Management of Intelligent Learning Agents in Distributed Data Mining Systems

Andreas Leonidas Prodromidis

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

1999

ABSTRACT

Management of Intelligent Learning Agents in Distributed Data Mining Systems

Andreas Leonidas Prodromidis

Data mining systems aim to discover patterns and extract useful information from facts recorded in databases. One means of acquiring 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.

Most of the current generation of learning algorithms, however, are computationally complex and require all data to be resident in main memory which is clearly untenable for many realistic problems and databases. In this dissertation we investigate data mining techniques that scale up to large and physically distributed data sets. Specifically, we describe the JAM system (Java Agents for Meta-learning), an extensible agent-based distributed data mining system that supports the remote dispatch and exchange of agents among participating data sites and employs meta-learning techniques to combine the multiple models that are learned. Several important desiderata of data mining systems are addressed (i.e., scalability, efficiency, portability, compatibility, adaptivity, extensibility and effectiveness) and a combination of AI-based methods and distributed systems techniques are presented.

We applied JAM on the real-world data mining task of modeling and detecting credit card fraud with notable success. Inductive learning agents are used to compute detectors of anomalous or errant behavior over inherently distributed data sets and meta-learning methods integrate their collective knowledge into higher level classification models or meta-classifiers. By supporting the exchange of models or classifier agents among data sites, our approach facilitates the cooperation between financial organizations and provides unified and cross-institution protection mechanisms against fraudulent transactions.

Contents

List of Tables

List of Figures

Acknowledgments

Pursuing a Ph.D. is similar to embarking on a long journey with unforseen destination. Being close to my Ithaca, I realize I wouldn't have reached my destination without the help of several people. First and foremost, I would like to thank my compass, my advisor Sal Stolfo, who helped me sail through rough waters and unfamiliar territories. I couldn't have asked for a better advisor; he was always present, but always at a distance. He fostered my research skills and presentation abilities, but most of all, he taught me to always keep a positive attitude.

I am very grateful to Dan Duchamp for inviting me to Columbia University, and for guiding me through the first two years of my journey. Dan gave me ample space to clear major hurdles while teaching me a great deal about operating systems and large software systems.

My sincere gratitude goes to a select group of people from IBM Research, Ajei Gopal, Nayeem Islam, Liana Fong and Mark Squillante. Concurrently to my studies at Columbia, I was fortunate to intern for two and a half years at the T.J. Watson Research Center and collaborate with them on a number of exciting projects. Much of what I know about distributed systems and resource management came out of some notorious and intense evening discussions. During this long internship, I was mentored by Ajei Gopal first, and by Nayeem Islam later. Ajei and Nayeem helped me build my confidence and offered me a safe harbor during a stormy transition period that marked my journey. Special thanks to Liana for her continuous encouragement and trust and her kind friendship and to Mark for generously offering his advice and support.

I am indebted to Kathy McKeown, Luis Gravano, Alex Tuzhilin and Shih-Fu Chang for serving in my thesis committee and for their careful review and valuable comments on this document. Moreover, I would like to thank Kathy and Luis for supporting this work at a very early stage as members of my thesis proposal committee and for providing invaluable help in my job search a few months ago. Many thanks go to Timos Sellis and Zvi Galil for aiding my job hunt as well. I was lucky to meet Timos during my last year as an undergraduate student in Greece and since then, he was very supportive of my graduate studies and always open to offer advice. Zvi Galil was one of the first and most helpful and considerate people I met when I joined the Computer Science Department of

Columbia University. Six years later, I classify him as my favorite Dean. Besides, I had the pleasure of joining him for long-distance running on many occasions.

I am very thankful to the Department of Computer Science of Columbia University for allowing me to their facilities and their prompt advice on many administrative issues; in particular, Rosemary Addarich, Alice Cueba, Melbourne Francis, Patricia Hervey, Mary Van Starrex, Renate Valencia, and Martha Zadok.

My officemates, Harry Harjono, Wei Fan and Wenke Lee, engaged me in many fruitful discussions both technical and non-technical. Religion, politics, sports, movies and entertainment have always been our favorite subjects. Philip Chan helped me shape and clarify some ideas of my thesis, while Shelley Tselepis, Jeffrey Sherwin, Terrance Truta and Dave Kalina contributed to the development of the data mining system described in this dissertation. I am very much obliged to Christopher Merz who generously shared his SCANN software and the Central Research Facilities staff for their patience when I practically seized nearly every machine of the department for my computationally expensive experiments.

