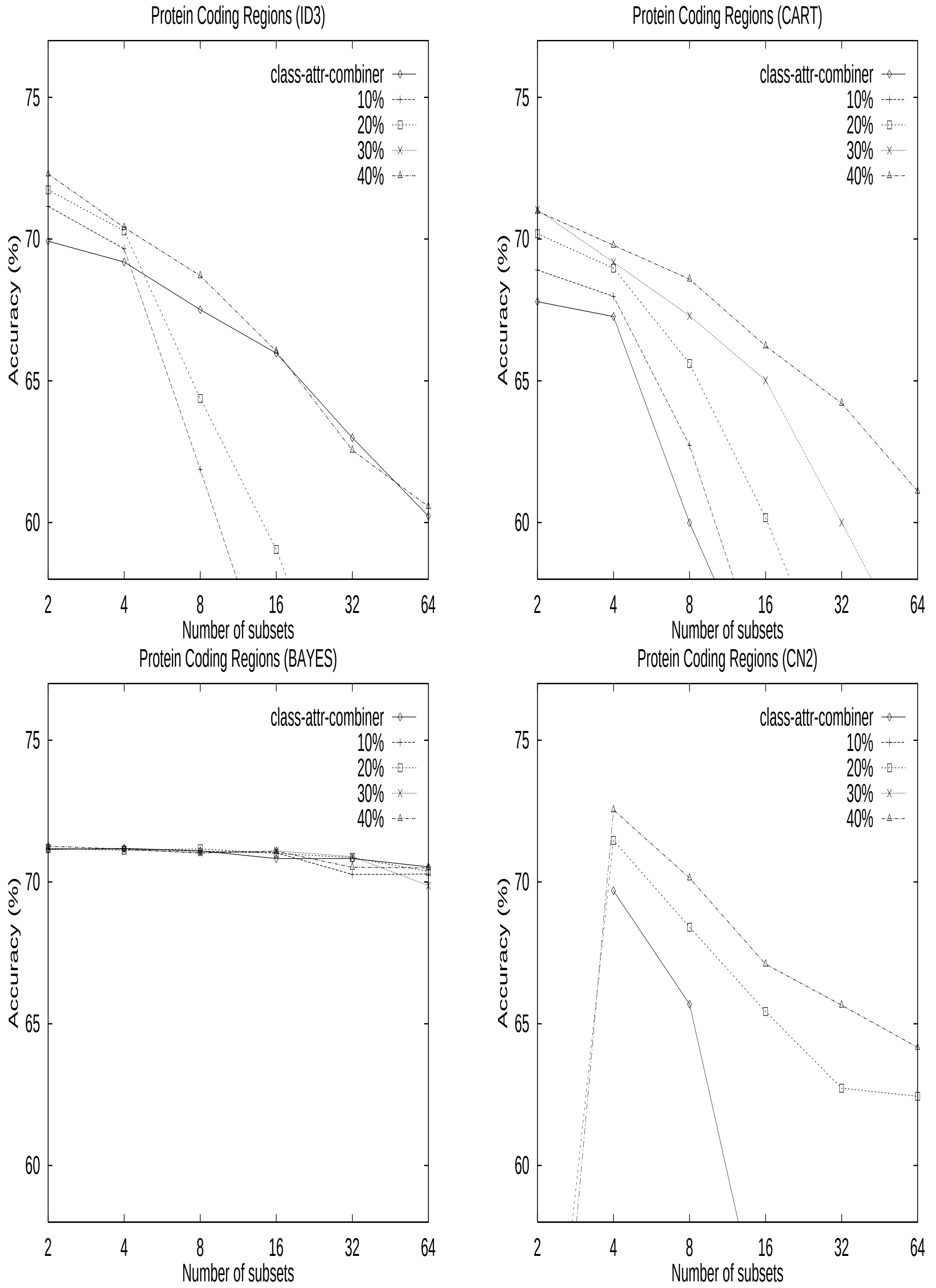


Figure 7.12: Accuracy for the *class-attribute-combiner* technique trained over varying amounts of replicated protein coding region data. $\Delta$ ranges from 0% to 40%.

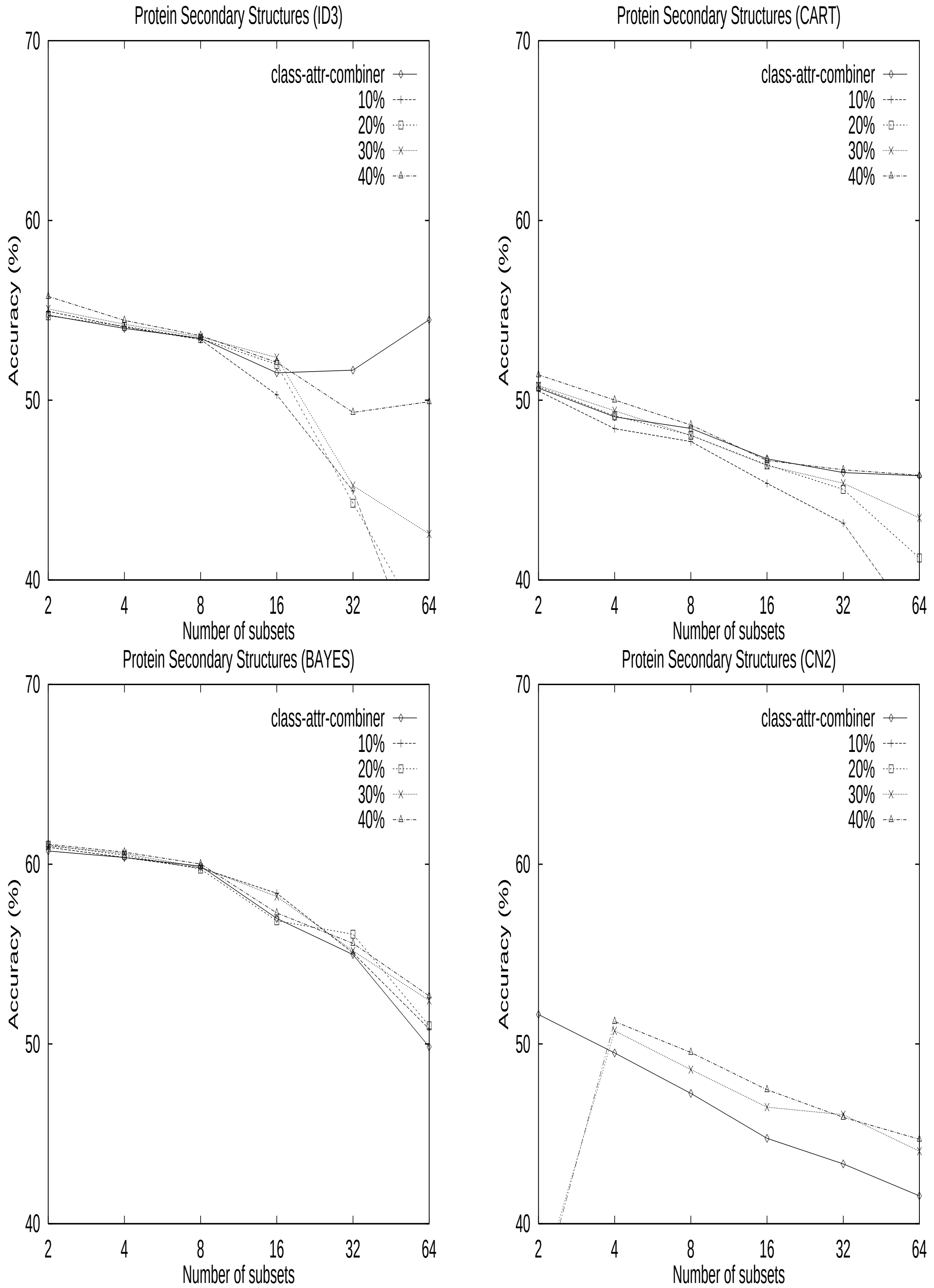


Figure 7.13: Accuracy for the *class-attribute-combiner* technique trained over varying amounts of replicated secondary structure data. $\Delta$ ranges from 0% to 40%.

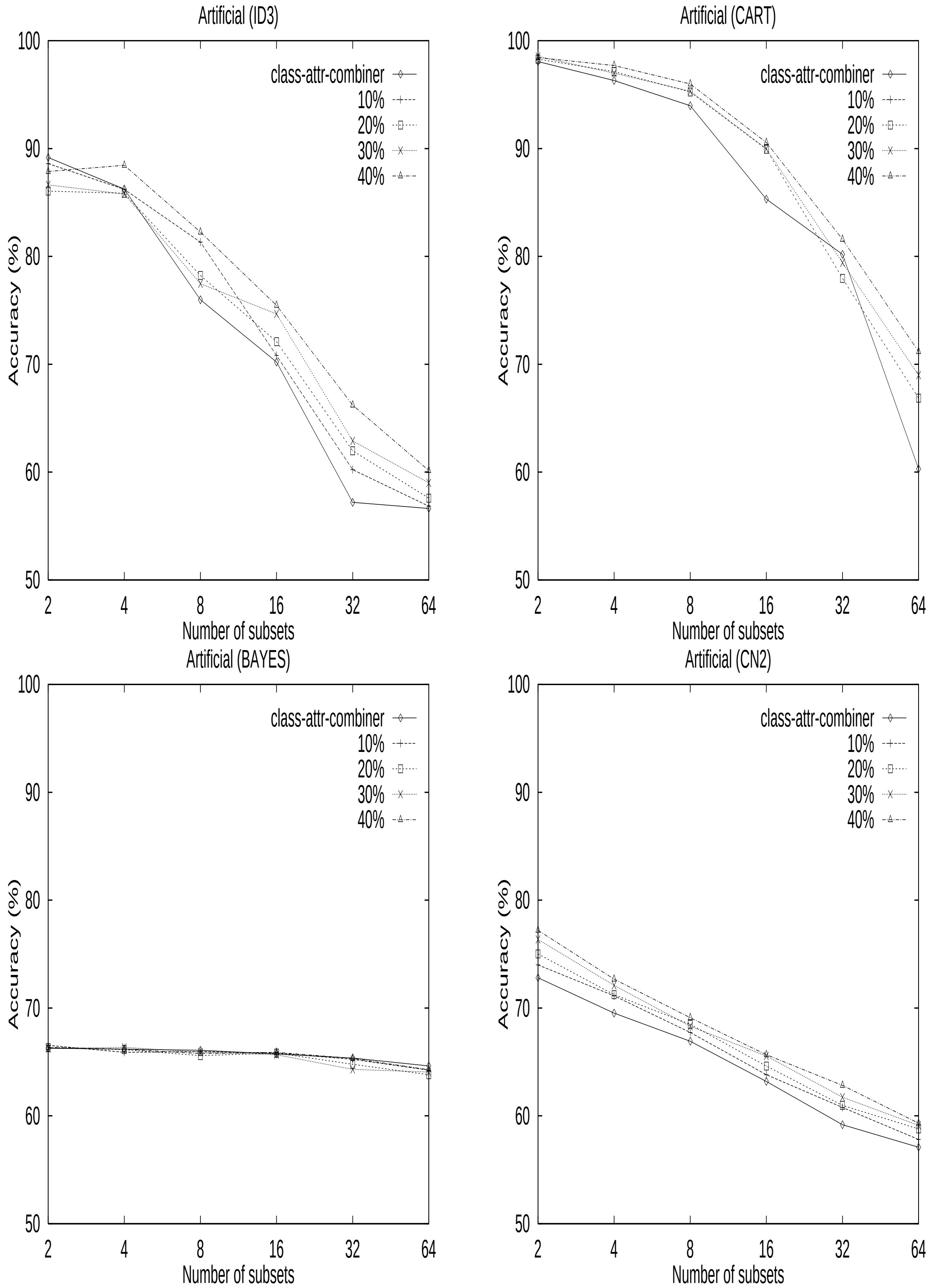


Figure 7.14: Accuracy for the *class-attribute-combiner* technique trained over varying amounts of replicated artificial data. $\Delta$ ranges from 0% to 40%.

distribution of replicated values. No attempt was made to maintain a prior distribution of training data when incrementing the amount of replication. This "shot gun" approach provides us with some sense of a "random learning problem" that we may be faced with in real world scenarios where replication of information is likely inevitable or purposefully orchestrated.

The same experimental setup was used as in the prior experiments. Results for the replicated data scenario using the class-combiner and class-attr-combiner strategies are plotted in Figures 7.7 through 7.14 (a total of 32 cases). 7 out of 32 cases show significant accuracy difference when the degree of replication increases; 6 of these 7 cases occur in the Coding Regions dataset. 20 out of 32 cases show no significant accuracy changes across all subset sizes and degrees of replication. The remaining 5 cases have some significant accuracy difference at certain subset sizes.

In summary, the majority doesn't show significant accuracy difference when the degree of replication increases. This is contrary to one's intuition since one would expect the accuracy to increase when the local sites have a higher percentage of all the available data combined. That implies that local meta-learning is quite effective in integrating models from remote sites without the help of replicated data. Our findings here are consistent with those from one-local meta-learning in Section 5.2.2.

## 7.4 Summary

We have presented techniques for improving local learning by integrating remote classifiers through local meta-learning. Our experimental results suggest local meta-learning techniques, especially the arbiter scheme, can significantly raise the accuracy of the local classifiers. Furthermore, results from our data replication experiments suggest local meta-learning can integrate local and remote classifiers effectively without having a larger share of global data at a local site.

# Chapter 8

# Analyzing the Integration of Multiple Learned Classifiers

In previous chapters we demonstrated the effectiveness of integrating multiple learned classifiers. In this chapter we define and apply analytical metrics to gain a deeper understanding of the effectiveness, which can then guide us to develop improvements for our methods.

## 8.1 Notations

To facilitate the formal definitions of metrics discussed in this chapter, we adhere to the following notations:

- $n$ = number of unseen instances used for evaluation
- $y_i$ = $i$-th instance
- $b$ = number of base classifiers
- $C_j$ = $j$-th base classifier

- $C_j(y_i)$ = classification of instance $y_i$ by base classifier $C_j$
- $\mathcal{OC}$ = overall classifier
- $\mathcal{OC}(y_i)$ = classification of $y_i$ by the overall classifier $\mathcal{OC}$
- $c$ = number of classes
- $class_k$ = $k$-th class
- $class(y_i)$ = correct classification of $y_i$
- $OneIfTrue(pred)$ = a function that returns one if predicate $pred$ is true and zero otherwise. That is,

$$OneIfTrue(pred) = \begin{cases} 1 & \text{if } pred \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

- $\alpha$ = overall prediction accuracy; formally,

$$\alpha = \frac{1}{n}\sum_{i}^{n} OneIfTrue(\mathcal{OC}(y_i) = class(y_i)) \tag{8.2}$$

- $\beta$ = average prediction accuracy of base classifiers; mathematically,

$$\beta = \frac{1}{b}\sum_{j}^{b}\frac{1}{n}\sum_{i}^{n} OneIfTrue(C_j(y_i) = class(y_i)) \tag{8.3}$$

We next investigate some of the metrics we developed to analyze the different characteristics in integrating multiple learned classifiers.

## 8.2 Metrics

Along with the definitions of metrics, empirical results using those metrics are presented and discussed. The results are based on the different permutations of 4 learning algorithms (ID3, CART, BAYES, and CN2) and 4 learning tasks (RNA Splice Junctions, Protein Coding Regions, Protein Secondary Structures, and Artificial). 4

integrating schemes (*class-combiner*, *class-attribute-combiner*, *arbiter*, and *weighted voting*) were used to merge base classifiers trained from 8 data subsets. We did not vary the number of data subsets in this evaluation because the variation generates vastly different base classifiers. The 3 meta-learning schemes were used in a one-level manner (Chapter 5), that is, not hierarchical. 10-fold cross validation runs were performed for each of the 64 permutations. Many of the figures below have four plots, one for each integrating schemes. Within each plot, results from the 16 permutations of learning algorithms and tasks are plotted. When a general trend is observed, a line is fitted to the 16 data points using the Marquardt-Levenberg algorithm (Ralston & Rabinowitz, 1978; Press *et al.*, 1988), a nonlinear least squares curve fitting mechanism, available in the GNUFIT (Grammes, 1993) package.

Before we discuss the different metrics used in this study. We first inspect how the average accuracy of base classifiers ($\beta$) affects the overall accuracy ($\alpha$). Their relationship is plotted in Figure 8.1. We observe that having highly accurate base classifiers is a definite contributing factor for achieving high overall accuracy.

### 8.2.1 Accuracy Difference and Improvement

In order to measure how well the integrating structures perform, we first define *accuracy difference* as the difference between the accuracy of the overall accuracy and the average accuracy of base classifiers. Formally,

$$accuracy\ difference = (\alpha - \beta) \times 100\% \tag{8.4}$$

Since different permutations of data sets and learning algorithms yield diverse levels of prediction accuracy, in Figure 8.2, we shows the wide range of accuracy achieved by the base classifiers among different permutations, and the resulting *accuracy difference*. Base classifiers learned from the secondary structure data set have an accuracy at around 45–60%, whereas those from the splice junction data set have a higher range from 80–95%.

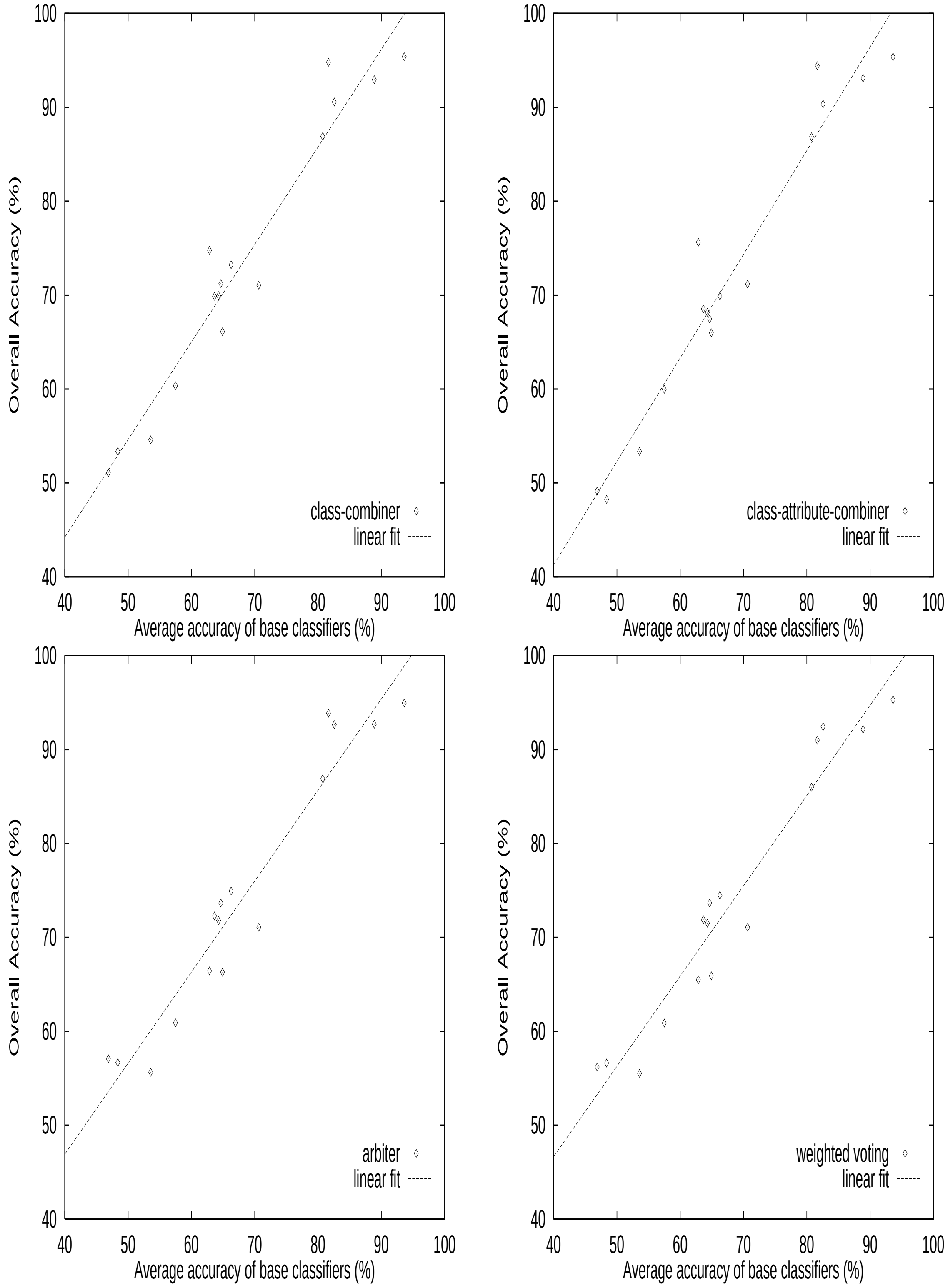


Figure 8.1: Average accuracy of base classifiers vs. overall accuracy.

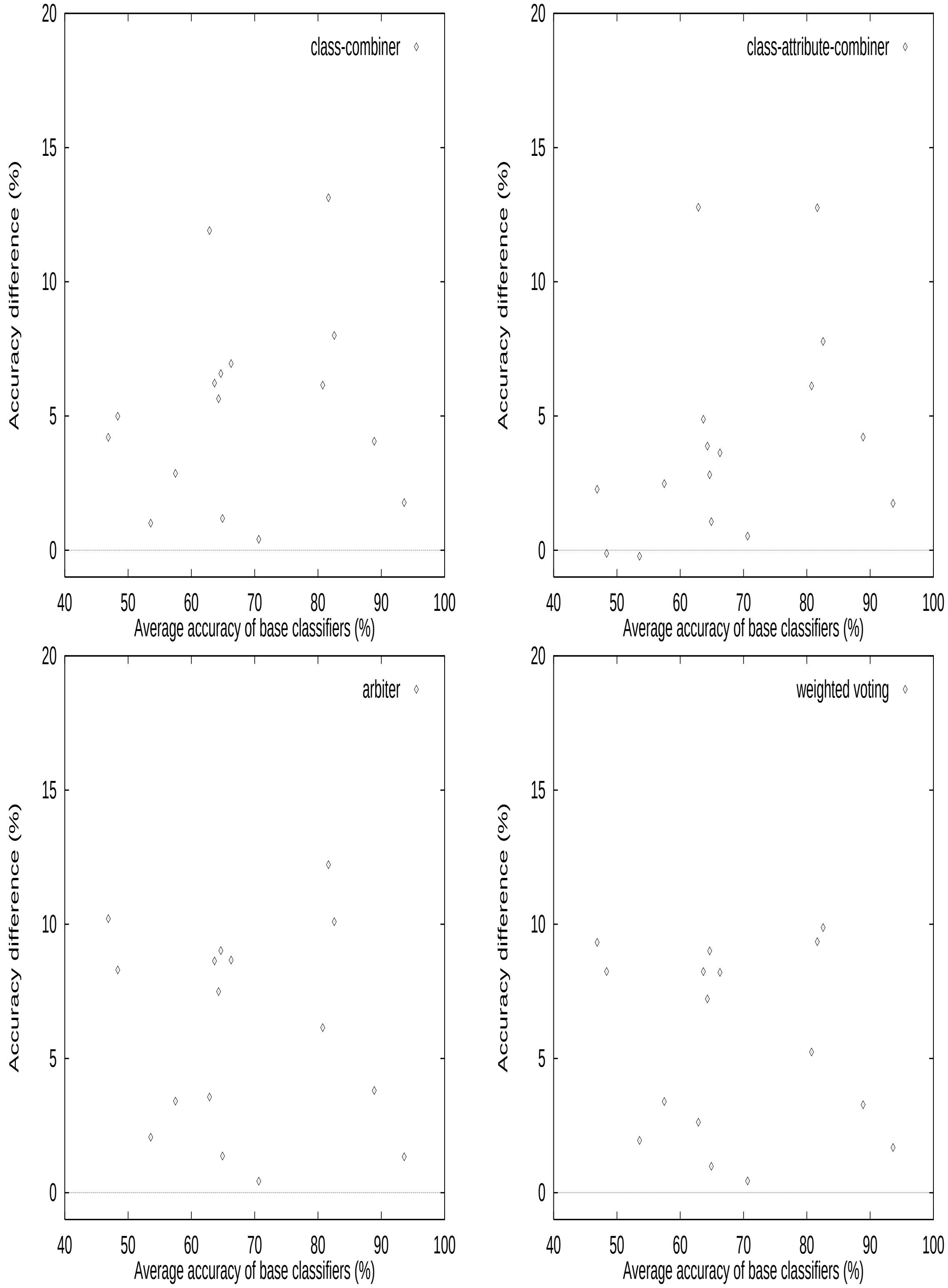


Figure 8.2: Average accuracy of base classifiers vs. accuracy difference.

Moreover, we observe that the data points are quite scattered, suggesting that the initial accuracy of base classifiers does not have much effect on the amount of improvement that can be achieved by the integrating structures. This might be due to two opposing arguments. One side of the coin suggests it is harder to gain accuracy from higher initial accuracy due to less room for improvement (imagine all the base classifiers have 100% accuracy). However, the other side of the coin suggests it is more difficult to gain accuracy from lower initial accuracy because of a weaker foundation to build upon (imagine all the base classifiers have 0% accuracy).

In an attempt to factor in the wide range of accuracy levels in the base classifiers, we define *relative accuracy difference* or *accuracy improvement* as:

$$accuracy\ improvement = \frac{accuracy\ difference}{\beta} = \frac{\alpha - \beta}{\beta} \times 100\% \qquad (8.5)$$

Figure 8.3 plots *accuracy improvement* against *average accuracy of base classifiers*.

Ali and Pazzani (1996) use *error ratio* to measure the performance of the overall classifier. Error ratio is defined as the ratio between overall error and error of the base classifiers. That is, in our notations,

$$error\ ratio = \frac{1 - \alpha}{1 - \beta}$$

They also mentioned the option of using *error difference* $((1 - \beta) - (1 - \alpha))$, which is the same as *accuracy difference* $(\alpha - \beta)$. They chose *error ratio* because they believe that "it becomes increasingly difficult to obtain reductions in error as the error of the single model approaches zero."

### 8.2.2 Diversity

In information theory (Abramson, 1963), given the probabilities of different events, entropy measures the average amount of information required to represent each event. For digital communication channels, amount of information is measured in bits. Entropy can also measure how random the different events can occur. The larger the

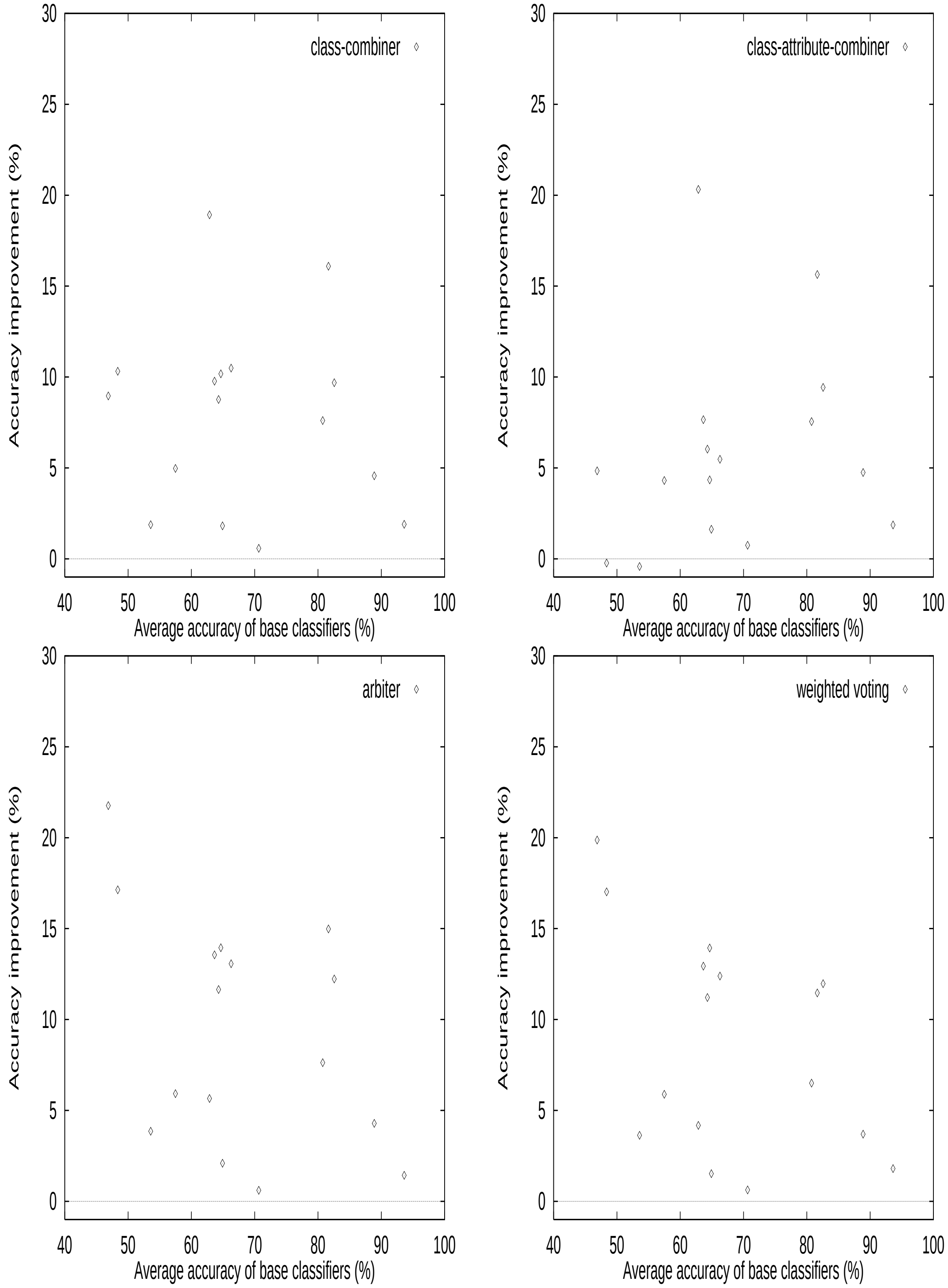


Figure 8.3: Average accuracy of base classifiers vs. accuracy improvement.

entropy is, the probabilities are more evenly distributed (more random). The smaller the entropy is, the probabilities are more biased (some events are more likely).

$$entropy = -\sum_{i}^{m} p_i \log(p_i) \tag{8.6}$$

where $m$ is the number of events and $p_i$ is the probability of event $i$. The range of *entropy* is [0, $\log m$]. In this study, because $m$ varies within our metrics, entropy is usually normalized by $\log m$. That is, normalized entropy has a range of [0, 1]. Also, we use base 2 for logarithm.

Our first metric is called *diversity*. It measures how different the base classifiers are based on their predictions. For each instance $y_i$, the fraction of base classifiers, $p_{ik}$, predicting $class_k$ is calculated as follows:

$$p_{ik} = \frac{1}{b}\sum_{j}^{b} OneIfTrue(C_j(y_i) = class_k))$$

Using $p_{ik}$, the entropy in the predictions for each instance is calculated, which is then normalized by $\log c$ and averaged by the number of instances, $n$. That is,

$$diversity = \frac{1}{n}\sum_{i}^{n} \frac{1}{\log c}\sum_{k}^{c} -p_{ik}\log(p_{ik}) \tag{8.7}$$

The range of *diversity* is [0 ,1]. When the value of *diversity* grows, the predictions from the base classifiers are more evenly distributed and, therefore, more diverse.

Figure 8.4 plots how *diversity* affects the amount of accuracy improvement. The four graphs, as mentioned previously, present results from the four schemes evaluated in this study. The fitted line shows a general trend of the relationship between *diversity* and *accuracy improvement*. We observe that, in all four graphs, accuracy improvement increases with diversity. That is, larger improvement in accuracy can be achieved by integrating more diverse base classifiers. This result concurs with other results in the literature. However, we formally define and quantitatively measure diversity using entropy.

Krogh and Vedelsby (1995) measures diversity, called *ambiguity*, by calculating the mean square difference between the prediction made by the ensemble and the

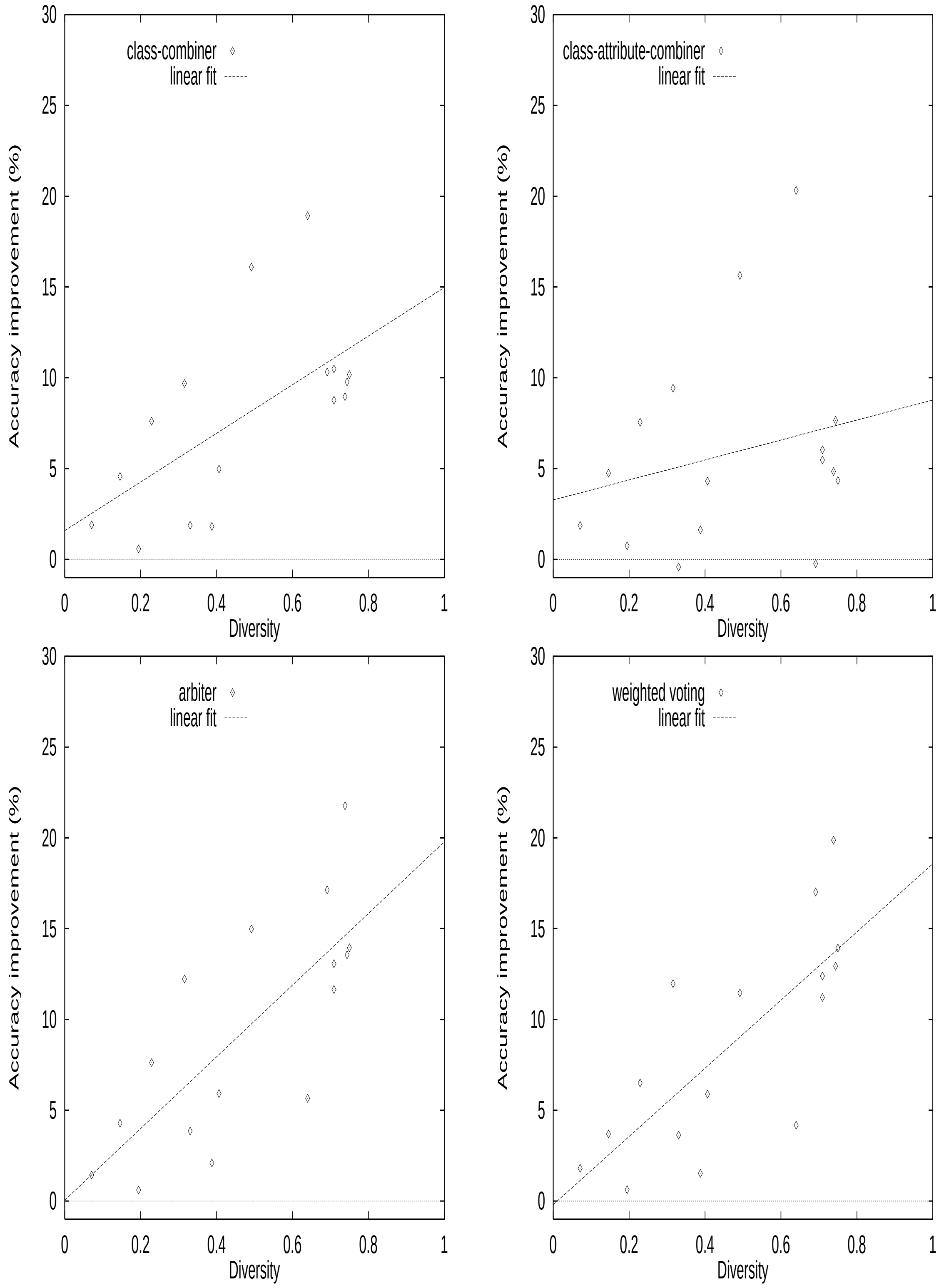


Figure 8.4: Diversity of base classifiers vs. accuracy improvement.

base classifiers. They proved that increasing ambiguity/diversity decreases overall error. Our diversity metric does not involve the predictions generated by integrating structures (ensemble) and only measures the variation among the base classifiers. Brodley and Lane (1996) measure diversity, called *overlap*, by counting the number of instances that are classified the same way by each of the base classifiers. The *overlap* metric does not differentiate instances that gather two different predictions from those that gather more.

Recent statistical work formulates classification error via *bias-variance decomposition*. In short, *bias* measures, on the average over all possible training sets of a given size, the error rate of the learned classifiers and *variance* measures how different the learned classifiers are when different training sets are used. That is, classification error can be explained by errors caused by bias and variance. Kong and Dietterich (1995) show that their error-correcting code method for combining binary classifiers reduces errors by correcting both bias and variance errors. Their decomposition is based on the commonly used zero-one loss functions (misclassification rates). Breiman (1996a) explains that *unstable* methods/algorithms (those with high variance) benefit from aggregating/combining classifiers learned from different samples of the training set. Kohavi and Wolpert (1996) provide a more robust decomposition that eliminates the possibility of negative variance. Although not explicitly stated, Krogh and Vedelsby's (1995) decomposition of squared classification error follows the same spirit of bias-variance decomposition and their *ambiguity* metric measures variance. Our *diversity* metric tries to approximate the variance characteristics as well.

### 8.2.3 Coverage

*Coverage* (Brodley & Lane, 1996) measures the fraction of instances for which at least one of the base classifiers produces the correct predictions. That is, an instance is not *covered* if and only if all the base classifiers generate an incorrect prediction for that instance. If an integrating method does not make a prediction other than those from the base classifiers, coverage is the maximum possible accuracy. That is, coverage is an upper bound on accuracy for certain integrating methods. Formally,

$$\begin{aligned} coverage = & \\ & 1 - \frac{1}{n}\sum_{i}^{n} OneIfTrue((C_1(y_i) \neq class(y_i)) \wedge (C_2(y_i) \neq class(y_i)) \wedge \\ & \qquad \cdots \wedge (C_b(y_i) \neq class(y_i))) \end{aligned} \tag{8.8}$$

The range of *coverage* is [0 ,1]. Coverage of one means that each of the instances is correctly predicted by at least one base classifier. A zero coverage implies none of the base classifiers can correctly predict any of the instances.

Figure 8.5 depicts the relationship between *coverage* and *accuracy improvement*. We observe that, in all four schemes, an increase in coverage implies larger accuracy improvement. A high *coverage* is particularly important for integrating schemes that utilize only the predictions generated by the base classifiers because the upper bound on accuracy for these schemes is coverage. One such scheme is voting—the final prediction is always one of the predictions generated by the base classifiers.

### Coverage-possible accuracy improvement

As stated earlier, *coverage* provides an upper bound on accuracy improvement assuming the integrating structure does not make a prediction other than the ones from the base classifiers. Although the assumption does not hold for our meta-learning strategies, we would like to see how close our strategies can get to that upper bound or maybe beat it. We note that the ultimate upper bound is 100% correct, not the *coverage*; however, *coverage* provides a practical and sensible yardstick for comparison. We define *coverage-possible accuracy improvement* as the largest possible improvement in accuracy provided by *coverage*. Formally,

$$coverage\text{–}possible\ accuracy\ improvement = \frac{coverage - \beta}{\beta} \times 100\% \tag{8.9}$$

That is, the metrics measures the accuracy improvement obtained when an integrating scheme achieves the *coverage* level of accuracy.

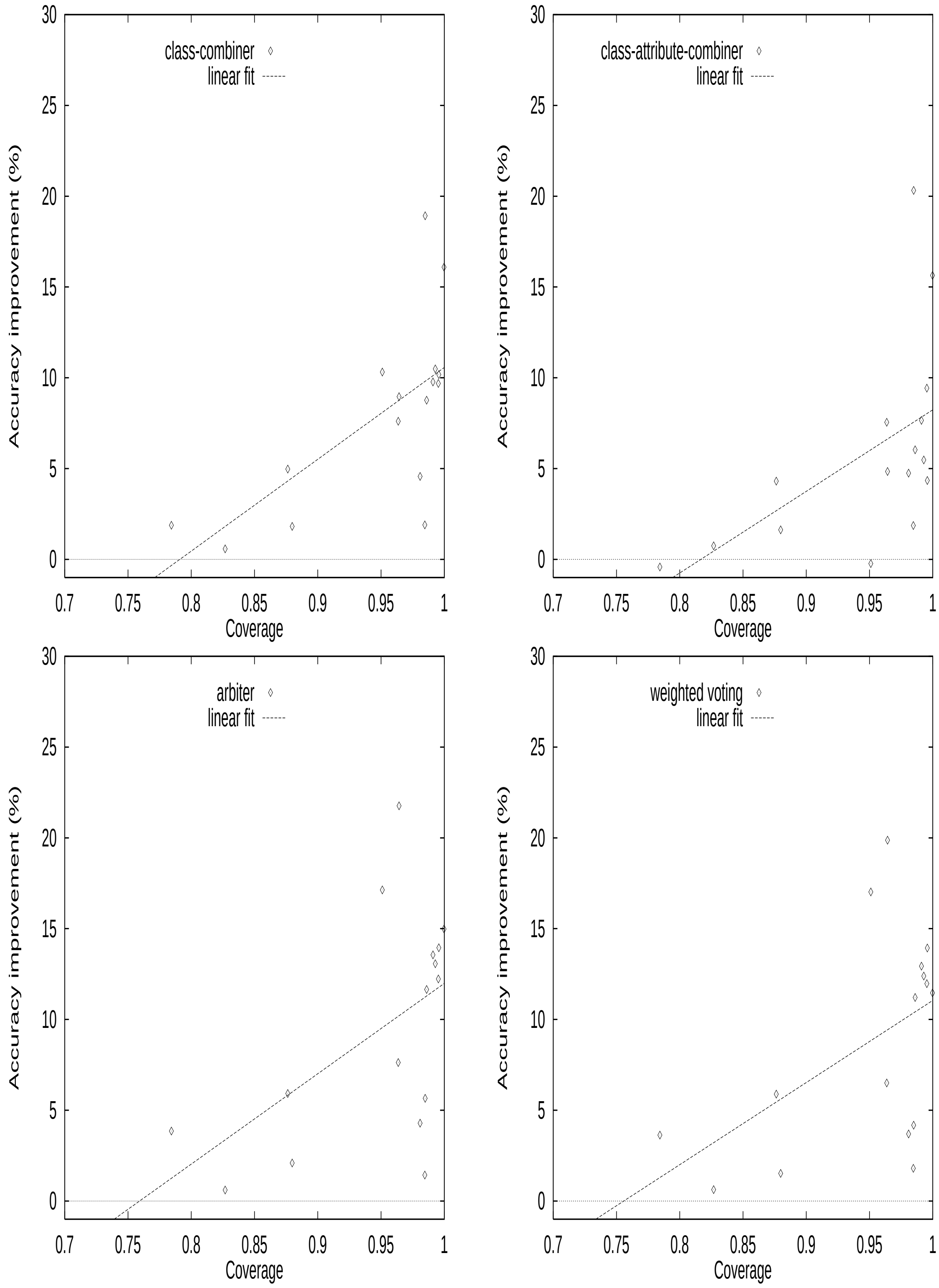


Figure 8.5: Coverage of base classifiers vs. accuracy improvement.

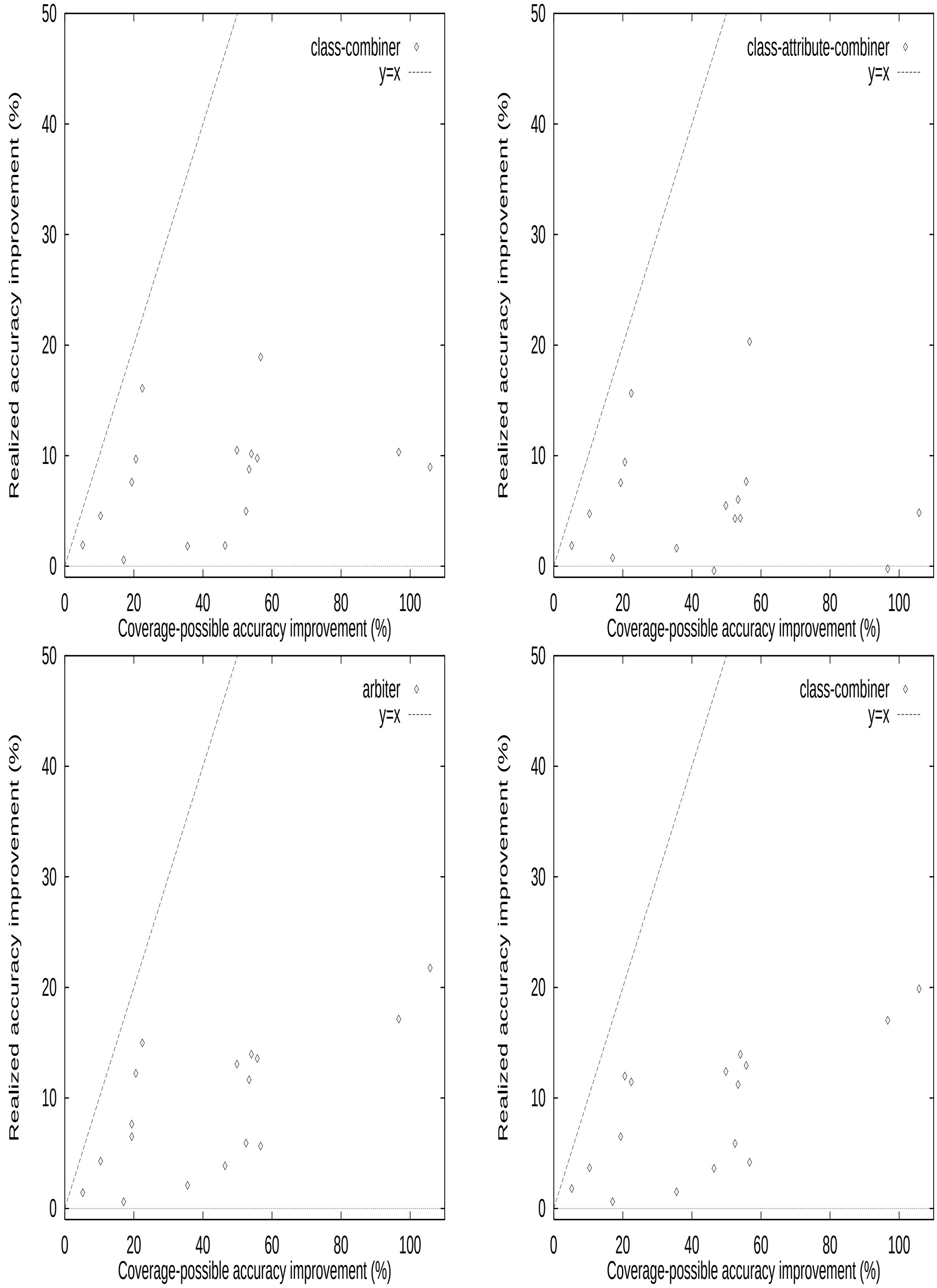


Figure 8.6: Coverage-possible accuracy improvement vs. realized accuracy improvement.

Figure 8.6 depicts the relationship between *coverage-possible accuracy improvement* and realized *accuracy improvement* by the four integrating schemes. A linear line, $y = x$, is also drawn in each graph. None of the cases achieve an improvement larger than the coverage-possible. However, for the cases with smaller improvement, five or six of them have improvement levels quite close to the coverage-possible ones.

### 8.2.4 Correlated error

*Correlated error*, introduced by Ali and Pazzani (1996), measures the fraction of instances for which a pair of base classifiers make the same incorrect prediction. The fraction is calculated for each pair of base classifiers ($C_j$ and $C_k$):

$$\frac{1}{n}\sum_{i}^{n} OneIfTrue(C_j(y_i) = C_k(y_i) \neq class(y_i))$$

This fraction is then summed and averaged over every possible pair of base classifiers. That is,

$$correlated\ error = \frac{1}{b \times (b-1)\ /\ 2}\sum_{j}^{b}\sum_{k=i+1}^{b}\frac{1}{n}\sum_{i}^{n} OneIfTrue(C_j(y_i) = C_k(y_i) \neq class(y_i)) \quad (8.10)$$

The range of *correlated error* is [0, 1]. A value close to one indicates the errors made by the base classifiers are not likely to be independent.

Figure 8.7 depicts the relationship between the *correlated error* of base classifiers and *accuracy improvement*. We observe a general decreasing trend in *accuracy improvement* when *correlated error* increases. Our findings here are consistent with those from (Ali & Pazzani, 1996).

Hansen and Salamon (1990) proved that for a neural-network ensemble, if the networks produce independent errors and have accuracy of at least 50%, the expected ensemble error rate goes to zero as the number of networks approaches infinity. *Correlated error* is attempt to characterize the degree of errors that are not independent.