Since I came to New York to study, I have been privileged with the friendship of many unique people. It is difficult to acknowledge everyone without risking to appear as a Hollywood celebrity that has just received a Film Academy award. My roommate Blair MacIntyre and I spent many hours fighting each other through PlayStation games, Apostolos Dailianas introduced me to a new form of art, the "Dolce Vita" movie by Fellini which sparked endless discussions and debates, Hui Lei entertained me over our lunch breaks and Yiannis Stamos survived my capricious little jokes whenever I needed short breaks. To be honest, many more were victims of my harmless small pranks over the last few years, including Damianos Chantziantoniou, Marilena Xiridou, Angeliki Pollali, Beatris Lexutt, Reza Jalili, Monika Kendall, Andrew Senior. I apologize for my behavior and I thank them for their patience. Many more friends, Dimitris Pendarakis, Natasa Kouparousou, Harriet Zois, Maria Papadopouli, Thomas Dunn, Evelina Dimitrova, Simon Baker, the Columbia University road-runners, the International House ballroom dance group and the Hellenic association of Columbia University have made my journey a wonderful experience.

My parents Kyprianos and Anastasia prepared me for this long-lasting trek. If I completed my Ph.D. studies successfully is because of their sacrifices, their devotion and their determination. I would have never known how to read the map and avoid the rocks without them. My brother Prodromos, my sister Anna, deserve much of the credit as well. I wouldn't have become what I am had I not grown up next to them. A very special thank you goes to my grandmother Anna and my late grandmother Kalliopi for their continuous cheering and goodwill.

My wife Mirka continues what my parents started. She has been an endless source of motivation, comfort and understanding, when I had to spend extra weekends and nights in front of the computer chasing deadlines. I would never have finished this journey or

decided to embark on the next one without her. She has been my pharos when navigating near reefs and my anchor when facing opposing winds and currents.

The research presented in this thesis was partly supported by an IBM fellowship and in part by the Intrusion Detection Program (BAA9603) from the Defense Advanced Research Projects Agency under grant F30602-96-1-0311, the Database and Expert Systems and Knowledge Models and Cognitive Systems Programs of the National Science Foundation under grant IRI-96-32225, the CISE Research Infrastructure Grant Program of the National Science Foundation under grant CDA-96-25374 and the Center for Advanced Technology at Polytechnic University of the New York State Science and Technology Foundation under grant Polytechnic 423115-445.

Στους γονείς μου
Κυπριανό και Αναστασία

To my parents
Kyprianos and Anastasia

Chapter 1

Introduction

The term "Information Age" is probably the most appropriate in describing the second half of the twentieth century. Indeed, it is estimated that a person today receives more information in a single day than a person could have received during his or her lifetime in the seventeenth century.

During the last decade, our ability to collect and store data has significantly outpaced our ability to analyze, summarize and extract "knowledge" from this continuous stream of input. A short list of examples is probably enough to place the current situation into perspective:

- NASA's Earth Observing System (EOS) of orbiting satellites and other spaceborne instruments send one terabyte of data to receiving stations every day [Way & Smith, 1991].
- The World Wide Web is estimated to have at least 6 terabytes of text data in 3 million servers and as many as 800 million HTML pages (as of June 1999), and it is still growing exponentially — as recently as 1993, there were a mere 50 servers [Lawrence & Giles, 1999].
- By the year 2,000 a typical Fortune 500 company is projected to possess more than 400 trillion characters in their electronic databases requiring 400 terabytes of mass storage [Belford, 1998].

Traditional data analysis methods that require humans to process large data sets are completely inadequate; to quote John Naisbett, "We are drowning in information but starving for knowledge"!

The relatively new field of *Knowledge Discovery and Data Mining* (*KDD*) has emerged to compensate for these deficiencies. *Knowledge discovery in databases* denotes the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data [Fayyad *et al.*, 1996]. The field spans several research areas such as databases, machine learning, neural networks, statistics, pattern recognition, artificial intelligence and reasoning with uncertainty, information retrieval, data visualization, summarization, distributed systems and high performance computing [Fayyad, Piatetsky-Shapiro, & Smyth, 1996], to name a few.