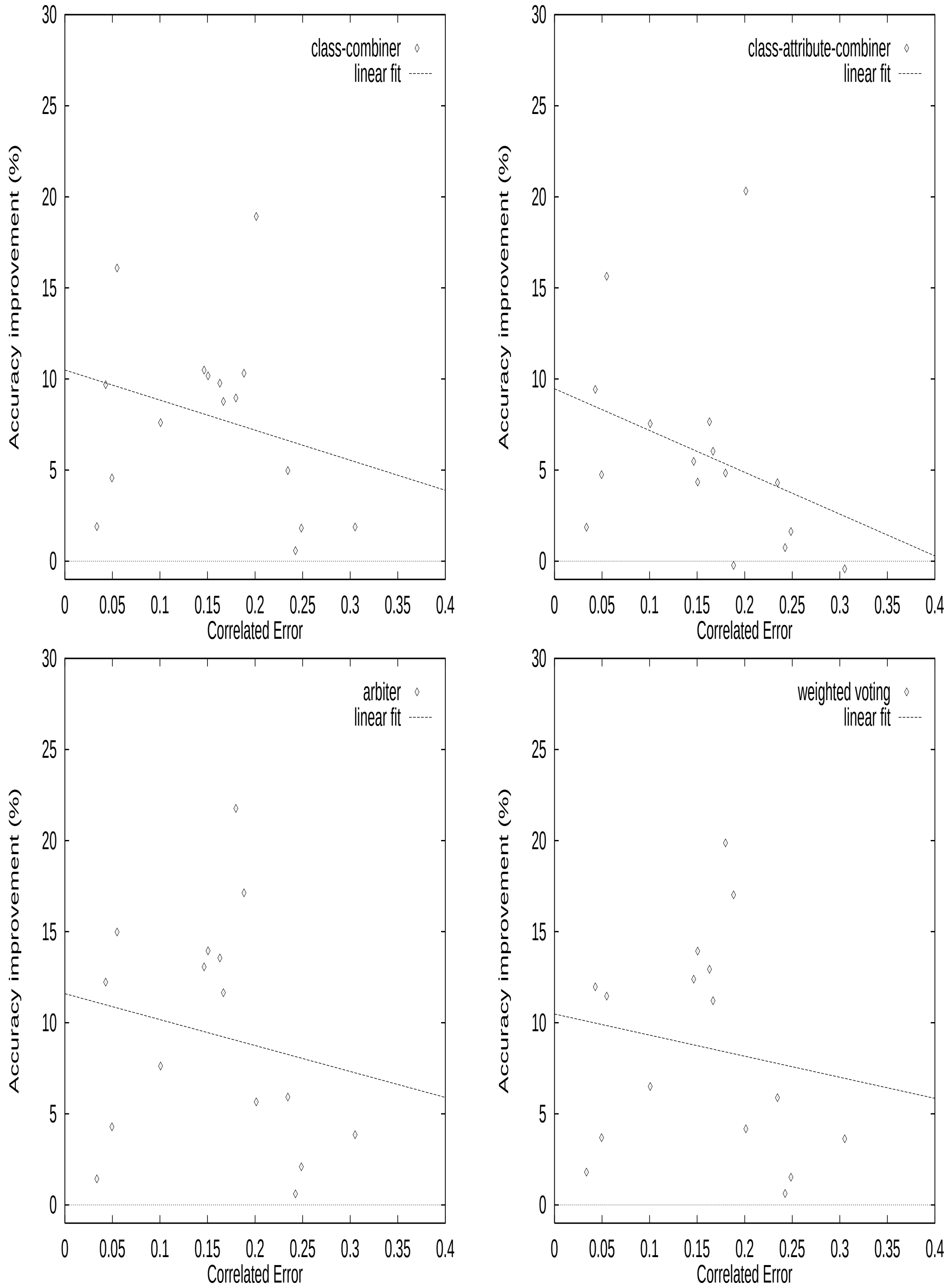


Figure 8.7: Correlated error of base classifiers vs. accuracy improvement.

### 8.2.5 Specialty

Some base classifiers might be biased toward certain classes. That is, they are more accurate in predicting certain classes than others. How specialized the base classifiers are can contribute to the behavior of various integrating schemes.

A *specialty* metric for the base classifiers is defined as follows. For each base classifier, we calculate its accuracy of predicting the different classes. $a_{jk}$ is the accuracy of classifier $j$ on class $k$. Formally,

$$a_{jk} = \frac{\sum_i^n OneIfTrue(C_j(y_i) = class(y_i) = class_k)}{\sum_i^n OneIfTrue(class(y_i) = class_k)}$$

For each classifier, $a_{jk}$ is normalized by the sum of $a_{jk}$, yielding $p_{jk}$. The sum of $p_{jk}$ is one.

$$p_{jk} = \frac{a_{jk}}{\sum^k a_{jk}}$$

For each classifier, using $p_{jk}$, the entropy is calculated, normalized by $\log c$. *Specialty* is average normalized entropy (Equation 8.6) over $b$ classifiers. Formally,

$$specialty = 1 - \frac{1}{b}\sum_j^b \frac{1}{\log c}\sum_k^c -p_{jk}\log(p_{jk}) \qquad (8.11)$$

The range of *specialty* is [0 ,1]. The larger the value is, the base classifiers are more biased and specialized to certain classes.

Figure 8.8 depicts the relationship between the *specialty* metric for the base classifiers and *accuracy improvement*. The fitted line indicates a slightly decreasing trend in *accuracy improvement* when *specialty* increases. A closer inspection reveals that there is an outlier with a .34 *specialty* value (generated by ID3 in the Secondary Structure data set). When we fit the data points except the outlier, we observe that the decreasing trend is reversed to an increasing trend for the two *combiner* schemes (top two graphs). This result is consistent with the notion that combiners are trained to recognize the behavior and relationship among base classifiers. Class bias or specialization is one such behavior. *Specialty* seems to have little effect on the *arbiter* and *weighted voting* schemes.

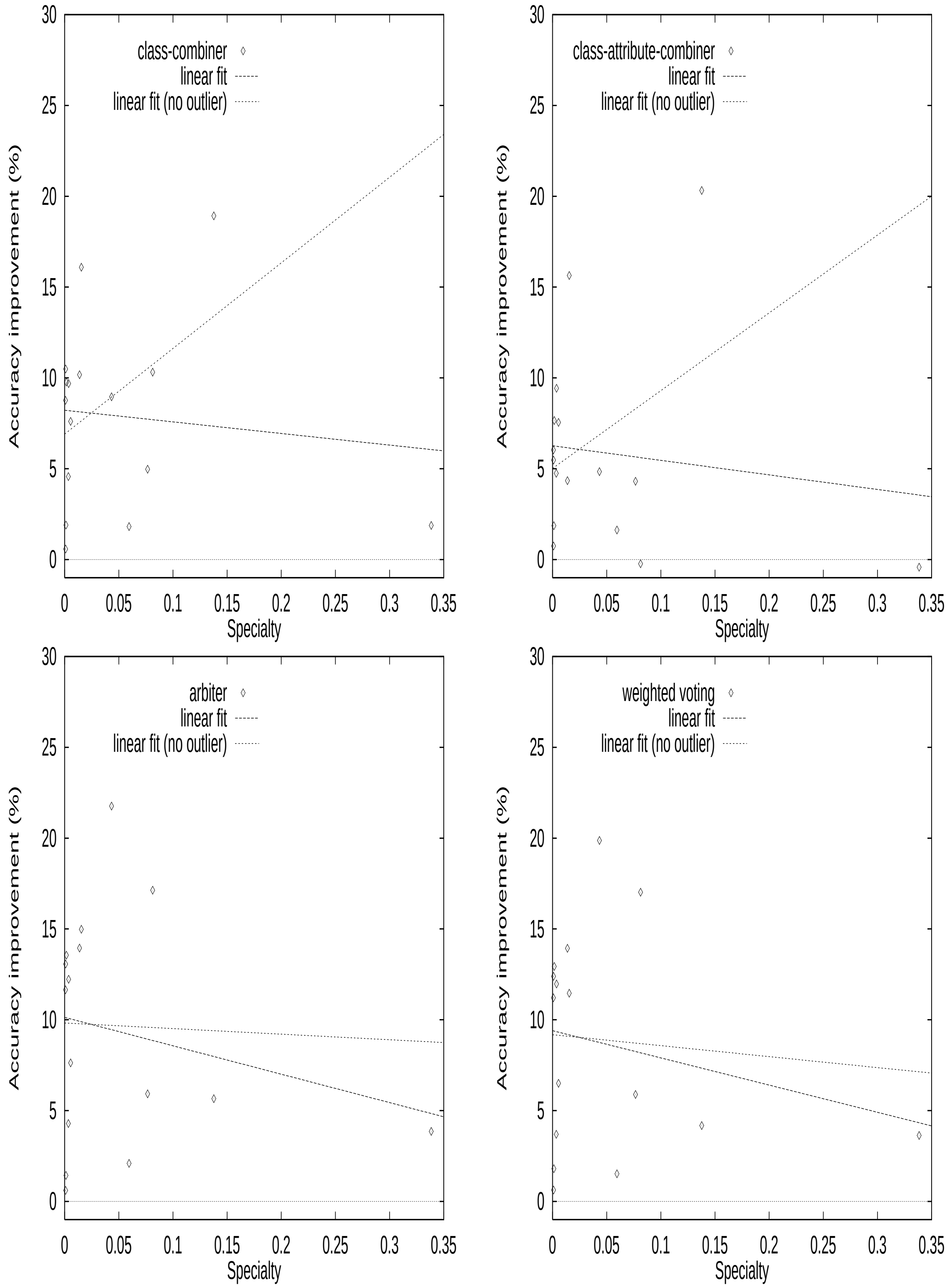


Figure 8.8: Specialty of base classifiers vs. accuracy improvement.

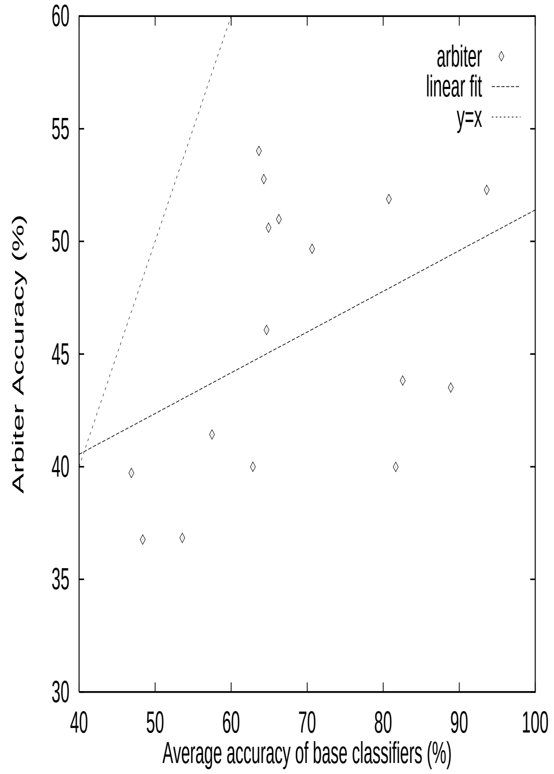


Figure 8.9: Average accuracy of base classifiers vs. arbiter accuracy.

## 8.3 Analyzing Arbiters

In the previous sections we focus on analyzing the base classifiers, here we concentrate on analyzing the behavior of the arbiter strategy.

### 8.3.1 Arbiter accuracy

*Arbiter accuracy* is defined as the accuracy of $\mathcal{ARB}$ on the unseen instances. Let $\mathcal{ARB}$ be an arbiter and $\mathcal{ARB}(y_i)$ be the $\mathcal{ARB}$'s classification of $y_i$. That is,

$$arbiter\ accuracy = \frac{1}{n}\sum_{i}^{n} OneIfTrue(\mathcal{ARB}(y_i) = class(y_i)) \times 100\% \qquad (8.12)$$

Figure 8.9 plots the *arbiter accuracy* against the *average accuracy of base classifiers*. We first observe that the arbiter accuracy is lower than the accuracy of base classifiers (to the right of the $y = x$ line). Recall that the training set for an arbiter contains examples that are confusing to the base classifiers. In other words, examples that are difficult to learn from are in the arbiter training set. This likely attributes to lower accuracy of the arbiters relative to the base classifiers.

Furthermore, the arbiter accuracy demonstrates an increasing trend when the base classifiers are more accurate. A higher accuracy in the base classifiers implies the employed learning algorithm is closely suited for the involved data set. Accordingly, the arbiter, essentially another classifier, also has a higher accuracy.

### 8.3.2 Arbiter usage

We next examine how frequent an arbiter is utilized. Recall that an arbiter is called upon when the majority of base classifiers do not agree on the same prediction. That is, an arbiter is not always used in determining the overall prediction. We define *arbiter usage* as the percentage of instances that do not have a majority prediction and uses the arbiter's prediction as the overall prediction. Let $p_1, p_2, \cdots, p_b$ be the predictions generated by the $b$ base classifiers and $class\_count_k$ be the number of predictions that are of class $k$. That is,

$$class\_count_k = \sum_i^b OneIfTrue(p_i = class_k).$$

Furthermore, let

$$no\_majority(p_1, p_2, \cdots, p_b) = \begin{cases} false & \exists k \; class\_count_k > b/2 \\ true & \text{otherwise} \end{cases} \tag{8.13}$$

Finally,

$$arbiter\ usage = \frac{1}{n}\sum_i^n OneIfTure(no\_majority(C_1(y_i), C_2(y_i), \cdots, C_b(y_i)) \wedge (\mathcal{OC}(y_i) = \mathcal{ARB}(y_i))) \times 100\% \tag{8.14}$$

Figure 8.10 depicts *arbiter usage* against the average accuracy of base classifiers. As expected, when the base classifiers are highly accurate, their predictions frequently

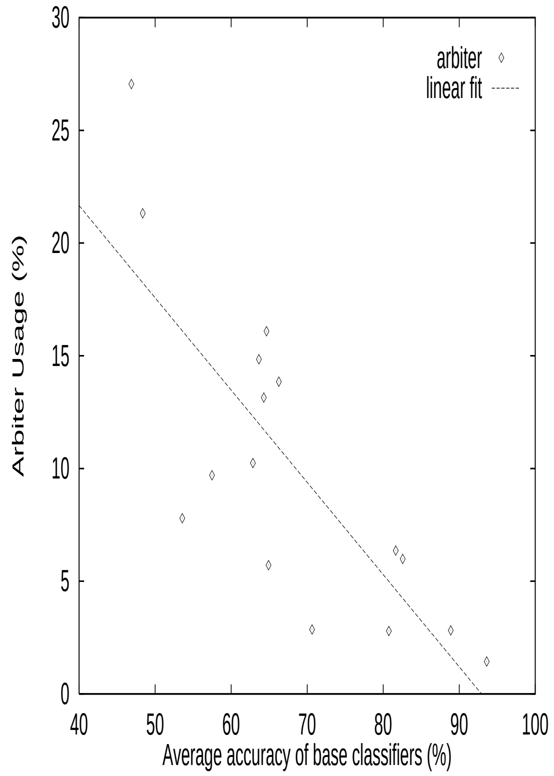


Figure 8.10: Average accuracy of base classifiers vs. arbiter usage.

reach a majority, and hence the arbiters are scarcely utilized. That is, the arbiters are only used when the base predictions are vastly different, which suggests some of the base predictions are incorrect.

### 8.3.3 Arbiter effectiveness

*Arbiter effectiveness* calculates the rate an arbiter is correct when its prediction is used in the overall prediction. That is, it measures how useful an arbiter is when it is used. We define

$$arbiter\ effectiveness = \sum_{i}^{n} \frac{\begin{array}{c} OneIfTure(no\_majority(C_1(y_i), C_2(y_i), \cdots, C_b(y_i)) \wedge (\mathcal{OC}(y_i) = \mathcal{ARB}(y_i)) \wedge \\ (\mathcal{ARB}(y_i) = class(y_i))) \end{array}}{OneIfTure(no\_majority(C_1(y_i), C_2(y_i), \cdots, C_b(y_i)) \wedge (\mathcal{OC}(y_i) = \mathcal{ARB}(y_i)))} \times 100\% \qquad (8.15)$$

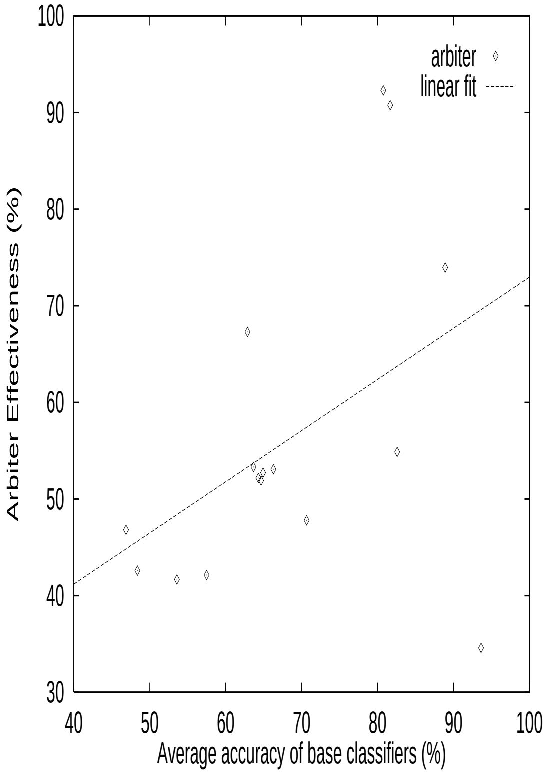


Figure 8.11: Average accuracy of base classifiers vs. arbiter effectiveness.

The numerator is the number of times arbiter $\mathcal{ARB}$'s prediction is used as the overall prediction and is correct; the denominator is the number of times the arbiter's prediction is used as the overall prediction.

Figure 8.11 plots *arbiter effectiveness* against the average accuracy of base classifiers. We observe that arbiters are more effective when the average accuracy of base classifiers is higher. That is caused by lower arbiter usage and higher arbiter accuracy.

As mentioned previously, when the base classifiers have a low accuracy, the arbiters are more frequently used. In order to improve the arbiter effectiveness, we need to improve the accuracy of arbiters. One approach is to more carefully choose the training set for the arbiters when the base classifiers are not very accurate. Another approach is to use an alternative learning algorithm for the arbiter since the algorithm used to generate the base classifiers is not well suited for the particular data domain.

## 8.4 Summary

We defined four metrics (*diversity*, *coverage*, *correlated error*, and *specialty*) for characterizing the base classifiers and explored the effects of these characteristics on the behavior of various integrating schemes. From our results, larger accuracy improvement can be achieved by more *diverse* base classifiers with higher *coverage* and fewer *correlated errors*. For integrating schemes (*combiner* in our case) that recognize relationships among the base classifiers, more *specialized* base classifiers can result in larger improvement in accuracy. Analyses on the arbiter strategy shows that when the base classifiers are less accurate, the arbiter needs to be built more carefully.

# Chapter 9

# Efficiency and Scalability

So far in this thesis, we have been concentrating on the accuracy performance of meta-learning. In this chapter we focus on the training time performance of meta-learning. We refer to *efficiency* as speed or how fast a system runs and *scalability* as the ability of a system to handle increasing amounts of data without needing an extra order of magnitude of increasing computational resources. Scalability is further defined in Section 9.2.1.

We first analyze the training time complexity and performance of the individual learning algorithms used in this thesis in a serial environment. We then examine the speedup that can be obtained by utilizing meta-learning in a parallel and distributed environment.

## 9.1 Serial Evaluation of Learning Algorithms

To evaluate the five learning algorithms (ID3, CART, BAYES, WPEBLS, CN2) in a serial environment, we first formulate their theoretical time complexity and then empirically investigate their speed with varying amounts of training data.

### 9.1.1 Theoretical time complexity

In the following discussion we sketch the worst-case time complexity for each of the five algorithms to help clarify the potential benefits of scaling by meta-learning techniques. For simplicity, we assume all the attributes of the training data have discrete values. Let

- $a$ = the number of attributes,
- $v$ = the largest number of distinct values for an attribute (i.e,., the size of its domain), and
- $n$ = the number of training examples.

The time complexity of ID3 (Quinlan, 1986) is a function of the number of levels in the decision tree it forms. The height of the tree is bounded by the number of attributes, $O(a)$. Since at each level $O(a)$ attributes are evaluated with $O(n)$ examples, the time spent at each level is $O(an)$. Therefore, the time complexity of ID3 is $O(a^2n)$ in the worst case.

In CART (Breiman *et al.*, 1984; Buntine & Caruana, 1991) the values of each attribute at each node are grouped into two disjoint subsets. Hence, each non-leaf node has only two branches and the learned tree has $O(2^a)$ nodes. At each node, CART uses a greedy scheme to group the values of each attribute, which takes roughly $O(v)$ time. That is, $O(av + an)$ time is needed to group $a$ attributes and evaluate $a$ attributes for $n$ examples. Although, CART employs a ten-fold cross-validation scheme to select the splitting attribute, the scheme only adds a constant factor to the time complexity at each node and hence the complexity remains at $O(av + an)$. The total time complexity for CART is therefore $O((av + an)2^a)$ in the worst case.

BAYES (Clark & Niblett, 1989) calculates the conditional probabilities for each attribute value given a class and the probabilities for each class. $O(av)$ conditional probabilities are calculated and each takes $O(n)$ time, hence $O(avn)$ time is needed.

The class probabilities can be calculated in $O(n)$ time. Therefore, the time complexity of BAYES is $O(avn + n)$ or $O(avn)$.

WPEBLS (Cost & Salzberg, 1993) calculates a set of value distance matrices (VDMs) and a vector of weights for the exemplars. Each attribute has a VDM of size $v$ by $v$, which takes $O(nv^2)$ to calculate. For $a$ attributes, $O(anv^2)$ time is needed for $a$ VDMs. The weight vector is incrementally updated and takes $O(n^2)$ time. The time complexity for WPEBLS is therefore $O(anv^2 + n^2)$ in the worst case.

The time complexity of CN2 (Clark & Niblett, 1989; Chan, 1988) is a function of how many *complexes* (candidate antecedents (or LHS's) of a rule) are evaluated. Since CN2 performs a general-to-specific beam search on the complexes, a fixed number of complexes is retained at each specialization step. The beam size is called the *star* size, which is denoted by $s$. At each specialization step, $O(avs)$ complexes are generated. Evaluating all the complexes against $n$ examples takes $O(avsn)$ time. The top $s$ complexes can be found in $O(avs^2)$ time. As a result, each step takes $O(avs(n + s))$ time. This step could be repeated $O(a)$ times to find a rule, which consequently takes $O(a^2vs(n + s))$ time to induce. Since at least one training example is covered by an induced rule, $O(n)$ rules can be produced. Accordingly, CN2's total time complexity is $O(a^2vsn(n + s))$. Because $s$ is a fixed parameter and $n$ is much larger than $s$, the complexity can be reduced to $O(a^2vn^2)$, which is quadratic in the number of examples.

Since we are considering problems with potentially large amounts of data, the dominating term is $n$. From the above analysis, only WPEBLS and CN2 are quadratic in the number of training examples and the rest are linear with respect to the number of examples. However, closer inspection reveals that $v$, the number of values of an attribute, could be a function of $n$. One can easily see that some values of an attribute which are present in a large data set might be absent from a small data set. That is, in addition to $n$, $v$ could be a significant factor in time performance when large amounts of data are used.

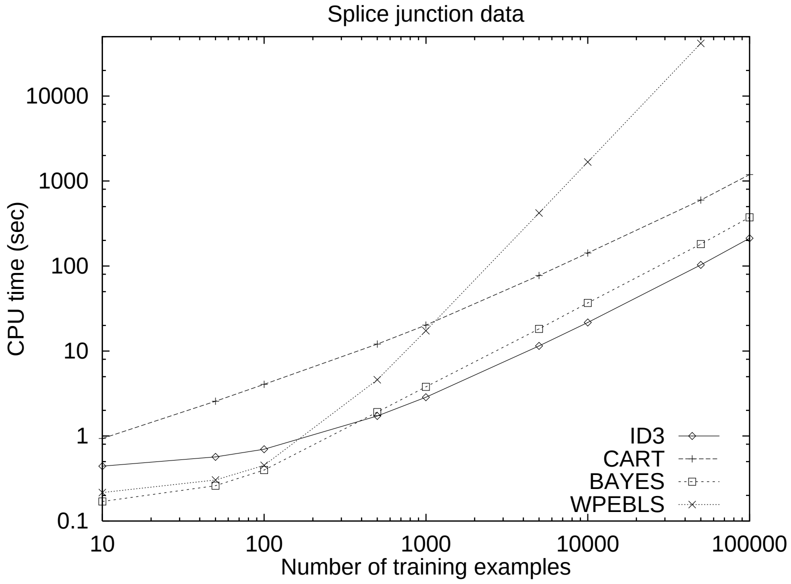


Figure 9.1: Training time vs. number of examples in splice junctions.

### 9.1.2 Empirical time performance

We performed two sets of experiments: the first set used the splice junction data with training set size up to 100,000 examples, the second set used the artificial data with set size up to 10 million when all algorithms exceeded the main memory.

#### Splice junctions

In a set of experiments we measured the *CPU* training time of ID3, CART, BAYES, and WPEBLS with the number of training examples varying from 10 to 100,000 in the splice junction domain (examples were randomly selected and duplicated from the original data set, which has 3,190 examples) (Chan & Stolfo, 1994). Thus, the training sets contain many duplicate examples. The results in CPU time on Sun IPXs are plotted in Figure 9.1. We observe that WPEBLS performed comparatively worse than the other three algorithms when more training examples were presented. With 100,000 examples, WPEBLS did not finish running after a couple of days. ID3 and BAYES were generally faster than CART. BAYES was faster than ID3 until the crossover at 500 examples. CART was slower than WPEBLS until the training set grew to 1,000 examples.

Figure 9.2 depicts our results in four graphs. Each graph plots the CPU training time against the number of training examples for a different learning algorithm. Polynomial curves are fitted to the data points to illustrate how the algorithms behave in terms of speed. We tried linear ($y = ax + b$), quadratic ($y = ax^2 + bx + c$), and cubic ($y = ax^3 + bx^2 + cx + d$) equations for curve fitting, where $x$ is the number of training examples and $y$ is training time in seconds. The curve approximations were computed using GNUFIT (Grammes, 1993) with the Marquardt-Levenberg algorithm (Ralston & Rabinowitz, 1978; Press *et al.*, 1988), a nonlinear least squares fit mechanism. To approximate the training speed with polynomial equations, we inspect how closely the three polynomials fit the data points. Curve fitting errors near the bottom left corner are less important since the values in question are much smaller due to the log scale.

The three curves seem to fit ID3, equally well, hence ID3 appear to have linear speed with up to 100,000 training examples. CART and BAYES seem to be close to having linear speed. However, WPEBLS clearly exhibits superlinear speed—the linear fitted curve does not fit at all. The quadratic and cubic curves fit much more closely. (These two curves overlap in WPEBLS plot in Figure 9.2). Hence, WPEBLS' speed appears to be quadratic in the number of examples. The fitted quadratic equation for WPEBLS is $y = .0000166x^2 + .000916x + .100$ and it projects that WPEBLS takes about 16.6 million CPU seconds (or 192 days or 6.4 months) to process 1 million records.

Three of the algorithms appear to exhibit linear speed with up to 100,000 training examples. We next investigate speed performance of the algorithms with much more data.

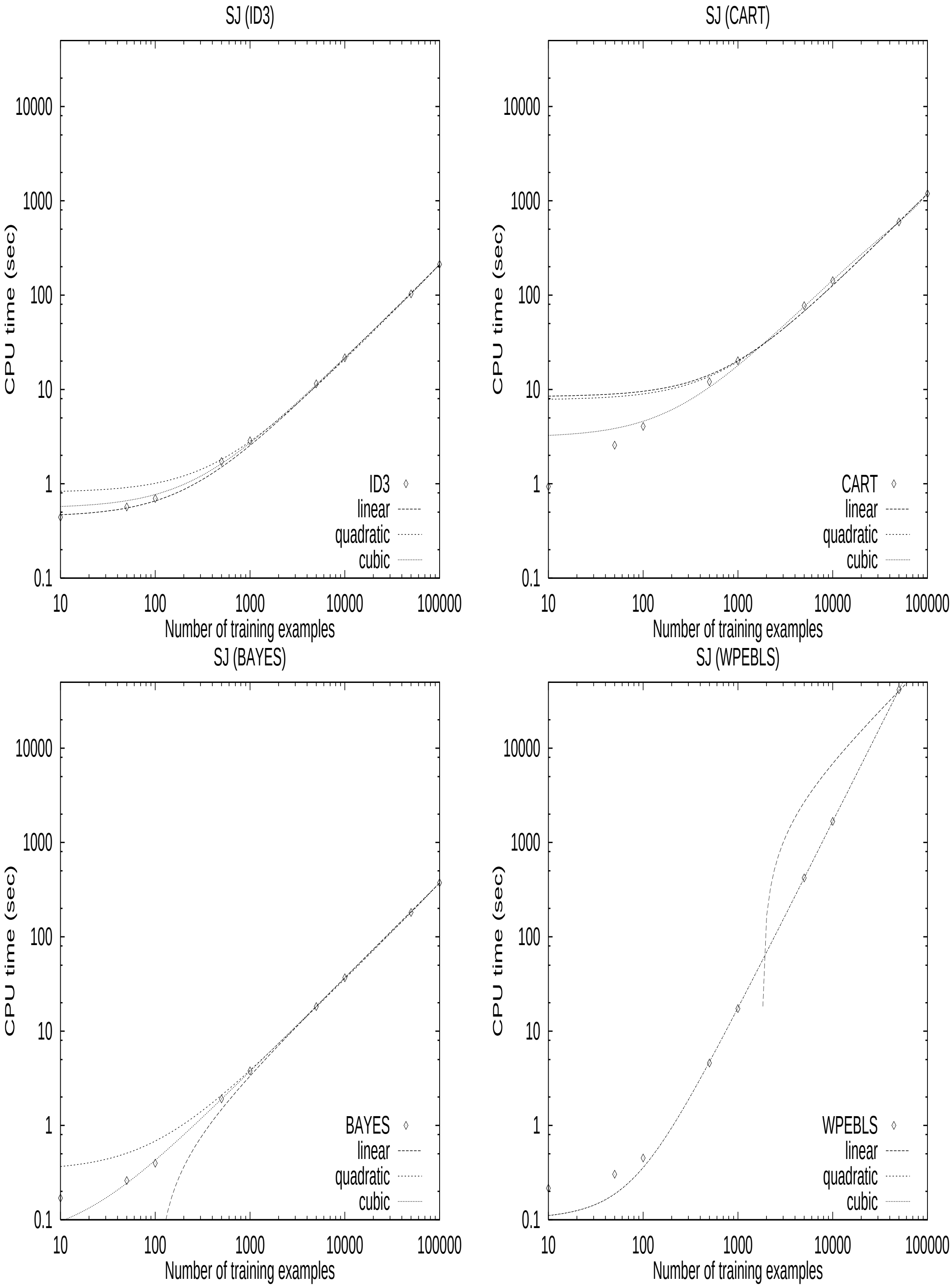


Figure 9.2: Training time vs. number of examples in splice junctions with polynomial curve fitting.

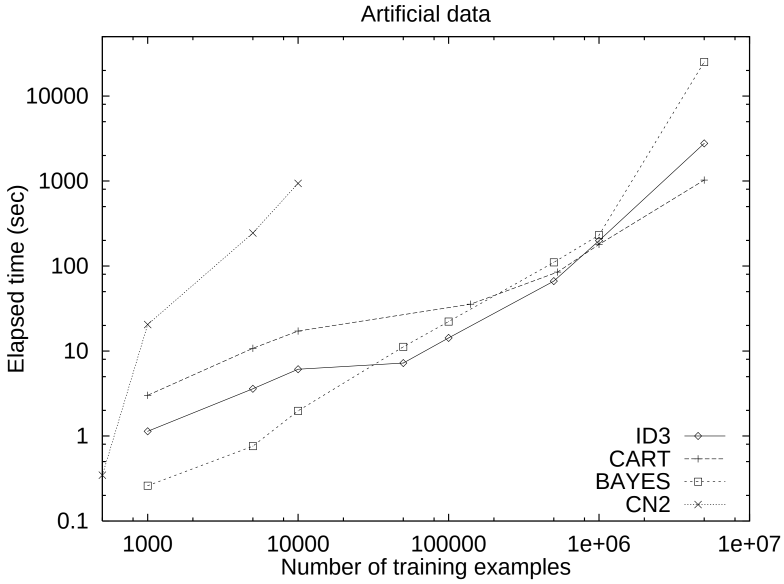


Figure 9.3: Training time vs. number of examples in artificial data.

### Artificial data

In the second set of experiments we measured the *elapsed* training time of ID3, CART, BAYES, and CN2 with different numbers of examples in the artificial domain. The experiments were performed on HP 9000/735 workstations, which are faster than the SUN IPX workstations used in the previous set of experiments. The number of training examples was exponentially increased until the algorithms/operating system reported insufficient-memory errors. That is, they ran out of memory when the training set got too large, which was expected for main-memory based algorithms.

Figure 9.3 plots the relative performance of the four algorithms. Each data point is an average of five runs using data sets generated by different seeds for the random number generator. Since $4.6 \times 10^{17}$ different examples are possible, the chances of having duplicates in a data set with up to 10 million examples are quite low. ID3, CART, and BAYES ran out of memory while processing 10 million records, and CN2 while processing 50,000 records. Memory resources do pose a limit on how much data learning algorithms can digest. With fewer than 10,000 examples, BAYES was the fastest, followed by ID3, CART, and CN2. Crossovers among ID3, CART, and

BAYES occur between 100,000 and 1 million examples. With 5 million examples, CART was faster than ID3 and BAYES was the slowest.

ID3 completed processing 5 million records in about 2,800 seconds (47 minutes), which is much less than Catlett's (1991) projection of several months for ID3 to process 1 million records. The huge gap merits some explanation. First, the projection was made five years ago, state-of-the-art processor speed has much improved since then. Second, the artificial data set has only eight attributes, four of which are numeric, and two (Boolean) classes, the data set Catlett used has seven numeric attributes and nine classes. Since ID3 performs a sort on the values of numeric attributes, symbolic attributes are faster to evaluate than numeric ones. Furthermore, a nine-class problem is more complex than a two-class problem. Third, the artificial data set has a well defined concept to learn—the Boolean expression that generates it. The NASA shuttle data set Catlett used is real-world and the target concept is potentially much more complex than the Boolean expression we used. Unfortunately, we were not able to obtain the full data set from Catlett for our investigation to validate the published result.

As in the previous set of experiments, we fitted linear, quadratic, and cubic equations to the training time of each algorithm and the plots are displayed in Figure 9.4. We observed that none of the algorithms exhibited linear speed. The quadratic and cubic curves fit ID3 and CART closely, hence, ID3 and CART appear to be quadratic. BAYES does not appear to be linear or quadratic; the closest is the cubic approximation. The cubic curve fits CN2 the closest, although the quadratic curve is also close. The quadratic fitted polynomial for CN2 is $y = .00000896x^2 + .00420x + 1.18$ and it projects that CN2 takes about 9 million elapsed seconds (104 days or 3.5 months) to process 1 million records. The cubic fitted polynomial for CN2 is $y = .000000000597x^3 - .000000363x^2 + 0.0398x - 19.5$ and it projects that CN2 consumes 597 million elapsed seconds (18.9 years) to learn from 1 million examples. Recall that CN2 did not have enough memory space to process 50,000 records. Even if sufficient memory resources are provided for CN2 to process 1 million records, a period of 3.5

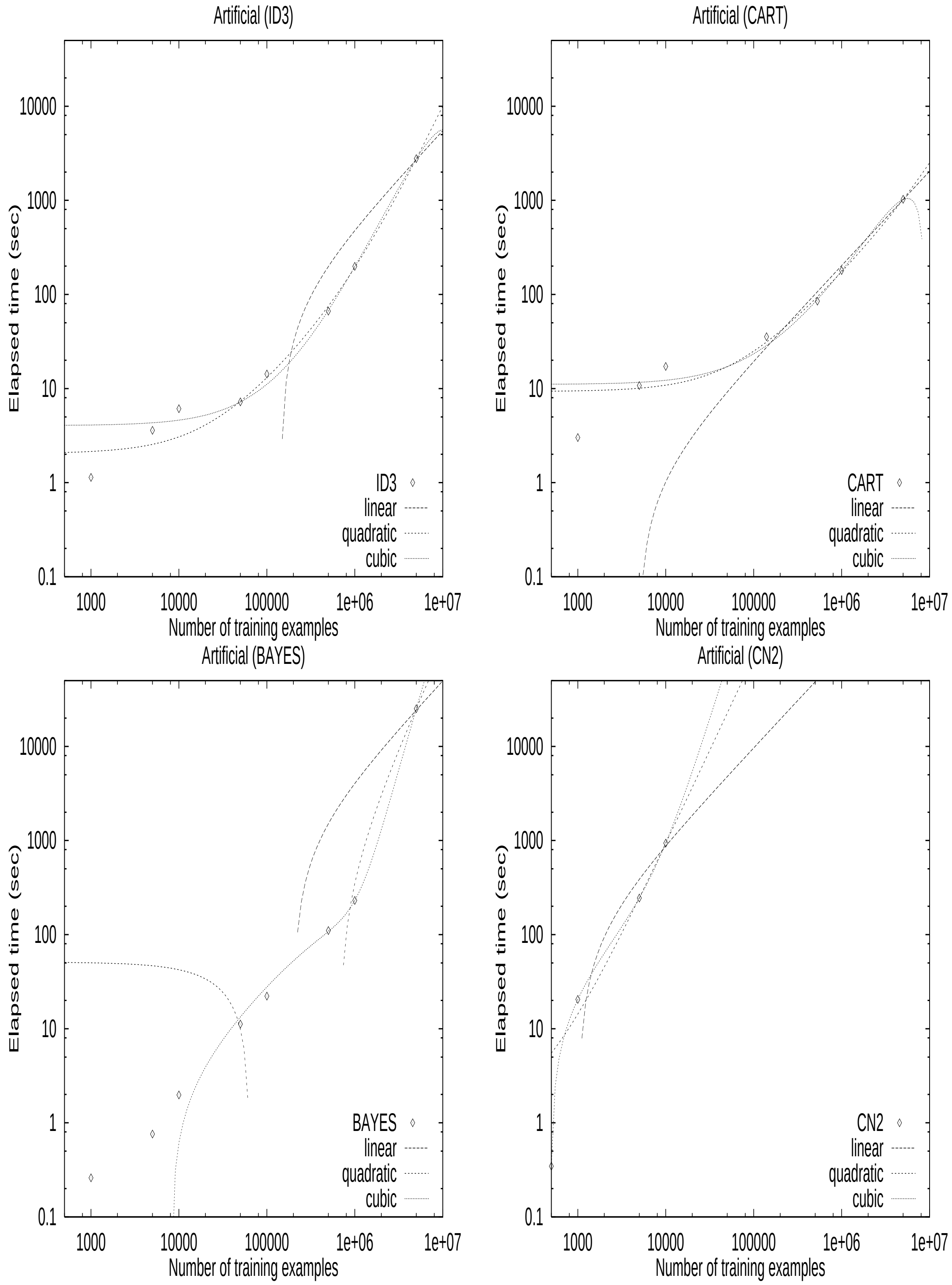


Figure 9.4: Training time vs. number of examples in artificial data with polynomial curve fitting.

months (quadratic approximation) or 18.9 years (cubic approximation) is a long time to wait.

The results from these experiments reinforce our hypotheses on the behavior of memory-based learning algorithms in the presence of large data sets in real life. First, theoretical analysis provides a powerful tool to analyze time complexity and produces close approximations. However, practical time performance might differ from theoretical analysis, especially when worst case-analysis is used as we did. With large amounts of data, one attributing factor is the characteristics of operating system's memory management, which might elect to utilize secondary storage and result in time-consuming memory transfers. Second, memory resources are limited and very large data sets can exceed them; consequently, these learning algorithms are rendered relatively useless when they are faced with too much information. Third, they exhibit superlinear behavior with large amounts of data, which is particularly undesirable in applications like data mining.

As we proposed in this thesis, data reduction and meta-learning techniques are used to alleviate the problem of limited memory resource and prolonged execution when large amounts of data are present. We next evaluate the efficiency of our proposed techniques in a parallel and distributed processing environment.

## 9.2 Parallel Evaluation of Hierarchical Meta-learning

The hierarchical meta-learning strategy described in Chapter 6 is designed to be utilized in a parallel and distributed processing environment. Here we analyze and evaluate how the hierarchical meta-learning behaves with leaf classifiers and intermediate tree node classifiers concurrently trained on multiple processors.

### 9.2.1 Notations and Definitions

Before we evaluate our approach in a parallel and distributed environment, our notations and definitions are described as follows:

- $T_S$ = serial execution time.
- $T_P$ = parallel execution time.
- $p$ = number of processors.
- $n$ = input size (number of training examples).
- $W$ = problem size (work) (Kumar *et al.*, 1994), which measures the total number of computational units needed for serial execution. That is, $T_S = W \times t_u$, where $t_u$ is the time spent for a unit of computation. Hence, $T_S \propto W$. For instance, a serial algorithm that is quadratic in input size has a problem size of $n^2$ or $W = n^2$.
- *Speedup* ($S$) is the number of times parallel execution is faster over serial execution with a fixed problem size. That is,

  $$Speedup = \frac{T_S}{T_P}.$$

  For this metric, $T_S$ is usually the time consumption for the fastest serial algorithm, which could be the parallel algorithm running serially.
- *Scaled speedup* (Gustafson, 1988; Kumar & Gupta, 1994) provides a metric for *scalability*. It measures the speedup of a parallel system when the problem size increases linearly with the number of processors.

  $$Scaled\ speedup = \frac{T_S(W \times p)}{T_P(W \times p)}, \tag{9.1}$$

  where $T_S$ and $T_P$ are expressed as functions of problem size. Parallel system with linear or near-linear *scaled speedup* (with respect to $p$, the number of processors) is considered *scalable*. Other scalability metrics can be found in (Kumar & Gupta, 1994).

- *Efficiency* is how fast an algorithm runs, which is characterized by the algorithm's time complexity. (We note that the term *efficiency* can be used another way in parallel computing—it measures how well a parallel algorithm utilizes the available processors and is defined as the ratio of *speedup* to number of processors or $\frac{S}{p}$.)

### 9.2.2 Speedup analysis

For simplicity reasons, we are going to focus our analysis on arbiter trees; similar results can be obtained for combiner trees. Recall that the training set size for an arbiter is restricted to be no larger than the training set size for a leaf classifier (Section 6.1). Hence, in a parallel environment, the amount of computation at each level is approximately the same. Assume the number of data subsets of the initial distribution is $s$ and $s = p$ (the number of parallel processors). (We note that when $s > p$, $s/p$ subsets are processed serially on each of the $p$ processors and a different complexity will result.) Let $d = n/p$ be the size of each data subset, where $n$ is the total number of training examples. Furthermore, assume the learning algorithm takes $O(n^2)$ time (for example, WPEBLS or CN2) in the sequential case. In the parallel case, if we have $p$ processors, there are $\log(p)$ iterations in building the arbiter tree and each takes $O(d^2)$ time. The total time is therefore $O(d^2 \log(p))$, which is

$$O(\frac{n^2 \log(p)}{p^2}) \tag{9.2}$$

For the same parallel algorithm that is run sequentially, there are $2p - 1$ $(p + p/2 + \cdots + 2 + 1)$, or $O(p)$, executions of the algorithm and each takes $O(d^2)$; the total time is therefore $O(d^2 p)$, which is

$$O(\frac{n^2}{p}) \tag{9.3}$$

As a result, a potential $O(\frac{n^2}{p} \times \frac{p^2}{n^2 \log(p)})$ or

$$O(\frac{p}{\log(p)})$$

fold speedup can be achieved. Moreover, if we directly compare the parallel algorithm to the pure serial algorithm, which is $O(n^2)$, the potential speedup is $O(n^2 \times \frac{p^2}{n^2 \log(p)})$ or

$$O(\frac{p^2}{\log(p)})$$

fold, which is superlinear. The standard way of calculating speedup uses the fastest serial algorithm. In our case, the serially run parallel algorithm is asymptotically faster than the pure serial algorithm. Hence, the first speedup analysis provides the proper measure. We include the second analysis as an indication of the speed difference between the parallel approach and the pure sequential approach.

To simplify the previous discussion, we did not take into consideration the classification time to generate the predictions, communication time to send the predictions to one site, and construction time to generate the meta-level training sets. Here, we consider a more detailed analysis, which includes the additional time consumption, but yields, under certain conditions, the same time complexity as before using a simpler analysis.

For the parallel case, when classification time is also considered, at each level $O(d)$ instances have to be classified in parallel on all processors. To generate the arbiters at the first tree level, $O(d)$ instances are classified by the base classifiers. To generate the arbiters at the second tree level, $O(2d)$ instances are classified because a different validation set is used and is classified by both the base classifiers and arbiters at the first tree level. Since there are $\log(p)$ levels, a total of $O(d\frac{(1+\log(p))\log(p)}{2})$ instances are classified. That is the total classification time is $O(d\frac{\log(p)+\log^2(p)}{2})$. Sending $O(d)$ predictions to one site takes $O(d)$ time so the total communication time is $O(d\frac{\log(p)+\log^2(p)}{2})$. $O(d)$ time is needed to construct the meta-level training sets at each level so $O(d\log(p))$ time is need for $\log(p)$ levels. The overall time to build an arbiter tree is the sum of the training time (Equation 9.2) (with a constant $K_1$ for each example), classification time (with a constant $K_2$), communication time (with a constant $K_3$), and construction time (with a constant $K_4$), which is

$$O(K_1 d^2 \log(p) + K_2 d \frac{\log(p) + \log^2(p)}{2} + K_3 d \frac{\log(p) + \log^2(p)}{2} + K_4 d \log(p)).$$

Since $d$ is much larger than $\log(p)$ and $K_1$ (constant for training from an example) is larger than $K_2$, $K_3$, and $K_4$, the first term $K_1 d^2 \log(p)$ is the dominating term. That is, the overall time consumption can be reduced to $O(d^2 \log(p))$, which is $O(\frac{n^2 \log(p)}{p^2})$.

When the parallel case in run serially, only classification time and construction time needs to be included. For generating the arbiters at the first level, $p$ batches of classification are needed. For the second level. $p + \frac{p}{2}$ batches are needed. For the root level, $p + \frac{p}{2} + \cdots + 1$ batches are needed. That is, a total of $\mathrm{O}(p \log(p))$ classification batches are needed. Since each batch takes $O(d)$ time, the total classification time is $O(dp \log(p))$. $O(p)$ meta-level training sets are constructed and each takes $O(d)$ time, hence, the total construction time is $O(dp)$. From above (Equation 9.3), excluding the classification and construction time, the training time is $O(d^2 p)$. Hence, with the additional time included, the total training time (using the constants as before) is

$$O(K_1 d^2 p + K_2 dp \log(p) + K_4 dp)$$

Since $d$ is much larger than $p$ and $\log(p)$, and $K_1$ (constant for training from one example) is larger than $K_2$ and $K_4$, the first term $K_1 d^2 \log(p)$ is the dominating term. That is, the overall time consumption can be reduced to $O(d^2 p)$, which is $O(\frac{n^2}{p})$.

These analyses assume the classification time to generate the arbiter training sets is relatively small compared to the training time. However, this might not the case for some algorithms. Since the number of processors needed for training an arbiter tree is reduced in half at each level and only one processor is used at the root level, the idle processors can be used to classify (Section 6.1) if the base classifiers and arbiters are communicated to other processors. Therefore, training and classifying can be overlapped in execution if we induce more communication overhead. That is, this method would be beneficial if the base classifiers and arbiters are not large. Furthermore, we assume that the processors have equal performance and thus load balancing and other issues in a heterogeneous environment raise interesting issues for future work.

### 9.2.3 Scalability analysis

Now we measure the scalability of our approach using *scaled speedup* (Equation 9.1). From Equation 9.3, the problem size $W$ is $n^2/p$. To calculate the scaled speedup, we first enlarge the problem size to $W \times p$ or $n^2/p \times p$, which is $n^2$. That is, when the parallel case is run serially, it takes

$$O(n^2)$$

time to complete the enlarged problem. Let the enlarged input size be $m$. The enlarged problem size is therefore $m^2/p$. By equating the two expressions for the enlarged problem size, $n^2 = m^2/p$, we arrive at $m = n\sqrt{p}$. From the analysis above (Equation 9.2), by substituting $n$ with $m$ or $n\sqrt{p}$, the parallel time complexity becomes $O(\frac{(n\sqrt{p})^2 \log(p)}{p^2})$, which is

$$O(\frac{n^2 \log(p)}{p}).$$

The scaled speedup is therefore $O(n^2 \times \frac{p}{n^2 \log(p)})$, which is

$$O(\frac{p}{\log(p)}).$$

Although the scaled speedup is sublinear with respect to the number of processors, it is quite close to linear. That is, our approach is quite scalable.