Database theories and tools provide the necessary infrastructure to store, access and manipulate data; machine learning, neural networks and pattern recognition research is concerned with inferring models and extracting patterns from data; statistics is used to evaluate and analyze the data and control the chances and risks that must be considered upon making generalizations; summarization and data visualization examine methods to easily convey a summary and interpretation of the information gathered; distributed and high performance computing deals with performance and scalability issues in distributed systems, the protocols employed between the data sites and the efficiency and scalability of the algorithms in the context of massive databases.

The *KDD* process is a multi-step interactive process, which includes data selection, data preprocessing, transformation and cleaning, incorporation of appropriate prior knowledge, data mining, knowledge evaluation, and refinement involving modifications. The process is non-trivial and involves search for structure, patterns or models in large and typically multi-dimensional data. Performing an aggregation query on a data set, for example, although useful, is not considered as new or discovered knowledge.

KDD is a very broad subject that cannot be exhaustively covered in this thesis. Instead, we build upon the findings, results and solutions provided by the database, machine learning, statistics and data visualization communities to design a scalable and distributed data mining system. The following pages frame the scope of the problem, and describe our contributions to this new and rapidly evolving field of computer science research.

1.1 Distributed Data Mining

Data mining refers to a particular step in the *KDD* process. According to the most recent and broad definition [Fayyad *et al.*, 1996], "*data mining* consists of particular algorithms (methods) that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (models) over the data".

In a relational database context, one of the typical *data mining tasks* is to explain and predict the value of some attribute given a collection of tuples with known attribute values. One means of performing such a task is to employ various machine learning algorithms. An existing relation, drawn from some domain, is thus treated as training data for a learning algorithm that computes a logical expression or a concept description, or a descriptive model, or a *classifier*, that can later be used to predict (for a variety of strategic and tactical purposes) the value of a desired or target attribute for some record whose desired attribute value is unknown.

The field of machine learning has made substantial progress over the last few decades and numerous algorithms, ranging from those based on stochastic models to those based on purely symbolic representations like rules and decision trees, have already been developed and applied to many problems in diverse areas. Over the past decade, machine learning has evolved from a field of laboratory demonstrations to a field of significant commercial value [Mitchell, 1997b]. Machine-learning algorithms have been deployed in heart disease diagnosis [Detrano *et al.*, 1989], in predicting glucose levels for diabetic patients [Carson & Fischer, 1990], in detecting credit card fraud [Stolfo *et al.*, 1997a], in steering vehicles driving autonomously on public highways at 70 miles an hour [Pomerleau, 1992], in predicting stock option pricing [Malliaris & Salchenberger, 1993] and in computing customized electronic newspapers [K.Lang, 1995], to name a few applications. Many large business institutions and market analysis firms attempt to distinguish the low-risk (high profit) potential customers by learning simple categorical classifications of their potential customer base. Similarly, defense and intelligence operations utilize similar methodologies on vast information sources to predict a wide range of conditions in various contexts. Recently, for example, data mining techniques have been successfully applied to intrusion detection in network-based systems [W. Lee, 1998].

One of the main challenges in machine learning and data mining, however, is the development of inductive learning techniques that scale up to large and possibly physically distributed data sets. Many organizations seeking added value from their data are already dealing with overwhelming amounts of information. The number and size of their databases and data warehouses grows at phenomenal rates, faster than the corresponding improvements in machine resources and inductive learning techniques. Most of the current generation of learning algorithms are computationally complex and require all data to be resident in main memory, which is clearly untenable for many realistic problems and databases.

Furthermore, in certain cases, data may be inherently distributed and cannot be localized on any one machine (even by a trusted third party) for a variety of practical reasons including security and fault tolerant distribution of data and services, competitive (business) reasons, statutory constraints imposed by law as well as physically dispersed databases or mobile platforms like an armada of ships. In such situations, it may not be feasible to inspect all of the data at one processing site to compute one primary "global" concept or model.

1.2 Thesis Statement

Meta-learning is a technique recently developed [Chan, Stolfo, & Wolpert, 1996; Dietterich, 1997; Provost & Kolluri, 1997] that deals with the problem of learning useful new information from large and inherently distributed databases. Meta-learning aims to compute a number of independent classifiers (concepts or models) by applying learning programs to a collection of independent and inherently distributed databases in parallel. The "base classifiers" computed are then collected and combined by another learning process. Here meta-learning seeks to compute a "meta-classifier" that integrates in some principled fashion the separately learned classifiers to boost overall predictive accuracy.