For completeness, we also derive the scaled speedup with respect to the pure serial algorithm. The problem size $W$ is $n^2$ (quadratic learning algorithm). The enlarged problem size is $W \times p$, which is $n^2 p$. That is, the pure serial algorithms takes

$$O(n^2 p)$$

time to complete the enlarged problem. Let the enlarged input size be $m$. The enlarged problem size is therefore $m^2$. By equating the two expressions for the enlarged problem size, $n^2p = m^2$, we arrive at $m = n\sqrt{p}$. From the analysis above (Equation 9.2), by substituting $n$ with $m$ or $n\sqrt{p}$, the parallel time complexity becomes $O(\frac{(n\sqrt{p})^2 \log(p)}{p^2})$, which is

$$O(\frac{n^2 \log(p)}{p}).$$

Therefore, the scaled speedup is $O(n^2p \times \frac{p}{n^2 \log(p)})$, which is

$$O(\frac{p^2}{\log(p)}).$$

Similar to the superlinearity of the regular speedup analysis, the scaled speedup is also superlinear when the parallel algorithm is compared to the pure serial algorithm. Again, we note that the fastest serial case should be used for speedup analysis; the second analysis is presented for completeness.

An alternate scalability metric is *memory-bound scaled speedup* (Sun & Ni, 1993), which measures the increase in possible problem size with increasing number of processors, each with limited available memory. For our approach, this measure is linear since adding one more processor translates to an increase in problem size of one more subset of the training data that fits on one processor. That is, our approach is scalable according to the memory-bound scaled speedup metric. The next section describes our experiments and results on the meta-learning strategies.

### 9.2.4 Empirical simulation

We ran a series of experiments to test our strategies based on the splice junction prediction task. Four different learning algorithms (ID3, CART, WPEBLS, and BAYES) were used. As in experiments with arbiter trees, we varied the number of subsets from 2 to 64 and the equal-size subsets were disjoint with proportional partitioning of classes. Figure 9.5 and 9.6 plot our estimated speedup calculated from measured timing statistics.

However, we measured the CPU time taken to generate each arbiter and approximate the overall CPU time of meta-learning, had we executed the code in a parallel

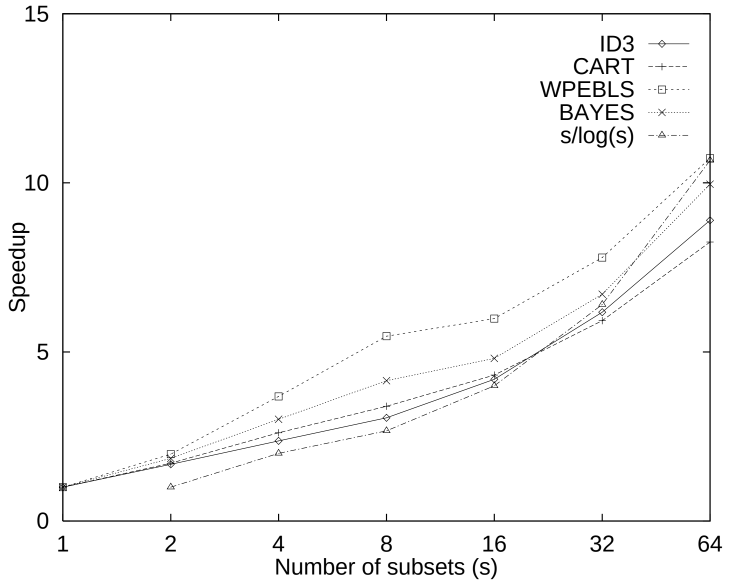


Figure 9.5: Speedup of simulated parallel meta-learning over serial meta-learning.

environment. The approximation is calculated by summing over the longest time needed to generate an arbiter at each level of the arbiter tree. As noted above, the cost of classification needed for selecting examples for the arbiter training sets is not included. Also, the effects of communication and multiple I/O channels on speed are not taken into account, as well as preprocessing such as data partitioning. In addition, since our training set of 2,500+ examples is still relatively small, we duplicated each example ten times in each subset before learning begins. This also has the effect of increasing the size of each arbiter training set by ten. Note that a training set with 25,000+ examples is still a relatively small set, but due to the limitation of the current serial implementation, much larger sets require more computer resources than currently available to us.

In Figure 9.5 we plot the speedup of the parallel meta-learning case (approximated) with respect to the time for meta-learning using only one processor. In Figure 9.6 we plot the speedup of the approximation of the parallel meta-learning case with respect to the time used by the pure sequential algorithm (without meta-learning). The plotted results are from arbiter trees trained with the *different* selection rule and the arbiter training set size limited to the size of the initial training subset

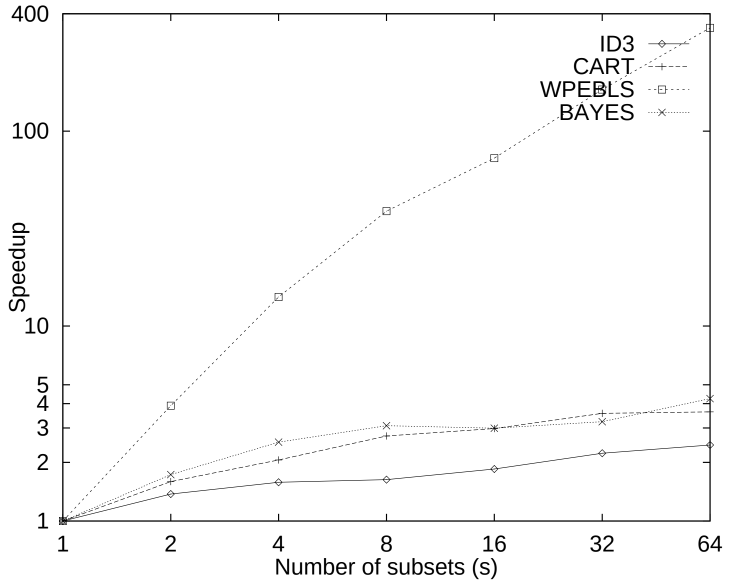


Figure 9.6: Speedup of simulated parallel meta-learning over pure serial learning.

size. All timing statistics were obtained from an Sun IPX workstation.

As shown in Figure 9.5, speedup was observed in all cases as expected. All speedup curves approximate $O(p/\log(p))$, derived in Section 9.2.2. Compared to the pure sequential version of the algorithms (Figure 9.6), our strategies posted small speedup, except in the WPEBLS case, which showed, as expected, superlinear speedup. The small speedup observed in the other three algorithms is mainly due to the relatively small data set we were using (25,000+ training examples) and their low order time complexities (Section 9.1.1). In addition, the overhead of invoking the training and classification processes becomes significant when the data set is small, which is the case in our experiments. Next, we discuss our parallel implementation and empirical experiments on very large data sets.

### 9.2.5 Parallel implementation

The hierarchical meta-learning strategies were implemented on a parallel and distributed platform based on the message-passing model. To satisfy our goal of portability, we chose PVM (Parallel Virtual Machine) (Geist *et al.*, 1993) to provide message

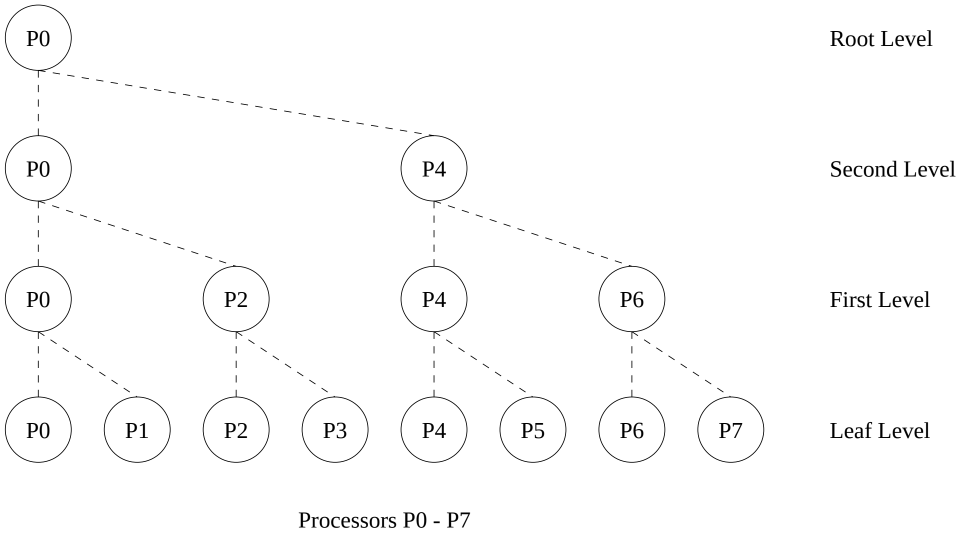


Figure 9.7: Processor allocation for each node in a binary arbiter/combiner tree with 8 leaf nodes.

passing support—PVM supports a common interface for message passing among machines of diverse architectures. The computing platform we used consists of eight HP 9000/735 workstations on a dedicated FDDI (Fiber Distribution Data Interface) network.

Figure 9.7 depicts how the 8 processors (P0-P7) are allocated to a binary arbiter/combiner tree with 8 leaf nodes. At the leaf level, the 8 processors generate 8 base classifiers, which are then used to produce predictions on the *validation set*. At the first tree level, 4 of the 8 processors become parent processors and each of them receives predictions from its 2 respective child processors, one of which is the parent processor itself. The other 4 processors are left idle. Each parent processor then generates the meta-level training set and the meta-level classifier. Then, at the second tree level, 2 of the 4 processors become parent processors. The process is repeated until the meta-classifier at the root is formed.

Because of the hierarchical nature of an arbiter/combiner tree, it is unavoidable to leave half of the active processors more or less idle when each level of the tree is formed. That is, not all the processors are in use at all times. Also, each node in an arbiter/combiner tree is a synchronization point, which reduces parallelism. However, individual subtrees are independent of each other and are built asynchronously.

With 8 processors, we only experimented with binary trees. Higher-order trees would require more processors (for example, a two-level 4-ary tree would need 16 processors). Although we are limited to 8 physical processors, we can always simulate multiple virtual processors on each processor (which is not included in this study).

### 9.2.6 Experiments on the parallel implementation

To reduce the need of transferring large data files across the network from remote file systems, we stored the necessary data for each processor on its local file system. However, some of the local file systems are small on our 8-processor system. Hence, the size of the data files is limited by the smallest local file system among the 8 processors. Currently, we can run parallel experiments on all 8 processors with up 5 million examples.

We varied the number of examples from 1,000 to 5 million. The time results reported here measure the elapsed time between the start and the end of the learning process, which includes the communication overhead among processors. Data preparation and distribution prior to learning are not included in our time measurements. Each plotted point is the average of five runs on different data sets produced by our artificial data generator using different random seeds. We ran experiments using different combinations of learning algorithms (ID3, CART, BAYES, and CN2) and hierarchical meta-learning schemes (arbiter tree, class-combiner tree, and class-attribute-combiner tree).

Figure 9.8 plots the elapsed learning time against the number of training examples. Both axes are in log scale. Each plot in the figure shows the results from a learning algorithm used in the three different hierarchical meta-learning schemes. The plots for CART and CN2 stop at 100,000 examples because CART started to core dump

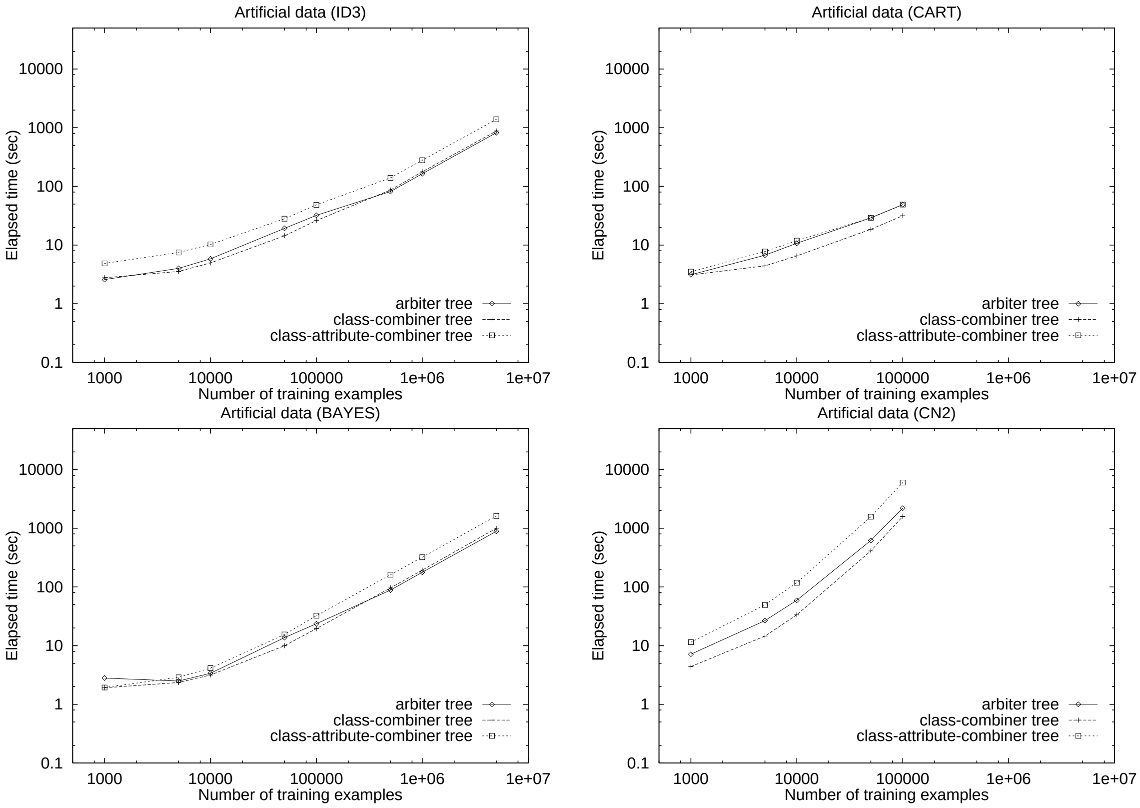


Figure 9.8: Training time for parallel meta-learning on 8 processors (grouped by learning algorithms).

and CN2 ran out of memory with 500,000 or more examples.

As expected, the class-attribute-combiner tree scheme takes more processing time the class-combiner tree scheme since the meta-level training set in the first scheme includes the original attributes of the training examples. The arbiter-tree scheme seems to be between the two combiner tree schemes. Although the arbiter-tree scheme usually create fewer examples in the meta-level training set, each example has all the attributes from the original training data.

We group the timing results by hierarchical meta-learning schemes in Figure 9.9. Both axes are in log scale. Each plot shows the results from a hierarchical meta-learning scheme using the four different learning algorithms. We observe that, as expected, CN2 takes longer than the other three learning algorithms in processing the training data. Furthermore, CN2's time consumption increases more rapidly than

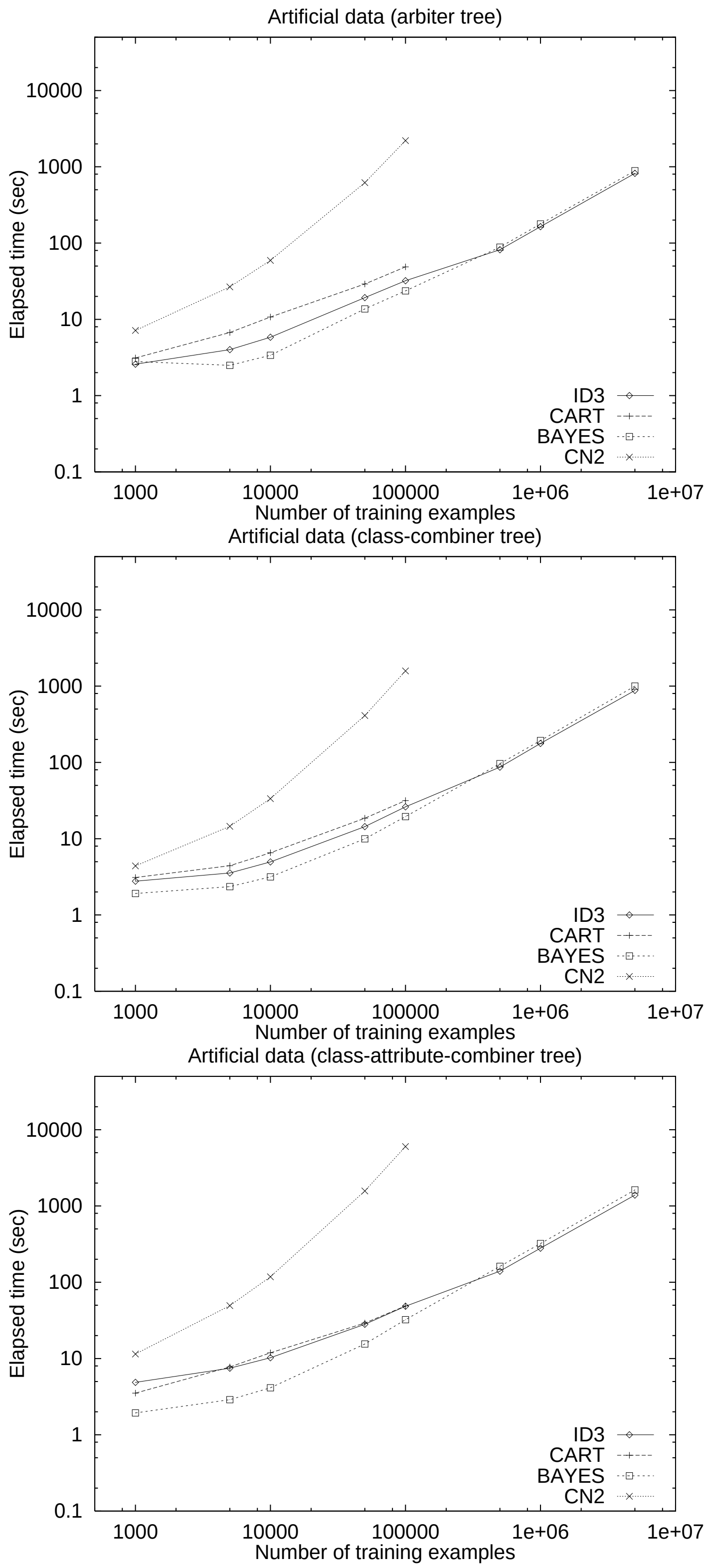


Figure 9.9: Training time for parallel meta-learning on 8 processors (grouped by schemes).

the others'. These results are consistent with the empirical timing results from serial execution.

Figure 9.10 shows the speedup of our parallel meta-learning implementation on 8 processors over pure serial learning. We note that superlinear speedup (more than 8 in our case) is possible because we are using the pure serial learning case for comparison. Experiments were not performed on running the parallel meta-learning schemes serially (because as we see later, some pure serial algorithms are quite efficient with relatively large data sets). Each graph presents results from a hierarchical meta-learning scheme using the four different learning algorithms. Because CART core dumped with different number of examples in the serial and parallel cases, only the speedup for training from 1,000 to 10,000 examples can be calculated. Since CN2 ran out of memory with more than 10,000 examples, speedup for CN2 can only be computed for processing 1,000 to 10,000 examples.

Substantial speedup is observed for CN2 with as few as 5,000 examples. CART shows some, but not sizeable, speedup at the few data points we can gather. However, for ID3 and BAYES, we observe that the parallel meta-learning schemes are not worthwhile until the training set contains more than one million examples. The parallel case is not faster than the serial case with up to about 1 million examples (speedup $\leq 1$). With 5 million examples, ID3 shows some speedup while BAYES achieves substantial speedup. Because the class-combiner tree scheme takes less time than the other two schemes, larger speedup is obtained.

CN2's superlinear time requirement leads to large speedup in a parallel environment. ID3 and BAYES are quite efficient in processing up to about 1 million examples in a serial environment. That is, parallel meta-learning is greatly beneficial to superlinear learning algorithms like CN2 in terms of speed. In terms of scalability, parallel meta-learning is beneficial to ID3 and BAYES when the memory requirement for processing large amounts of data is getting close to or exceeds the available resources on one processor (sizable speedup was obtained with 5 million examples).

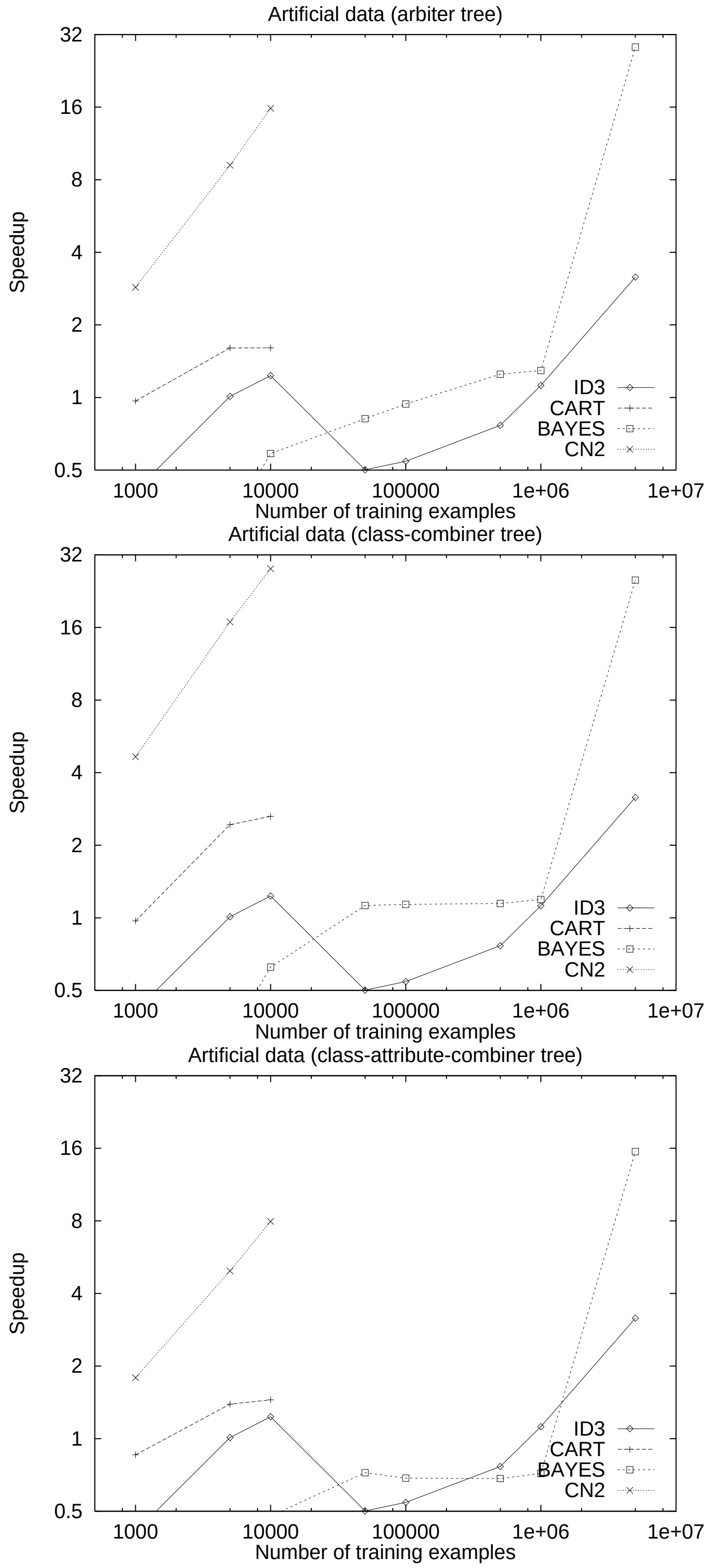


Figure 9.10: Speedup of parallel meta-learning on 8 processors over serial learning.

Our current parallel implementation is used to demonstrate the utility of hierarchical meta-learning in a parallel and distributed environment. Refinements of the implementation is left for future work. For example, the tradeoff between classification time (during training) and communication time for exchanging classifiers and meta-classifiers was not studied. We mentioned earlier that communicating the classifiers among the processors can make use of the idle processors for classification while the active ones are used for training. This method is advantageous if the communication time is small compared to the classification time. Because we are limited to 8 processors, some of our results will improve when more processors are available—higher degree of parallelism, higher order trees, larger data sets... Furthermore, if we relax our portability goal, using customized platform-dependent message-passing routines rather than portable ones reduces communication overhead among processors and improves overall time performance.

## 9.3 Summary

The theoretical time complexity of five learning algorithms was analyzed. WPEBLS and CN2 clearly exhibit superlinear complexity with respect to the number of training examples. Although ID3, CART, and BAYES show linear complexity with respect to training set size, practical time performance indicates superlinear behavior. That is, all five algorithms exhibit superlinear time performance when very large training sets are encountered. In fact, at a certain point, the learning algorithms ran out of memory and terminated abnormally. Quadratic approximations estimate that CN2 would take 3.5 months, and WPEBLS 6.4 months, to process one million training examples if they were given sufficient memory resources. Certainly, these estimates will change with more powerful machines in the future.

Theoretical speedup and scalability of our hierarchical meta-learning schemes were analyzed. Empirical results from our parallel implementation show that CN2 (and probably WPEBLS) benefits greatly from our methods. Other superlinear-time learning algorithms like genetic algorithms and neural networks can also benefit much from our methods. Parallel hierarchical meta-learning is more advantages for ID3 and BAYES when their memory requirement for processing large amounts of data is getting close to or exceeds the available resources on one processor.

# Chapter 10

# Multistrategy Meta-Learning

The objective here is to improve prediction accuracy by exploring the diversity of multiple learning algorithms 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 learners. The meta-learner may employ the same algorithm as one of the base learners or a completely distinct algorithm. The training set for the meta-learner (meta-level training data) varies according to the strategies described in Section 3.4 and is quite different from the original training set. We experimented with three types of meta-learning strategies (*combiner*, *arbiter*, and *hybrid*). Each base-learner generates a *base classifier* and the meta-learner generates a *meta-classifier*. Note that the meta-learner does not aim at picking the "best" base classifier; instead it tries to combine the classifiers. That is, the prediction accuracy of the overall system is not limited to the most accurate base classifier. It is our intention to generate an overall system that outperforms the underlying base classifiers.

We first study, in Section 10.1, multistrategy meta-learning on unpartitioned data, where base classifiers are trained on the whole data set. We then explore, in Section 10.2, multistrategy meta-learning on partitioned data, where base classifiers are trained on disjoint data subsets. Lastly, in Section 10.3, we compare our *combiner* strategy with the related *stack generalization* proposed by Wolpert (1992). The com-

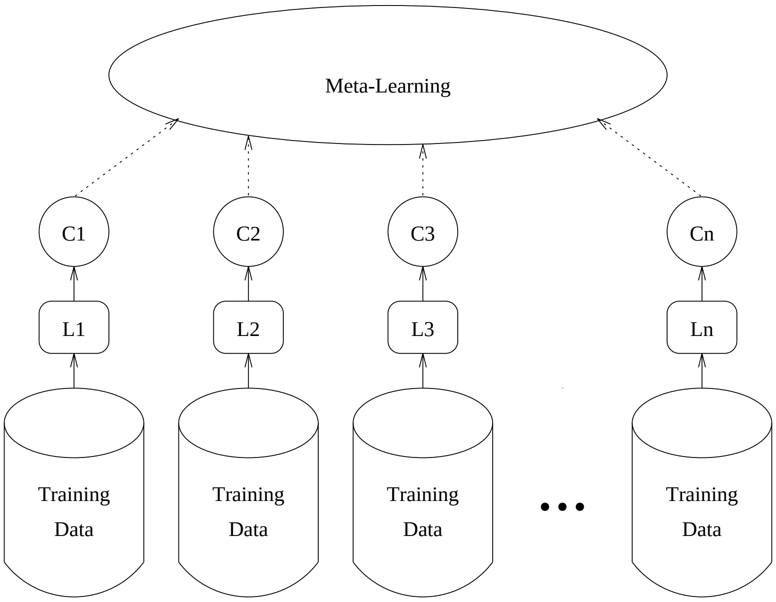


Figure 10.1: Multistrategy Meta-learning on Unpartitioned Data

parison is based on whole (unpartitioned) data sets.

## 10.1 Multistrategy Meta-learning on Unpartitioned Data

Here we investigate multistrategy meta-learning on whole (unpartitioned) data sets (Chan & Stolfo, 1993a). Each of the base learners is provided with the entire training set of raw data. That is, a different learning algorithm is applied to the entire data set to generate the base classifiers and then learn a meta-classifier to integrate the base ones. Figure 10.1 depicts this approach. This is a common approach adopted by much of the work in using multiple algorithms to improve overall prediction accuracy. However, we try to learn correlations rather than using different variations of voting.

The predictions used in the training set of the meta-learner were generated by a two-fold cross validation scheme. The training set is first split in two halves. Each of the three base classifiers were trained on the first half and the second half is used to generate predictions. Similarly, each base classifier is trained on the second half and the first half is used to generate predictions. The predictions from the two halves are merged and then used in constructing the training set for the meta-learner. The objective is to mimic the behavior of the learned classifiers when unseen instances are classified. That is, the meta-learner is trained on predictions of unseen instances in the training set. The base learners are also trained on the entire training set to generate base classifiers, which are then used with the learned meta-classifier in the classification process.

### 10.1.1 Experiments

We performed experiments on the different schemes for the combiner, arbiter, and hybrid strategies. Four inductive learning algorithms: ID3, CART, WPEBLS and BAYES and two data sets: secondary structures (SS) and splice junctions (SJ) were used in the experiments. Different combinations of three base and one meta-learner are explored on the two data sets and the results are presented in Tables 10.1 through 10.4. Each table has two subtables and each subtable presents results from a different combination of base learners. Results for the two data sets with single-strategy classifiers are displayed in Table 10.5. In addition, we experimented with a windowing scheme used in Zhang's (1992) work, which is specific to the secondary structure data. This scheme is similar to the *class-combiner* 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 *class-window-combiner* (or *class-window-combiner* in the tables).

Furthermore, several 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

Table 10.1: Summary of prediction accuracy (%) for secondary structures (SS) (Part 1).

| Base learners: ID3, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 56.3+ | 55.8+ | 55.7+ | 55.1 |
| class-attribute-combiner | 60.3+ | 55.4 | 48.7 | 48.5 |
| binary-class-combiner | 55.6+ | 56.6+ | 56.6+ | 52.7 |
| class-window-combiner | 56.9+ | 54.5 | 49.9 | 50.6 |
| different-arbiter | 60.7+ | 56.4+ | 56.1+ | 53.3 |
| different-incorrect-arbiter | 59.8+ | 56.4+ | 53.9 | 52.4 |
| different-class-attribute-hybrid | 60.5+ | 56.5+ | 55.7+ | 54.4 |
| different-incorrect-class-attribute-hybrid | 59.1+ | 56.4+ | 54.1 | 53.1 |
| vote | 56.3+ | | | |
| freq | 56.5+ | | | |
| vote-b | 57.1+ | | | |

| Base learners: BAYES, ID3 & CART | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 61.4 | 62.1 | 62.1 | 57.3 |
| class-attribute-combiner | 62.1 | 61.0 | 51.0 | 50.4 |
| binary-class-combiner | 61.1 | 61.9 | 61.7 | 54.4 |
| class-window-combiner | 60.7 | 60.1 | 52.5 | 53.3 |
| different-arbiter | 62.2+* | 57.2 | 57.6 | 57.6 |
| different-incorrect-arbiter | 60.8 | 58.3 | 57.7 | 56.6 |
| different-class-attribute-hybrid | 62.1 | 61.5 | 58.9 | 58.5 |
| different-incorrect-class-attribute-hybrid | 60.4 | 58.6 | 58.0 | 56.7 |
| vote | 60.6 | | | |
| freq | 62.1 | | | |
| vote-b | 60.9 | | | |

Keys:
$*$ better than the best single strategy
+ better than the best base classifier

Table 10.2: Summary of prediction accuracy (%) for secondary structures (SS) (Part 2).

| Base learners: BAYES, ID3 & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 60.4 | 62.1 | 62.1 | 55.9 |
| class-attribute-combiner | 61.9 | 60.6 | 51.0 | 52.6 |
| binary-class-combiner | 60.5 | 61.8 | 61.8 | 55.2 |
| class-window-combiner | 59.9 | 59.5 | 51.4 | 52.9 |
| different-arbiter | 62.0 | 57.4 | 57.3 | 55.7 |
| different-incorrect-arbiter | 60.8 | 59.0 | 56.6 | 54.4 |
| different-class-attribute-hybrid | 61.6 | 60.7 | 57.9 | 56.8 |
| different-incorrect-class-attribute-hybrid | 60.8 | 59.3 | 56.1 | 54.6 |
| vote | 59.3 | | | |
| freq | 62.2+* | | | |
| vote-b | 59.3 | | | |

| Base learners: BAYES, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 60.7 | 62.1 | 61.8 | 56.8 |
| class-attribute-combiner | 61.4 | 60.5 | 50.4 | 50.9 |
| binary-class-combiner | 60.5 | 61.7 | 61.3 | 52.6 |
| class-window-combiner | 59.7 | 57.9 | 52.6 | 54.2 |
| different-arbiter | 62.0 | 58.1 | 57.4 | 54.6 |
| different-incorrect-arbiter | 61.4 | 58.6 | 56.8 | 52.0 |
| different-class-attribute-hybrid | 61.1 | 60.3 | 58.0 | 56.1 |
| different-incorrect-class-attribute-hybrid | 59.4 | 59.1 | 59.1 | 59.2 |
| vote | 59.6 | | | |
| freq | 60.7 | | | |
| vote-b | 60.9 | | | |

in the training set[1]. 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. *vote-b* is a simple voting scheme applied to the predictions from the binary base classifiers.

We note that we did not repeat the experiments over many different training and

[1] *freq* was suggested by Wolpert (1993).

Table 10.3: Summary of prediction accuracy (%) for splice junctions (SJ) (Part 1).

| Base learners: ID3, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 95.1+ | 94.8 | 94.8 | 72.7 |
| class-attribute-combiner | 96.6+* | 95.0+ | 95.9+ | 95.5+ |
| binary-class-combiner | 95.1+ | 94.4 | 94.4 | 74.1 |
| different-arbiter | 96.4+ | 94.7 | 95.5+ | 95.3+ |
| different-incorrect-arbiter | 96.6+* | 95.8+ | 95.8+ | 95.5+ |
| different-class-attribute-hybrid | 96.1+ | 94.5 | 94.8 | 95.3+ |
| different-incorrect-class-attribute-hybrid | 95.9+ | 94.2 | 95.0+ | 95.0+ |
| vote | 95.0+ | | | |
| freq | 95.0+ | | | |
| vote-b | 95.1+ | | | |

| Base learners: BAYES, ID3 & CART | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 95.6 | 96.6+* | 95.3 | 74.3 |
| class-attribute-combiner | 97.2+* | 96.4 | 95.0 | 96.9+* |
| binary-class-combiner | 95.8 | 96.2 | 96.2 | 75.2 |
| different-arbiter | 96.9+* | 95.3 | 95.6 | 95.9 |
| different-incorrect-arbiter | 96.9+* | 95.5 | 95.9 | 96.2 |
| different-class-attribute-hybrid | 95.8 | 94.5 | 95.1 | 95.9 |
| different-incorrect-class-attribute-hybrid | 95.3 | 94.2 | 94.8 | 95.5 |
| vote | 95.6+ | | | |
| freq | 95.9+ | | | |
| vote-b | 95.6 | | | |

Keys:
* better than the best single strategy
+ better than the best base classifier

test sets so our results presented here may or may not be due to statistical variation.

### 10.1.2 Results

There are two ways to analyze the results. First, we look at whether the employment of a meta-learner improves accuracy with respect to the underlying three base

Table 10.4: Summary of prediction accuracy (%) for splice junctions (SJ) (Part 2).

| Base learners: BAYES, ID3 & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 97.2+* | 96.9+* | 96.9+* | 73.7 |
| class-attribute-combiner | 97.6+* | 96.9+* | 95.9 | 96.1 |
| binary-class-combiner | 96.6+* | 96.1 | 96.1 | 75.6 |
| different-arbiter | 96.2 | 95.6 | 95.8 | 96.1 |
| different-incorrect-arbiter | 96.1 | 95.3 | 96.6+* | 95.6 |
| different-class-attribute-hybrid | 95.6 | 94.4 | 94.5 | 95.0 |
| different-incorrect-class-attribute-hybrid | 95.1 | 94.4 | 94.2 | 95.0 |
| vote | 97.0+* | | | |
| freq | 97.0+* | | | |
| vote-b | 96.1 | | | |

| Base learners: BAYES, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 97.0+* | 96.6+* | 95.3 | 73.7 |
| class-attribute-combiner | 97.2+* | 96.4 | 95.3 | 96.2 |
| binary-class-combiner | 96.1 | 96.2 | 96.2 | 76.2 |
| different-arbiter | 96.7+* | 95.0 | 96.2 | 96.2 |
| different-incorrect-arbiter | 96.6+* | 95.0 | 95.8 | 96.2 |
| different-class-attribute-hybrid | 94.8 | 94.2 | 94.7 | 94.5 |
| different-incorrect-class-attribute-hybrid | 94.7 | 94.2 | 94.5 | 94.8 |
| vote | 97.0+* | | | |
| freq | 96.9+* | | | |
| vote-b | 96.2 | | | |

Table 10.5: Prediction accuracy (%) of single-strategy classifiers

| Data Set/Algorithm | BAYES | ID3 | CART | WPEBLS |
|---|---|---|---|---|
| Secondary Structure (SS) | 62.1 | 55.4 | 50.8 | 48.1 |
| Splice Junction (SJ) | 96.4 | 93.9 | 94.8 | 94.4 |

classifiers. (The presence of an improvement is denoted by a '+' in the tables.) For both sets of data, improvements were always achieved when BAYES was used as the meta-learner and the other three learning algorithms we used as the base learners, regardless of the meta-learning strategies.

Now let us consider various combinations of meta-learner and strategies with any of base learning algorithms. For the SJ data, a higher or equal accuracy was consistently attained when BAYES was the meta-learner in the *class-attribute-combiner* strategy. Similarly, higher accuracy was attained when ID3 served as the meta-learner in the *class-combiner* and *class-attribute-combiner* strategies, regardless of the base learners used. Improvements were also observed in the *vote* and *freq* strategies. For the SS data, none of the various combinations of meta-learners and strategies attained a consistent improvement in overall accuracy.

Next, we consider whether the use of meta-learning achieves higher accuracy than the most accurate single-strategy learner (BAYES). (The presence of an improvement is denoted by a '*' in the tables.) For the SJ data, an improvement was consistently achieved when BAYES served as the meta-learner in the *class-attribute-combiner* strategy, regardless of the base learners used. In fact, when the base learners were BAYES, ID3, and CART, the overall accuracy was the highest obtained. For the SS data, almost all the results did not outperform BAYES as a single-strategy learner.

In general, the *combiner* strategies performed more effectively than the *arbiter* and *hybrid* strategies. To our surprise, the hybrid schemes did not improve the arbiter strategies. In addition, Zhang's (1992) *class-window-combiner* strategy for the SS data did not improve accuracy with the base and meta-learners used here. His study employed a neural net algorithm and different Bayesian and nearest-neighbor learners than those reported here.

The two data sets chosen represent two different kinds of data sets: one is difficult to learn (SS) (50+% accuracy) and the other is easy to learn (SJ) (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 comparable. This can be attributed to the quality of predictions from the base classifiers for the two data sets. Consider the statistics we gathered from the predictions for the test set from classifiers trained by BAYES, ID3, and WPEBLS (other combinations of learners have similar statistics). In the SJ data set, on 89% of the instances all predictions from the three learned classifiers were correct, on 7% two predictions were correct, on 2% only one, and on 1% none (all incorrect). In the SS data set, on 29% of the instances all three predictions were correct, on 33% only two, on 18% only one, and on 20% none. 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.

### 10.1.3 Discussion

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 combining 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 crucial fact may not be exhibited in larger learning tasks with massive amounts of data.)

Among the combiner schemes, the *class-attribute-combiner* scheme generally performed more effectively than the others. This might be due to the additional information (attribute vectors) present in the training examples, suggesting that information from the predictions alone is not sufficient to achieve higher prediction accuracy. To our surprise, the *binary-class-combiner* scheme did not perform more effectively than the *class-combiner* scheme. We postulate that more specialized binary classifiers would provide more precise information for the meta-learner. However, that was not the case in our experiments.

We also postulate 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 and its probabilistic means of combining evidence. Our empirical results indeed show that BAYES is a more effective meta-learner.

## 10.2 Multistrategy Meta-learning on Partitioned Data

Here we use meta-learning to combine different learners to improve prediction accuracy and speed (Chan & Stolfo, 1993c). The dual objectives 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 to the base-classifiers generated from the different subsets. That is, instead of utilizing the same learning algorithm to train the base classifiers (as in previous chapters), different algorithms are employed. Figure 10.2 illustrates this approach.

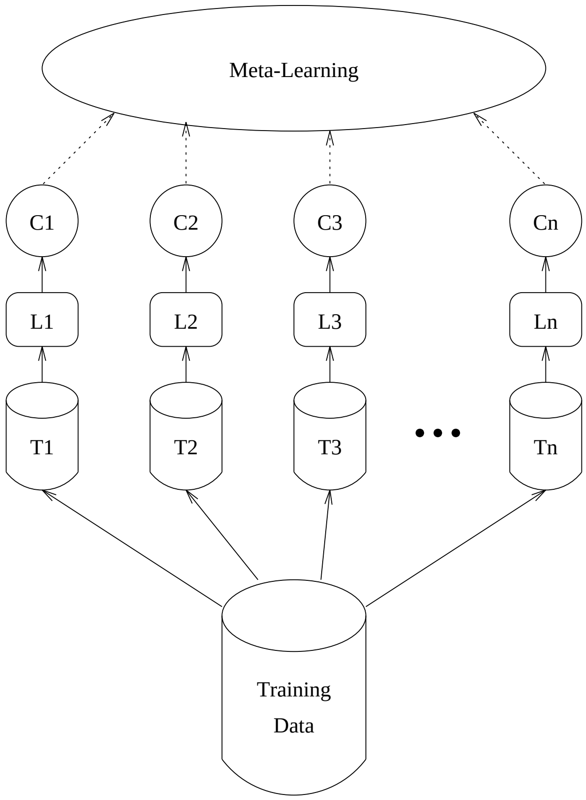


Figure 10.2: Multistrategy meta-learning on partitioned data.

### 10.2.1 Issues

Load balancing is essential in minimizing the overall training time due to the variance in completion times of different algorithms. However, 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 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 (as discussed in the previous section)? 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.

### 10.2.2 Experiments

Our approach was empirically evaluated with four inductive learning algorithms (ID3, CART, WPEBLS and BAYES) and two data sets (splice junctions and secondary structures).

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.

We performed experiments on the different schemes for the combiner, arbiter, and

Table 10.6: Summary of prediction accuracy (%) for the secondary structure data (Part 1).

| Base learners: ID3, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 53.1 | 51.8 | 51.8 | 53.8 |
| class-attribute-combiner | 58.0+ | 50.8 | 48.5 | 49.1 |
| binary-class-combiner | 52.8 | 52.4 | 52.2 | 52.2 |
| different-arbiter | 55.2+ | 54.4+ | 54.1+ | 54.3+ |
| different-incorrect-arbiter | 55.2+ | 53.7 | 54.5+ | 54.1+ |
| different-class-attribute-hybrid | 55.5+ | 54.9+ | 54.4+ | 54.3+ |
| different-incorrect-class-attribute-hybrid | 55.0 | 54.1+ | 54.1+ | 54.0+ |
| class-window-combiner | 56.5+ | 54.7+ | 51.8 | 53.4 |
| vote | 54.7+ | | | |
| freq | 51.9 | | | |

| Base learners: BAYES, ID3 & CART | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 57.9 | 53.8 | 54.5 | 54.9 |
| class-attribute-combiner | 58.8 | 54.5 | 50.9 | 52.2 |
| binary-class-combiner | 57.9 | 57.4 | 57.2 | 54.8 |
| different-arbiter | 59.5 | 58.8 | 58.5 | 58.1 |
| different-incorrect-arbiter | 59.0 | 58.6 | 58.5 | 58.4 |
| different-class-attribute-hybrid | 59.1 | 58.8 | 58.6 | 58.6 |
| different-incorrect-class-attribute-hybrid | 58.9 | 58.8 | 58.6 | 58.9 |
| class-window-combiner | 58.0 | 55.8 | 53.0 | 53.6 |
| vote | 58.5 | | | |
| freq | 57.1 | | | |

Keys:
+ better than the 3 base classifiers (subsets)
$*$ better than all 4 classifiers (subsets)

hybrid strategies. Different combinations of three base and one meta-learner were explored on the two data sets and the results are presented in Tables 10.6 through 10.9. Each table has two 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. Results

Table 10.7: Summary of prediction accuracy (%) for the secondary structure data (Part 2).

| Base learners: BAYES, ID3 & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 54.7 | 53.8 | 53.4 | 56.5 |
| class-attribute-combiner | 59.7 | 53.0 | 49.0 | 50.3 |
| binary-class-combiner | 57.4 | 54.8 | 54.9 | 54.4 |
| different-arbiter | 58.0 | 57.6 | 57.2 | 57.5 |
| different-incorrect-arbiter | 57.9 | 57.4 | 57.5 | 57.4 |
| different-class-attribute-hybrid | 57.8 | 57.8 | 57.6 | 57.7 |
| different-incorrect-class-attribute-hybrid | 58.0 | 57.8 | 57.6 | 57.5 |
| class-window-combiner | 59.4 | 56.8 | 53.6 | 53.4 |
| vote | 57.8 | | | |
| freq | 54.0 | | | |

| Base learners: BAYES, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 55.7 | 54.0 | 53.1 | 54.9 |
| class-attribute-combiner | 59.3 | 53.0 | 49.7 | 49.8 |
| binary-class-combiner | 55.3 | 52.4 | 53.9 | 54.3 |
| different-arbiter | 57.2 | 56.8 | 56.8 | 56.6 |
| different-incorrect-arbiter | 57.0 | 56.5 | 56.6 | 56.3 |
| different-class-attribute-hybrid | 57.3 | 56.9 | 56.8 | 56.7 |
| different-incorrect-class-attribute-hybrid | 57.0 | 56.4 | 56.4 | 56.5 |
| class-window-combiner | 59.2 | 55.3 | 53.4 | 55.2 |
| vote | 57.2 | | | |
| freq | 54.0 | | | |

for the two data sets with single-strategy classifiers are displayed in Table 10.10. 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 *class-combiner* scheme. 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 *class-window-combiner* in the tables.

Table 10.8: Summary of prediction accuracy (%) for the splice junction data (Part 1).

| Base learners: ID3, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 95.1+ | 95.0+ | 95.0+ | 71.5 |
| class-attribute-combiner | 96.2+* | 94.8+ | 94.8+ | 95.0+ |
| binary-class-combiner | 95.8+* | 96.1+* | 95.6+ | 75.5 |
| different-arbiter | 95.1+ | 95.0+ | 95.1+ | 95.1+ |
| different-incorrect-arbiter | 95.1+ | 95.1+ | 95.1+ | 95.1+ |
| different-class-attribute-hybrid | 95.1+ | 95.0+ | 95.1+ | 95.1+ |
| different-incorrect-class-attribute-hybrid | 95.1+ | 95.1+ | 95.1+ | 95.1+ |
| vote | 95.1+* | | | |
| freq | 95.6+* | | | |

| Base learners: BAYES, ID3 & CART | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 94.5 | 95.9+* | 95.5 | 73.7 |
| class-attribute-combiner | 96.7+*! | 96.1+* | 95.5 | 95.1 |
| binary-class-combiner | 95.5 | 95.6 | 95.0 | 76.0 |
| different-arbiter | 95.0 | 95.0 | 95.0 | 95.0 |
| different-incorrect-arbiter | 95.0 | 94.5 | 94.5 | 95.0 |
| different-class-attribute-hybrid | 95.0 | 95.0 | 95.0 | 94.8 |
| different-incorrect-class-attribute-hybrid | 95.0 | 94.8 | 94.8 | 94.8 |
| vote | 95.0 | | | |
| freq | 95.8+* | | | |

Keys:
+ better than the 3 base classifiers (subsets)
* better than all 4 classifiers (subsets)
! better than all 4 classifiers (full set)

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

[2] *freq* was suggested by Wolpert (1993).

Table 10.9: Summary of prediction accuracy (%) for the splice junction data (Part 2).

| Base learners: BAYES, ID3 & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 95.8+* | 95.5 | 95.1 | 72.7 |
| class-attribute-combiner | 97.0+*! | 95.5 | 95.1 | 95.6 |
| binary-class-combiner | 96.2+* | 95.1 | 95.0 | 76.5 |
| different-arbiter | 95.9+* | 95.9+* | 95.9+* | 95.5 |
| different-incorrect-arbiter | 95.9+* | 95.9+* | 95.9+* | 95.9+* |
| different-class-attribute-hybrid | 95.9+* | 95.9+* | 95.9+* | 95.5+* |
| different-incorrect-class-attribute-hybrid | 95.9+* | 95.8+* | 95.8+* | 95.8+* |
| vote | 95.9+* | | | |
| freq | 95.3 | | | |

| Base learners: BAYES, CART & WPEBLS | | | | |
|---|---|---|---|---|
| | Meta-learner | | | |
| Scheme | BAYES | ID3 | CART | WPEBLS |
| class-combiner | 96.7+*! | 95.9+* | 96.1+* | 72.4 |
| class-attribute-combiner | 97.2+*! | 95.9+* | 95.9+* | 95.8+* |
| binary-class-combiner | 95.9+* | 94.8 | 95.6 | 93.6 |
| different-arbiter | 96.9+*! | 96.9+*! | 96.9+*! | 96.7+*! |
| different-incorrect-arbiter | 96.9+*! | 96.9+*! | 96.6+*! | 96.7+*! |
| different-class-attribute-hybrid | 96.9+*! | 96.9+*! | 96.9+*! | 96.7+*! |
| different-incorrect-class-attribute-hybrid | 96.9+*! | 96.7+*! | 96.7+*! | 96.7+*! |
| vote | 96.9+*! | | | |
| freq | 96.7+*! | | | |

Table 10.10: Single-strategy 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 |

for $m$ classes and $c$ classifiers), *freq* predicts the most frequent correct class in the training data.

We note that we did not repeat the experiments over many different training and test sets so our results presented here may or may not be due to statistical variation.

### 10.2.3 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 *class-attribute-combiner* 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, *class-attribute-combiner* 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, *class-attribute-combiner* is the more effective scheme and BAYES is the more successful meta-learner. Therefore, it reinforces our conjecture that combining results are more effective than arbitrating among them and predictions alone may not be enough for meta-learning. Compared to the results obtained and described in the previous section, 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) *class-window-combiner* 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.

As mentioned in the previous section, the combiner schemes generally performed more effectively than the arbiter or hybrid schemes. This suggests that combining the base predictions 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 *class-attribute-combiner* scheme performed more effectively than the others. This might be due to the additional information (attribute vectors) present in the training examples, suggesting that information from the predictions alone is not sufficient to achieve higher prediction accuracy. To our surprise, the *binary-class-combiner* scheme did not perform more effectively than the *class-combiner* 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.

## 10.3 Comparing Multistrategy Combiner with Stacked Generalization

Wolpert's (1992) *stacked generalization* is very similar to our *combiner* strategy (the *class-combiner* scheme in particular) with multiple learning algorithms for training base-classifiers. The difference is how the meta-level (level-1 in Wolpert's terms) training set is generated. In both methods cross-validation partitioning is used to generate the meta-level training set.

$k$-fold cross-validation partitioning involves making $k$ pairs of training and test sets. To generate the $k$ pairs, the original training set $T$ is first partitioned into $k$ subsets: $T_1$, $T_2$, ..., $T_k$. Then each of these subsets becomes a test set of a pair and the union of the remaining subsets becomes the training set of the pair. For example, when $T_2$ is the test set of a pair, the union of $T_1$, $T_3$, $T_4$, ..., $T_k$ forms the training set of the pair. A classifier is learned from the training set of each pair and is applied to the test set of the pair. That is, from $k$ pairs of training and test sets, $k$ sets of predictions are obtained. Note that in each pair, the training and test sets are disjoint. That is, the predictions from each pair are made on "unseen" data that are not involved in training.

The predictions from the $k$ pairs provides an approximation of how predictions are made on unseen data by a classifier learned from the whole original training set. In both our combiner strategy and stacked generalization, these predictions constitute part of the meta-level training set.

Our combiner strategy uses 2-fold cross-validation partitioning, whereas stacked generalization uses $n$-fold, where $n$ is the number of training examples. That is, combiner uses two pairs of training and test sets to generate the meta-level training set, whereas stacked generalization uses $n$ pairs. When $k$ is $n$, the test set in each pair has only one example and the training set has $n-1$ examples. Intuitively, $n$-fold cross-validation partitioning provides a closer approximation, and hence more accurate meta-level training data, than $2-fold$. However, $n$-fold is clearly much more computationally expensive than 2-fold. The tradeoffs are further discussed through the following experimental results (Fan *et al.*, 1996).

### 10.3.1 Experiments

Three inductive learning algorithms (ID3, CART, and BAYES) and two molecular biology sequence analysis data sets (secondary structures and splice junctions) were used in our experiments. Results for combiner and stacked generalization were obtained from 5-fold cross validation runs. Different combinations of three base and one meta-learner were applied to the two data sets and the results are shown in the Table 10.11. Table 10.12 shows the prediction accuracy of individual algorithms for the two data sets.

### 10.3.2 Results

There are several ways to look at the results. First, we see if the employment of both combiner and stacked generalization improves prediction accuracy with respect to the underlying three base classifiers. As shown in Table 10.12, the accuracy of single-strategy classifiers (ID3 and CART) on secondary structures is around 57%. The accuracy of both combiner and stacked generalization is about 61%. That is an improvement of 4% or 173 (out of 4325) more examples correctly classified. In the

Table 10.11: Prediction accuracy of combiner and stack generalization for secondary structure and splice junction data

| Results from secondary structures | | | | | | |
|---|---|---|---|---|---|---|
| Base learners: ID3, CART & BAYES | | | | | | |
| | Meta-learner in Stacked Generalization | | | Meta-learner in Combiner | | |
| accuracy in percentage(%) | | | | | | |
| Fold | ID3(%) | CART(%) | BAYES(%) | ID3(%) | CART(%) | BAYES(%) |
| 1 | 59.6 | 60.0 | 61.4 | 60.0 | 60.0 | 61.1 |
| 2 | 62.3 | 59.7 | 61.3 | 62.6 | 62.6 | 61.5 |
| 3 | 62.7 | 62.7 | 62.5 | 62.8 | 62.7 | 62.1 |
| 4 | 60.3 | 60.3 | 60.0 | 60.7 | 60.8 | 60.8 |
| 5 | 59.7 | 60.1 | 59.2 | 60.9 | 60.9 | 60.7 |
| $\mu$ | 60.9 | 60.6 | 60.9 | 61.4 | 61.4 | 61.2 |
| $\sigma$ | 1.5 | 1.2 | 1.3 | 1.2 | 1.2 | 0.6 |

| Results from splice junctions | | | | | | |
|---|---|---|---|---|---|---|
| Base learners: ID3, CART & BAYES | | | | | | |
| | Meta-learner in Stacked Generalization | | | Meta-learner in Combiner | | |
| accuracy in percentage (%) | | | | | | |
| Fold | ID3(%) | CART(%) | BAYES(%) | ID3(%) | CART(%) | BAYES(%) |
| 1 | 95.6 | 95.8 | 94.8 | 95.0 | 95.1 | 94.8 |
| 2 | 96.1 | 96.4 | 95.8 | 96.4 | 96.9 | 95.6 |
| 3 | 95.5 | 95.5 | 96.2 | 95.8 | 95.8 | 96.4 |
| 4 | 96.7 | 96.2 | 96.2 | 96.6 | 96.2 | 96.1 |
| 5 | 95.0 | 95.6 | 93.9 | 95.1 | 95.6 | 93.9 |
| $\mu$ | 95.8 | 95.9 | 95.4 | 95.8 | 95.9 | 95.4 |
| $\sigma$ | 0.7 | 0.4 | 1.0 | 0.7 | 0.7 | 1.0 |

splice junction data set (also shown in Table 10.12), the improvement of combiner and stacked generalization (with average accuracy of around 95.5%) over single classifiers (ID3 and CART, average accuracy is around 92%) is 3.5%. These improvements are significant (more than one standard deviation). We also notice that in both the SS and SJ data sets, the best single-strategy classifier is BAYES. The accuracy of both combiner and stacked generalization is close to that of BAYES. For the two particular data sets under study and the particular combination of learning algorithms, we could pick BAYES instead of using combiner or stacked generalization. However, if we do not know apriori which learning algorithm generates the most accurate classifier,

Table 10.12: Prediction accuracy single-strategy classifiers on secondary structures and splice junction data

| Single Classifier Accuracy on SS | | | |
|---|---|---|---|
| | Learner | | |
| Fold | ID3(%) | CART(%) | BAYES(%) |
| 1 | 56.1 | 57.6 | 60.0 |
| 2 | 52.9 | 57.1 | 62.5 |
| 3 | 54.1 | 57.2 | 62.7 |
| 4 | 52.2 | 57.1 | 60.8 |
| 5 | 56.6 | 56.9 | 61.4 |
| $\mu$ | 56.4 | 57.2 | 61.5 |
| $\sigma$ | 1.9 | 0.3 | 1.2 |

| Single Classifier Accuracy on SJ | | | |
|---|---|---|---|
| | Learner | | |
| Fold | ID3(%) | CART(%) | BAYES(%) |
| 1 | 90.1 | 93.4 | 95.1 |
| 2 | 91.2 | 84.8 | 96.9 |
| 3 | 90.4 | 94.0 | 95.3 |
| 4 | 89.7 | 94.5 | 95.3 |
| 5 | 88.4 | 93.9 | 94.5 |
| $\mu$ | 90.0 | 94.1 | 95.4 |
| $\sigma$ | 1.0 | 0.6 | 0.9 |

combiner and stack generalization has demonstrated that they can achieve at least the same level of accuracy as the most accurate underlying learning algorithm.

Second, we see which of combiner and stacked generalization has a higher accuracy improvement. The two methods are comparable, but combiner performs a little better. For the secondary structure data set, combiner has an average of 0.5% higher accuracy than stacked generalization 1% standard deviation, so their accuracy are essentially the same. For the splice junction data set, their accuracy levels are nearly the same.

Third, we examine the correlation between these two methods. We have applied a simple approach to determine the number of examples that:

- they both correctly label,

Table 10.13: Summary of correlation analysis between combiner and stacked generalization.

Results on SS

| Meta-learner: ID3 | | | | | | Meta-learner: CART | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Fold | SC | CI | IC | SI | DI | Fold | SC | CI | IC | SI | DI |
| 1 | 2560 | 17 | 35 | 1699 | 14 | 1 | 2595 | 0 | 0 | 1730 | 0 |
| 2 | 2677 | 18 | 30 | 1594 | 6 | 2 | 2397 | 185 | 308 | 1325 | 110 |
| 3 | 2619 | 94 | 95 | 1493 | 24 | 3 | 2713 | 0 | 0 | 1612 | 0 |
| 4 | 2310 | 298 | 314 | 1280 | 123 | 4 | 2533 | 77 | 97 | 1579 | 39 |
| 5 | 2376 | 205 | 256 | 1395 | 93 | 5 | 2460 | 141 | 172 | 1486 | 66 |

| Meta-learner: BAYES | | | | | |
|---|---|---|---|---|---|
| Fold | SC | CI | IC | SI | DI |
| 1 | 2402 | 252 | 241 | 1336 | 94 |
| 2 | 2452 | 201 | 206 | 1394 | 72 |
| 3 | 2407 | 295 | 278 | 1253 | 92 |
| 4 | 2309 | 288 | 321 | 1270 | 137 |
| 5 | 2277 | 283 | 349 | 1272 | 144 |

Results on SJ

| Meta-learner: ID3 | | | | | | Meta-learner: CART | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Fold | SC | CI | IC | SI | DI | Fold | SC | CI | IC | SI | DI |
| 1 | 604 | 6 | 2 | 24 | 2 | 1 | 604 | 7 | 3 | 24 | 0 |
| 2 | 610 | 3 | 5 | 20 | 0 | 2 | 611 | 4 | 7 | 16 | 0 |
| 3 | 605 | 4 | 6 | 22 | 1 | 3 | 605 | 4 | 6 | 23 | 0 |
| 4 | 610 | 7 | 6 | 15 | 0 | 4 | 612 | 2 | 2 | 22 | 0 |
| 5 | 595 | 11 | 12 | 20 | 0 | 5 | 609 | 1 | 1 | 27 | 0 |

| Meta-learner: BAYES | | | | | |
|---|---|---|---|---|---|
| Fold | SC | CI | IC | SI | DI |
| 1 | 605 | 0 | 0 | 33 | 0 |
| 2 | 610 | 1 | 0 | 27 | 0 |
| 3 | 614 | 0 | 1 | 22 | 1 |
| 4 | 613 | 1 | 0 | 24 | 0 |
| 5 | 599 | 0 | 0 | 39 | 0 |

*Notes:*

- SC: Same Correct, or both of stacked generalization and combiner predict correctly.
  CI: Correct/Incorrect, or stacked generalization correctly predicts but combiner incorrectly labels
  IC: Incorrect/Correct, or stacked generalization incorrectly labels, while combiner correctly labels
  SI: Same Incorrect, or stacked generalization and combiner make the same wrong labels
  DI: Different Incorrect, both are incorrect, but their answers are different
- Results are shown in number of predictions on five test sets.

- one method can label correctly, but the other cannot,
- they both give the same wrong answer, and
- they both give wrong but different answers.

Correlation analysis results are shown in Table 10.13. The results indicate that the two methods are very close to each other. In most of the cases they either both give the correct answer or both give the same wrong answer, indicating that they both have effectively learned the same knowledge.

Fourth, we display the training cost of both methods in Table 10.14. As expected, the difference in their training cost is huge. For the SS data set, while combiner spent no more than 7 minutes to learn, stacked generalization spent about 9 days. For the SJ data set, combiner took half a minute to learn, but stacked generalization took nearly 6 hours and a half. There are orders of magnitude difference in efficiency performance for comparable accuracy gain.

Finally, we examine the meta-level (or level-1 in Wolpert's terms) training set. Each meta-level training example is composed of the predictions generated by the base classifiers using the *class-combiner* scheme. The meta-level training set can be divided into components, each of which originated from a base classifier. That is, in the meta-level training set, the predictions generated by a base classifier constitutes a component, which we call *meta-component training data*. In our case the meta-level training set has predictions from base classifiers generated by ID3, CART, and BAYES. That is, we have three sets of meta-component training data.

The accuracy of meta-component training data is measured by comparing them to the correct labels of the original training examples. The closer the accuracy of meta-component training data is to the accuracy of a single-strategy classifier, the more accurate it approximates the behavior of the base classifier. Intuitively, stacked generalization produces a closer approximation than combiner. (Recall that, in generating the meta-component training data in stacked generalization, the prediction

Table 10.14: Training time (CPU seconds) for combiner and stacked generalization.

| Results on SS | | | |
|---|---|---|---|
| | Combiner | | |
| Fold | ID3 | CART | BAYES |
| 1 | 336.9 | 340.4 | 336.8 |
| 2 | 340.7 | 344.4 | 340.7 |
| 3 | 346.1 | 349.6 | 346.3 |
| 4 | 340.3 | 344.5 | 340.3 |
| 5 | 351.0 | 354.7 | 351.1 |
| $\mu$ | 343.0 | 346.7 | 343.0 |

| Results for SJ | | | | | | |
|---|---|---|---|---|---|---|
| | Stacked Generalization | | | Combiner | | |
| Fold | ID3 | CART | BAYES | ID3 | CART | BAYES |
| 1 | 23720.4 | +0.50 | -0.16 | 30.6 | 30.93 | 29.9 |
| 2 | 23865.7 | +0.50 | -0.16 | 29.8 | 30.3 | 30.0 |
| 3 | 23442.9 | +0.50 | -0.16 | 31.6 | 31.2 | 30.1 |
| 4 | 23784.3 | +0.50 | -0.16 | 30.5 | 30.2 | 30.0 |
| 5 | 23805.9 | +0.50 | -0.16 | 29.9 | 30.7 | 29.8 |
| $\mu$ | 23723.8 | 23724.3 | 23723.7 | 30.5 | 30.7 | 30.0 |

*Notes:*

- The training cost of stacked generalization for the SS data set is huge, we did not have exclusive use of the machines to measure it. The method we used to estimate its cost is to measure the time used to generate 1000 meta-level training data. The estimate we give below is the result made by multiplying the actual time for 1000 items by 17.3 (1,7300/1,000=17.3) plus the time to learn the 3 base classifiers and meta classifier. Stacked generalization, therefore, requires 771,760 seconds (nearly 9 days) for the SS data set.
- For the SJ data set, we could not measure the training cost for stacked generalization for different meta-learner(ID3, CART and BAYES) from start, that would require a lot of computer resources. We accurately measured the time cost of stacked generalization using ID3 as the meta-learner from start, estimated the case for CART as the meta-learner by adjusting the difference ID3 versus CART to learn the meta-classifier. The difference was that it took 0.50 seconds more for CART to learn the meta-level training data than ID3. Using the same method, we estimated the training cost of having BAYES as the meta-learner.
- Also, only CPU time of learning is measured.

Table 10.15: Summary of accuracy of meta component data for Secondary Structure and Splice Junction(%)

| Results on SS | | | | | | |
|---|---|---|---|---|---|---|
| | Base learner in Stacked Generalization | | | Base learner in Combiner | | |
| Fold | ID3(%) | CART(%) | BAYES(%) | ID3(%) | CART(%) | BAYES(%) |
| 1 | 54.0 | 56.9 | 62.6 | 49.3 | 55.3 | 61.4 |
| 2 | 55.3 | 56.8 | 62.3 | 49.8 | 55.5 | 60.2 |
| 3 | 53.7 | 57.5 | 61.9 | 50.0 | 56.7 | 60.2 |
| 4 | 55.9 | 57.5 | 62.2 | 52.4 | 55.5 | 61.2 |
| 5 | 57.3 | 57.3 | 62.2 | 53.2 | 55.5 | 61.0 |
| $\mu$ | 55.2 | 57.9 | 62.2 | 50.9 | 55.7 | 60.8 |
| $\sigma$ | 1.5 | 0.5 | 0.3 | 1.74 | 0.6 | 0.6 |

| Results on SJ | | | | | | |
|---|---|---|---|---|---|---|
| | Base learner in Stacked Generalization | | | Base learner in Combiner | | |
| Fold | ID3(%) | CART(%) | BAYES(%) | ID3(%) | CART(%) | BAYES(%) |
| 1 | 90.3 | 94.2 | 95.5 | 88.6 | 94.4 | 95.2 |
| 2 | 90.8 | 94.3 | 95.2 | 86.2 | 91.3 | 94.4 |
| 3 | 91.5 | 94.3 | 95.7 | 87.5 | 93.5 | 94.8 |
| 4 | 90.8 | 94.1 | 95.4 | 88.4 | 94.1 | 95.7 |
| 5 | 91.7 | 94.4 | 95.8 | 88.7 | 94.4 | 95.4 |
| $\mu$ | 91.0 | 94.3 | 95.5 | 87.9 | 93.5 | 95.1 |
| $\sigma$ | 0.6 | 0.1 | 0.2 | 1.05 | 1.3 | 0.5 |

of an example is made by a classifier trained from the remaining $n-1$ examples.)

By comparing Table 10.15 with the single-strategy classifiers' accuracy in Table 10.12, we can see that the accuracy of the meta-component training data of stacked generalization is closer to the accuracy of a single-strategy classifier than that of combiner, which means it closely mimics the behavior of single-strategy classifiers. The accuracy of the stacked generalization's meta-component training data and the accuracy of single-strategy classifier differs by no more than 1%, but the accuracy of the combiner's meta-component training data and the accuracy of single classifier differs by as much as 5%. However, this seems not to boost the accuracy obtained from stacked generalization more than the accuracy obtained from combiner. That is, closer approximation did not seem to produce more accurate meta-classifiers. This is an interesting finding.

### 10.3.3 Discussion

It may be intuitive to guess that the accuracy boost of stacked generalization will be more than that of combiner. However, the empirical results we obtained do not support this intuition. There may be three possible explanations for this, all requiring further study:

1. One argument is that the correlation of meta-component training data actually decides the overall accuracy boost. The accuracy of meta-component training data that reflect the behavior of the base classifiers may not be the decisive factor. We performed correlation analysis at the meta-classifier level.

2. Another explanation is that the complexity (or the difficulty to learn) of the meta-level training data may also be important. The meta-level training data of stacked generalization may actually reflect the behavior of the base classifiers, but the relationship among the base classifiers is very subtle and very difficult for the meta-learner to learn effectively, so the overall accuracy did not improve as much as we would hope. Specialized learning algorithms may be developed for the sole purpose of learning how to integrate classifiers.

3. Conflicts in the meta data may contribute to the problem. As an example, look at Figure 10.3. In the first example, there is a data element that ID3 labels 1, CART labels 2, BAYES labels 0, but the actual answer is 1. In the second case, ID3, CART and BAYES give the same predictions as in the first example, but this time the true answer is 2. This means that the mapping from attribute vector to label is not one to one. This would be very difficult for any algorithm to learn the correct answer; even a human may not be able to do it correctly. This kind of example represents *conflicts* in the training data. Decision tree algorithms, like ID3 and CART, can be very sensitive to conflicts. BAYES is better equipped to handle conflicts because of its probabilistic nature. We need to see how many conflicts there are in the meta-level training data of both combiner and stacked generalization. One approach that can reduce (but may

| Correct Class | ID3 | CART | BAYES |
|---|---|---|---|
| 1 | 1 | 2 | 0 |
| 2 | 1 | 2 | 0 |

Figure 10.3: Conflicts in meta-level training data

not eliminate) the number of conflicts is to introduce more base learners. This method is not applicable if the number of base classifiers cannot be increased. However, the *binary-class-combiner* scheme in Section 3.4.1 can be used to generate multiple base classifiers for each base learner. Another approach is to include the attribute vector of each example in the meta-level training data as described in the *class-attribute-combiner* scheme (Section 3.4.1). If all the attribute vectors are unique, the meta-level training set will be free of conflicts.

Earlier we introduced the idea of $k$-fold meta-learning. Our results indicate that the accuracy boost of $k$=2 (combiner) and $k$=$n$ (stacked generalization) are similar. So is $k$ arbitrary or is there a $k$ that will lead to significant maximal accuracy gain? How much is this gain? Do we pay for what we get? If the accuracy boost with different $k$ is almost equivalent, $k$=2 or combiner is obviously the preferred choice– accurate and cheap. Breiman (1996b) studied the effects of changing $k$ to the accuracy of stacking regression (real-value) estimators. He found that $k = 10$ achieves comparable accuracy as $k = n$. His results may or may not apply to classifiers (discrete estimators).

## 10.4 Summary

We also studied meta-learning with the use of multiple learning algorithms to improve the overall predictive accuracy. Our empirical results indicate that multistrategy meta-learning produced slightly higher accuracy than the most accurate underlying learning algorithm, but the improvement is not statistically significant.

However, meta-learning is usually at least as accurate as the best base learning algorithm. Often times, one does not know apriori which learning algorithm can generate the most accurate classifier without extensive experimentation. Multistrategy meta-learning provides a mechanism to avoid the extra work and generates a meta-classifier that is at least as effective as the best classifier.

From the comparison of multistrategy class-combiner and stacked generalization, we discover that they achieved the same level of accuracy even though $n$-fold cross-validation (CV) partitioning used in stacked generalization provides a closer base-classifiers' approximation than 2-fold CV partitioning used in combiner. However, the cost of $n$-fold CV partitioning is much higher than 2-fold CV partitioning. Moreover, correlation analysis indicates that both methods learned similar concepts. Hence, combiner compares favorably to stack generalization.

# Chapter 11

# Conclusion

With the rapid advance in computer networking technology, more and more data will be accessible with the touch of a few key stokes, mouse clicks, or even a few uttered words, hand gestures, eye movements, or brain waves. Analyzing and gaining knowledge from this massive collection of information is an important and increasingly difficult task. Algorithms are limited by their time/space complexity and computers are limited by their hardware resources. Although processor speed and memory capacity increase at an amazing pace, data generated and gathered by more powerful computers grow at an even faster pace. Consequently, for instance, on the world wide web, information is so abundant that search engines were developed to help us locate information we seek. Recently, "meta (super)" search engines, like MetaCrawler (Selberg & Etzioni, 1996), are emerging that locate information by searching a number of search engines.

In this thesis we attempt to address the problem of efficiently and accurately analyzing massive amounts of data using inductive learning algorithms. Our proposed *meta-learning* approach and its diverse specific techniques have been systematically evaluated and compared. In Section 11.1 we summarize our findings from this thesis investigation. Possible future research directions are discussed in Section 11.2.

## 11.1 Results and Contributions

We proposed *meta-learning* as an unified approach for integrating multiple learning processes or algorithms. This approach encompasses the use of learning algorithms to learn how to integrate results from multiple learning systems efficiently and accurately. Meta-learning is intended to be scalable (by data reduction partitioning), extensible (algorithm-independent), and portable (architecture-independent). That is, our goal is to devise a general mechanism that can be used for the wide variety of learning algorithms and computer architectures. This led us to focus our attention on integrating predictions from classifiers and coarse-grain parallelism. We demonstrated that meta-learning can be used to improve speed and accuracy for a wide range of inductive learning algorithms.

We identified three meta-learning strategies: *combiner*, *arbiter*, and *hybrid*. The *combiner* strategy tries to learn the relationship and correlations among the base classifiers. The *arbiter* strategy, however, attempts to learn from instances that are *confusing* to the base classifiers. The hybrid strategy seeks to synergistically integrate the combiner and arbiter strategies. Specific schemes within these strategies were developed and detailed, most notably the *class-combiner*, *class-attribute-combiner*, and *different-arbiter* schemes. A substantial number of experiments were performed to systematically evaluate these schemes under diverse circumstances with different collections of constituent learning algorithms and tasks.

Our empirical results indicate that a simple classifier learned from a sample randomly selected from the original data set could not achieve the same level of accuracy as the classifier trained from the entire original data set (the *global classifier*). That is, to achieve the global classifier's level of accuracy, we need more data than a small random sample and integration of classifiers learned for disjoint subsets could be beneficial. We systematically compared our meta-learning schemes with common voting-based and Bayesian techniques in the literature and the results show that our arbiter scheme outperformed the others. Although the integration techniques yielded significantly higher accuracy than a classifier learned from a random subset, the global classifier's level of accuracy was not always achieved. To raise the effectiveness of our approach, we partially replicated data across the subsets. Unexpectedly, improvement was generally not observed. However, this result demonstrates that the subsets can remain disjoint to allow the highest degree of parallelism.

Thus, a more sophisticated *hierarchical* approach was devised. Two strategies were developed: *arbiter tree* and *combiner tree*. Different classifiers are learned in a *bottom-up* tree fashion. Empirical results indicate that hierarchical meta-learning could usually achieve the same level of accuracy as the global classifier. No degradation in accuracy was always achieved when the training sets at each level were allowed to double in size. For some of the trees generated by the *class-attribute-combiner* scheme, to our surprise, a significantly higher accuracy was achieved. This further demonstrates the viability of our hierarchical meta-learning approach.

We also investigated the different aspects of arbiter trees. Results indicate that lower-order trees were more effective and accurate than higher-order ones. This seems mainly attributed to the increase in the number of opportunities in correcting the base classifiers since there are more levels in the lower order trees to filter and compose good training data. Proportional class partitioning in the base-level training sets yielded more accurate trees than non-proportional partitioning. When the meta-level training set size at each level of the tree was unbounded, accuracy could always be maintained and only about 30% of the entire data set was needed at any time. Proportional class partitioning reduced the percentage to around 10%. That is, a site can process a larger learning task (about 10 times in the domain we studied) without increasing memory resources. At the leaf level, pairing base classifiers that disagree the most could also reduce the percentage. Resolving disagreements at the leaf level, rather than piling them higher in the tree, seems to be the contributing factor.

Data can be distributed across remote sites belonging to diverse organizations. These organizations might be reluctant to share "raw data" due to proprietary or confidentiality reasons. However, they might be willing to share "black-box" models. In our case, the black-boxes are encoded classifiers, whose content is not revealed. Meta-learning techniques were applied to improve a local classifier by importing remote black-box classifiers. Our results show that meta-learning can significantly improve the accuracy of a local classifier. We also studied the effects of data overlap among sites. In many cases the degree of overlap did not affect the amount of accuracy improvement for the local classifier. In other cases additional accuracy gain was observed.

We defined four metrics (*diversity*, *coverage*, *correlated error*, and *specialty*) for characterizing the base classifiers and explored the effects of these characteristics on the behavior of various integrating schemes. From our results, larger accuracy improvement can be achieved by more *diverse* base classifiers with higher *coverage* and fewer *correlated errors*. For integrating schemes that recognize relationships among the base classifiers, more *specialized* base classifiers can result in larger improvement in accuracy. Analyses on the arbiter strategy show that when the base classifiers are less accurate, the arbiter needs to be built more carefully.

The theoretical time complexity of five learning algorithms were analyzed. WPEBLS and CN2 clearly exhibit superlinear complexity with respect to the number of training examples. Although ID3, CART, and BAYES show linear complexity with respect to training set size, practical time performance indicates non-linear behavior. That is, all five algorithms exhibit non-linear time performance when very large training sets are encountered. In fact, at a certain point, the learning algorithms ran out of memory and terminated abnormally. Quadratic approximations estimate that CN2 would take 3.5 months, and WPEBLS 6.4 months, to process one million training examples if they were given sufficient memory resources.

Results from our parallel implementation of the hierarchical meta-learning schemes show that CN2 (and probably WPEBLS) benefits greatly from our methods. Other non-linear-time learning algorithms like genetic algorithms and neural networks can also benefit much from our methods. Parallel hierarchical meta-learning is more advantages for ID3 and BAYES when their memory requirement for processing large amounts of data is getting close to or exceeds the available resources on one processor.

We also studied meta-learning with the use of multiple learning algorithms to improve the overall predictive accuracy. Our empirical results indicate that multistrategy meta-learning produced slightly higher accuracy than the most accurate underlying learning algorithm, but the improvement is not statistically significant. However, meta-learning is usually at least as accurate as the best base learning algorithm. Often times, one does not know a-priori which learning algorithm can generate the most accurate classifier without extensive experimentation. Multistrategy meta-learning provides a mechanism to avoid the extra work and generates a meta-classifier that is at least as effective as the best classifier.

From the comparison of multistrategy class-combiner and stacked generalization, we discover that they achieved the same level of accuracy even though $n$-fold cross-validation (CV) partitioning used in stacked generalization provides a closer base-classifiers' approximation than 2-fold CV partitioning used in combiner. However, the cost of $n$-fold CV partitioning is much higher than 2-fold CV partitioning. Moreover, correlation analysis indicates that both methods learned similar concepts. Hence, combiner compares favorably to stack generalization.

## 11.2 Research Directions

Here we discuss some possible research directions based on this thesis work.

The hierarchical meta-learned tree structures are rather complicated and probably difficult for human inspection. Simplifying the structures without significantly degrading the overall accuracy would be beneficial. One idea is to measure the similarity among base classifiers and prune those that are closely related. The pruning process can be performed in a hill-climbing manner, where related classifiers are removed one by one until the overall accuracy is significantly reduced.

In this thesis the learning algorithms used for meta-learning are "off-the-shelf" algorithms and are the same as the base learning algorithms. More specialized meta-level attributes and algorithms can be devised. Learning algorithms that search M-of-N (Murphy & Pazzani, 1991) and other counting-related concepts might be useful in locating effective combining rules. Constructive induction techniques (Matheus & Rendell, 1989; Rendell, 1990) could also be beneficial in creating potentially relevant attributes. Moreover, learning algorithms that can incorporate weighted or probabilistic predictions from the base classifiers would produce more effective combining rules.

More diverse learning algorithms (for instance, genetic algorithms and neural networks) and learning tasks can be enlisted for larger-scale empirical evaluation, which will probably further increase the generality of our results obtained in this thesis. Since we could not, regrettably, secure a massive "real-world" data set, an artificial data generator was used to generate arbitrarily large data sets for our scaling experiments. Demonstrating similar results on a massive non-artificial data set would be an interesting addition.

To gain a deeper understanding of the reasons why meta-learning works, more sophisticated analysis tools are needed. With our current analysis tools, it is not clear when a particular meta-learning strategy performs better than another. Furthermore, a theoretical foundation like the hypothesis boosting work by Schapire (1990) would be a substantial contribution. We note that theoretical learning models (PAC (Valiant, 1984) for example) represent a class of algorithms that might not be close to the actual learning algorithms used in practice. However, work on bridging theory and practice is emerging—Dietterich et al (1996) applied the weak learning framework, introduced by Schapire (1990), to understand C4.5 (Quinlan, 1993). Moreover, recent statistical work on bias-variation decomposition, for example (Kong & Dietterich, 1995), provides some insights on the source of errors for integrating multiple learned models. Similar approaches can help explain the behavior of our meta-learning strategies.

For meta-learning on partitioned data, most of the results were obtained from using only a single learning algorithm. Employing multiple different algorithms increases the diversity of base classifiers and might improve the overall accuracy.

In parallel meta-learning, a study on the tradeoff between classification time (during training) and communication time for exchanging classifiers and meta-classifiers can be fruitful. We mentioned earlier that communicating the classifiers among the processors can make use of the idle processors for classification while the active ones are used for training. This method is advantageous if the communication time is small compared to the classification time. Because we are limited to 8 processors, some of our results will probably improve when more processors are available—higher degree of parallelism, higher order trees, larger data sets... Studies in a heterogeneous computing environment would introduce interesting load balancing issues that are not addressed in our current study in a homogeneous computing environment.

Learning algorithms and meta-learning techniques can be encapsulated in agents that can be sent across information networks. Using the new network-based architecture-independent language Java (Arnold & Gosling, 1996), learning and meta-learning agents can roam around the internet with ease. Databases on the network can be reached by these agents and the learned classifiers can then be encapsulated in agents to perform further analyses.

On a rather unrelated note, it is my belief that electronic computers might hit a ceiling in terms of gaining intelligence. Modern computers are still very much controlled by their creators and execute prescribed steps. Biochemical computers might be the source of future "real" intelligent computing. Adleman (1994) successfully demonstrated a rudimentary molecular computer. A Directed Hamiltonian Path problem was encoded in DNA sequences. Through biochemical interactions, solutions to the problem were searched via trillions of molecules in a massively parallel manner.

The solutions were then extracted through DNA analyses. Soon afterwards, Lipton (1995) showed how to use DNA to solve more general combinatorial problems. This might be the dawn of a new computing era.

## 11.3 Final Remarks

Partly because of this work, we identified a community of researchers and developers, and organized a well-participated workshop on integrating multiple learned models (briefly described in Section 2.5). Research in this area might become more refined in the future because of a focused forum for exchanging ideas and peer reviews.

Recently, ARPA awarded Prof. Stolfo and his colleagues a research grant to study the techniques described here in a fraud detection application. Learning and meta-learning agents written in Java will travel to different database sites to learn characteristics of fraudulent transactions. This indicates some degree of confidence and maturity in this area of research. It is our hope that our techniques and others' will be much improved in the not so distant future.

*Learning never ceases, nor should it.*

# Bibliography

Abramson, N. (1963). *Information Theory and Coding.* New York, NY: McGraw-Hill.

Adleman, L. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266, 1021–1024.

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

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine learning*, 6, 37–66.

Ali, K. & Pazzani, M. (1996). Error reduction through learning multiple descriptions. *Machine Learning.* to appear.

Arnold, K. & Gosling, J. (1996). *The Java Programming Language.* Reading, MA: Addison-Wesley.

Booker, L., Goldberg, D., & Holland, J. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, 40, 235–282.

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

Boswell, R. (1990). *Manual for CN2 version 6.1.* Turing Institiure. Int. Doc. IND: TI/MLT/4.0T/RAB/1.2.

Bratko, I. & Muggleton, S. (1995). Applications of inductive logic programming. *Commmunications of the ACM*, 38(11), 65–70.

Breiman, L. (1994). *Bagging Predictors.* (Technical Report 421), Berkeley, CA: Dept. of Statistics, Univ. of California.

Breiman, L. (1996a). *Bias, variance, and arcing classifiers.* (technical report, Berkeley, CA: Statistics Dept., U. of California.

Breiman, L. (1996b). Stacked regressions. *Machine Learning*, 24, 41–48.

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

Brodley, C. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, 20, 63–94.

Brodley, C. & Lane, T. (1996). Creating and exploiting coverage and diversity. *Work. Notes AAAI-96 Workshop Integrating Multiple Learned Models* (pp. 8–14).

Brunk, C. & Pazzani, M. (1995). A lexically based semantic bias for theory revision. *Proc. 12th Intl. Conf. Mach. Learning* (pp. 81–89).

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

Carbonell, J. (1989). Introduction: Paradigms for machine learning. *Artificial Intelligence*, 40, 1–9.

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

Catlett, J. (1992). Peepholing: Choosing attributes efficiently for megainduction. *Proc. Ninth Intl. Conf. Machine Learning* (pp. 49–54).

Chan, P. (1988). *A critical review of CN2: A polythetic classifier system.* (Technical Report CS-88-09 (Master's paper)), Nashville, TN: Department of Computer Science, Vanderbilt University.

Chan, P. (1991). *Machine Learning in Molecular Biology Sequence Analysis.* (Technical Report CUCS-041-91), New York, NY: Department of Computer Science, Columbia University.

Chan, P. & Stolfo, S. (1993a). Experiments on multistrategy learning by meta-learning. *Proc. Second Intl. Conf. Information and Knowledge Management* (pp. 314–323).

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

Chan, P. & Stolfo, S. (1993c). Toward multistrategy parallel and distributed learning in sequence analysis. *Proc. First Intl. Conf. Intelligent Systems for Molecular Biology* (pp. 65–73).

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

Chan, P. & Stolfo, S. (1994). *Toward Scalable and Parallel Learning: A Case Study in Splice Junction Prediction.* (Technical Report CUCS-032-94), New York, NY: Department of Computer Science, Columbia University. (Presented at the ML94 Workshop on Machine Learning and Molecular Biology).

Chan, P. & Stolfo, S. (1995a). A comparative evaluation of voting and meta-learning on partitioned data. *Proc. Twelfth Intl. Conf. Machine Learning* (pp. 90–98).

Chan, P. & Stolfo, S. (1995b). Learning arbiter and combiner trees from partitioned data for scaling machine learning. *Proc. Intl. Conf. Knowledge Discovery and Data Mining* (pp. 39–44).

Chan, P. & Stolfo, S. (1996a). Scaling learning by meta-learning over disjoint and partially replicated data. *Proc. Ninth Florida AI Research Symposium* (pp. 151–155).

Chan, P. & Stolfo, S. (1996b). Sharing learned models among remote database partitions by local meta-learning. *Proc. Second Intl. Conf. Knowledge Discovery and Data Mining* (pp. 2–7).

P. Chan, S. Stolfo, & D. Wolpert (Eds.) (1996). *Working Notes for the AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, Portland, OR.

Chesseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). Autoclass: A bayesian classification system. *Proc. Fifth Intl. Conf. Machine Learning* (pp. 54–64).

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3, 261–285.

Clearwater, S. & Provost, F. (1990). RL4: A tool for knowledge-based induction. *Proc. Second Intl. IEEE Conf. Tools for AI* (pp. 24–30). IEEE CS Press.

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

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

Craven, M. & Shavlik, J. (1994). Machine learning approaches to gene recognition. *IEEE Expert*, 9(2), 2–10.

Danyluk, A. (1991). Gemini: An integration of analytical and empirical learning. *Proc. First Intl. Work. Multistrategy Learning* (pp. 191–206).

DeJong, K. (1988). Learning with genetic algorithms: An overview. *Machine Learning*, 3, 121–138.

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

Dietterich, T. & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. *Proc. AAAI-91* (pp. 572–577). AAAI Press.

Dietterich, T. & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *J. AI Research*, 2, 263–286.

Dietterich, T., Kearns, M., & Mansour, Y. (1996). Applying the weak learning framework to understand and improve C4.5. *Proc. Thirteenth Intl. Conf. Machine Learning* (pp. 96–104).

Dietterich, T. & Michalski, R. (1983). A comparative review of selected methods for learning from examples. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, (pp. 331–363). Morgan Kaufmann.

Domingos, P. (1995). Rule induction and instance-based learning: A unified approach. *Proc. IJCAI-95* (pp. 1226–1232).

Domingos, P. (1996). Using partitioning to speed up specific-to-general rule induction. *Work. Notes AAAI-96 Workshop Integrating Multiple Learned Models* (pp. 29–34).

Drucker, H., Schapire, R., & Simard, P. (1993). Boosting performance in neural networks. *Intl. J. Pat. Recog. Art. Intel.*, 7, 705–719.

Duda, R. & Hart, P. (1973). *Pattern classification and scene analysis*. New York, NY: Wiley.

Fahlman, S. & Hinton, G. (1987). Connectionist architectures for artificial intelligence. *Computer*, 20, 100–109.

Fan, D., Chan, P., & Stolfo, S. (1996). A comparative evaluation of combiner and stacked generalization. *Working Notes AAAI-96 Work. Integrating Multiple Learned Models* (pp. 40–46).

Fayyad, U., Weir, N., & Djorgovski, S. (1993). SKICAT: A machine learning system for automated cataloging of large scale sky surveys. *Proc. Tenth Intl. Conf. Machine Learning* (pp. 112–119).

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2, 139–172.

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

Freund, Y. (1992). An improved boosting algorithm and its implications on learning complexity. *Proc. 5th Work. Comp. Learning Theory* (pp. 391–398).

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., & Sunderam, V. (1993). *PVM 3 user's guide and reference manual.* (Technical Report ORNL/TM-12187), Oak Ridge, TN: Oak Ridge National Laboratory.

Goodman, R. & Smyth, P. (1989). The induction of probabilisitc rule sets–the itrule algorithm. *Proc. Sixth Intl. Work. Machine Learning* (pp. 129–132).

Grammes, C. (1993). Gnufit v1.2. (Available at ftp://ftp.dartmouth.edu/pub/gnuplot/gnufit12.tar.gz).

Gustafson, J. (1988). Reevaluating Amdahl's law. *Comm. ACM*, 31(5), 532–533.

Hansen, L. & Salamon, P. (1990). Neural network ensembles. *IEEE Trans. Pattern Analysis and Mach. Itell.*, 12, 993–1001.

Hernandez, M. & Stolfo, S. (1995). The merge/purge problem for large databases. *Proc. SIGMOD-95* (pp. 127–138).

Hinton, G. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40, 185–234.

Holder, L. (1991). Selection of learning methods using an adaptive model of knowledge utility. *Proc. First Intl. Work. Multistrategy Learning* (pp. 247–254).

Hunter, L. (1993). Molecular biology for computer scientists. In L. Hunter (Ed.), *Aritficial Intelligence and Molecular Biology*, chapter 1, (pp. 1–46). AAAI Press.

Kohavi, R. & John, G. (1995). Automatic parameter selection by minimizing estimated error. *Proc. 12th Intl. Conf. Mach. Learn.* (pp. 304–312). Morgan Kaufmann.

Kohavi, R. & Wolpert, D. (1996). Bias plus variacne decomposition for zero-one loss functions. *Proc. Thirteenth Intl. Conf. Machine Learning* (pp. 275–283).

Kong, E. B. & Dietterich, T. (1995). Error-correcting output coding corrects bias and variance. *Proc. Twelfth Intl. Conf. Machine Learning* (pp. 313–321).

Krogh, A. & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in Neural Info. Proc. Sys. 7* (pp. 231–238). MIT Press.

Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing: Design and analysis of algorithms.* Redwood City, CA: Benjamin-Cummings.

Kumar, V. & Gupta, A. (1994). Analyzing scalability of parallel algorithms and architectures. *J. Parallel & Distributed Computing*, 22, 379–391.

Kwok, S. & Carter, C. (1990). Multiple decision trees. *Uncertainty in Aritificial Intelligence 4* (pp. 327–335).

Langley, P., Iba, W., & Thompson, K. (1992). An analysis of bayesian classifiers. *Proc. AAAI-92* (pp. 223–228).

Langley, P. & Sage, S. (1994). Induction of selective bayesian classifiers. *Proc. Tenth Conf. Uncertainty in AI* (pp. 399–406).

Langley, P. & Simon, H. (1995). Applications of machine learning and rule induction. *Communications of the ACM*, 38(11), 54–64.

Lewin, B. (1987). *Genes.* New York, NY: John Wiley & Son.

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, 5(2), 4–22.

Lipton, R. (1995). Using DNA to solve NP-complete problems. *Science*, 268, 542–545.

Littlestone, N. & Warmuth, M. (1989). *The weighted majority algorithm.* (Technical Report UCSC-CRL-89-16), Santa Cruz, CA: Computer Research Lab., Univ. of California.

Matheus, C., Chan, P., & Piatesky-Shapiro, G. (1993). Systems for knowledge discovery in databases. *IEEE Trans. Knowledge and Data Engineering*, 5(6), 903–913.

Matheus, C. J. & Rendell, L. A. (1989). Constructive induction on decision trees. *Proc. of IJCAI-89* (pp. 645–650).

Merz, C. & Murphy, P. (1996). UCI repository of machine learning databases [http://www.ics.uci.edu/~mlearn/mlrepository.html]. Dept. of Info. and Computer Sci., Univ. of California, Irvine, CA.

Michalski, R. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, (pp. 83–134). Morgan Kaufmann.

Michalski, R. & Stepp, R. (1983). Learning from observation: Conceptual clustering. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, (pp. 331–363). Morgan Kaufmann.

Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multipurpose incremental leaning system AQ51 and its testing application to three medical domains. *Proc. AAAI-86* (pp. 1041–1045).

Mitchell, T. (1980). *The Need for Biases in Learning Generalizaions.* (Technical Report CBM-TR-117): Dept. Comp. Sci., Rutgers Univ.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, 18, 203–226.

Murphy, P. & Pazzani, M. (1991). ID2-of-3: constructive induction of M-of-N concepts for discriminators in decisions trees. *Proc. Eigth Intl. work. Machine Learning* (pp. 183–187).

Naik, D. & Mammone, R. (1992). Meta-neural networks that learn by learning. *Proc. IJCNN* (pp. I:437–442).

Nakata, K., Kanchisa, M., & DeLisi, C. (1985). Prediction of splice junctions in mRNA sequences. *Nucl. Acids Res.*, 13, 5327–5340.

Optiz, D. & Shavlik, J. (1996). Generating accurage and diverse members of a neural-network ensemble. In D. Touretzky, M. Morzer, & M. Hasselmo (Eds.), *Advances in Neural Info. Proc. Sys. 8.* MIT Press. to appear.

Ourston, D. & Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. *Proc. AAAI-90* (pp. 815–820).

G. Piatesky-Shapiro & W. Frawley (Eds.) (1991). *Knowledge discovery in databases.* Cambridge, MA: AAAI Press.

Pomerleau, D. (1992). *Neural network perception for mobile robot guidance.* PhD thesis, Pittsburgh, PA: School of Computer Sci., Carnegie Mellon Univ. (Tech. Rep. CMU-CS-92-115).

Press, W., Flannery, B., Teukolsky, S., & Vetterling, W. (1988). *Numerical recipies in C: The art of scientific computing.* Cambridge, UK: Cambridge University Press.

Provost, F. & Aronis, J. (1996). Scaling up inductive learning with massive parallelism. *Machine Learning*, 23, 33–46.

Provost, F. & Hennessy, D. (1996). Scaling up: Distributed machine learning with cooperation. *Proc. AAAI-96.* AAAI Press. 74-79.

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

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.

Quinlan, J. R. (1993). *C4.5: programs for machine learning.* San Mateo, CA: Morgan Kaufmann.

Ralston, A. & Rabinowitz, P. (1978). *A first course in numerical analysis.* New York, NY: McGraw-Hill.

Rendell, L. (1990). Feature construction for concept learning. In D. P. Benjamin (Ed.), *Change of Representation and Inductive Bias*, (pp. 327–353). Kluwer Academic.

Rivest, R. (1987). Learning decision lists. *Machine Learning*, 2, 229–246.

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

Schleif, R. (1986). *Genetics and Molecular Biology.* Reading, MA: Addison-Wesley.

Selberg, E. & Etzioni, O. (1996). Multi-service search and comparison using the MetaCrawler. *Proc. Fourth Intl. World Wide Web Conf.*

Silver, B., Frawley, W., Iba, G., Vittal, J., & Bradford, K. (1990). ILS: A framework for multi-paradigmatic learning. *Proc. Seventh Intl. Conf. Machine Learning* (pp. 348–356).

Stanfill, C. & Waltz, D. (1986). Toward memory-based reasoning. *Comm. ACM*, 29(12), 1213–1228.

Stolfo, S., Galil, Z., McKeown, K., & Mills, R. (1989). Speech recognition in parallel. *Proc. Speech Nat. Lang. Work.* (pp. 353–373).

Sun, X. & Ni, L. (1993). Scalable problems and memory-bounded speedup. *J. Parallel & Distributed Comp.*, 19, 27–37.

Tcheng, D., Lambert, B., Lu, C.-Y., & Rendell, L. (1989). Building robust learning systems by computing induction and optimization. *Proc. IJCAI-89* (pp. 806–812).

Towell, G. & Shavlik, J. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, 13, 71–101.

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.

Valiant, L. (1984). A theory of the learnable. *Comm. ACM*, 27, 1134–1142.

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.

Wolpert, D. (1993). Personal communication.

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.