The **main objective** in this thesis is to take advantage of the inherent parallelism and distributed nature of meta-learning and design and implement a powerful and practical distributed data mining system. Assuming that the system consists of several databases interconnected through an intranet or internet, the goal is to provide the means for each data site to utilize its own local data and, at the same time, benefit from the data that is available at other data sites without transferring or directly accessing that data. In this context, this can be materialized by learning agents that execute at remote data and generate classifier agents that can subsequently be transfered among the sites. We have achieved this goal through the implementation and demonstration of a system we call JAM (Java Agents for Meta-Learning) [Stolfo *et al.*, 1997b].

JAM, however, is more than an implementation of a distributed meta-learning system. It is a distributed data mining system addressing many practical problems for which centralized or single-host systems are not appropriate. On the other hand, distributed systems have increased complexity. Their practical value depends on the **scalability** of the distributed protocols as the number of the data sites and the size of the databases increases, and on the **efficiency** of their methods to use the system resources effectively. Furthermore, distributed systems may need to run across heterogenous platforms (**portability**) or operate over databases that may (possibly) have different schemata (**compatibility**). There are other important problems, intrinsic within data mining systems, that should not be ignored. Data mining systems should be **adaptive** to environment changes (e.g., when data and objectives change over time), **extensible** to support new and more advanced data mining technologies and, last but not least, highly **accurate**. The focus of this thesis is to identify and describe each of these issues separately and to detail our approaches within the framework of JAM [Prodromidis, 1997; Prodromidis, Chan, & Stolfo, 1999].

JAM has been used in several experiments dealing with real-world learning tasks, such as solving crucial problems in fraud detection in financial information systems [Stolfo *et al.*, 1998; Chan *et al.*, 1999]. The objective here, is to employ pattern-directed inference systems using models of anomalous or errant transaction behaviors to forewarn impeding threats. This approach requires analysis of large and inherently distributed databases of information (e.g., from distinct banks) about transaction behaviors to produce models of "probably fraudulent" transactions.

1.2.1 Scalability and Efficiency

The **scalability** of a data mining system refers to the ability of the system to operate effectively and without a substantial or discernable reduction in performance as the number of data sites increases. **Efficiency**, on the other hand, refers to the effective use of the available system resources. The former depends on the protocols that transfer and manage the intelligent agents to support the collaboration of the data sites while the latter depends upon the appropriate evaluation and filtering of the available agents to minimize redundancy. Combining scalability and efficiency without sacrificing predictive performance is, however, an intricate problem. To understand the issues and better tackle the complexity of the problem, we examined scalability and efficiency at two levels, the system architecture level and the data site (meta-learning) level.

System architecture level

First we focus on the components of the system and the overall architecture. Assuming that the data mining system comprises several data sites, each with its own resources, databases, machine learning agents and meta-learning capabilities, we designed a protocol that allows the data sites to collaborate efficiently without hindering their progress. Our proposed distributed data mining system is JAM, a powerful agent based meta-learning system for large scale data mining applications. Briefly, JAM provides a set of learning programs, implemented or wrapped within JAVA agents, that compute models (or classifiers) over data stored locally at a site. JAM also provides a set of *meta-learning* agents for combining multiple models that were learned (perhaps) at different sites [Prodromidis & Stolfo, 1999c]. Furthermore, it employs a special distribution mechanism that allows the migration of the derived models or *classifier agents* to other remote sites.

Figure 1.1 depicts the JAM system with three data sites, Orange, Mango and Strawberry while exchanging their classifier agents. In this instance, Mango is shown to have imported the CART and Ripper classifier agents from Orange and the Bayes Classifier agent from Strawberry and combined with its own ID3 classifier agent into a local Bayes meta-classifier agent. Thus, Mango utilizes information from Orange and Strawberry without ever directly accessing their data.

The JAM system can be viewed as a coarse-grain parallel application where the constituent sites function autonomously and (occasionally) exchange classifiers with each other. JAM is designed with asynchronous, distributed communication protocols that enable the participating database sites to operate independently and collaborate with other peer sites as necessary, thus eliminating centralized control and synchronization barriers.

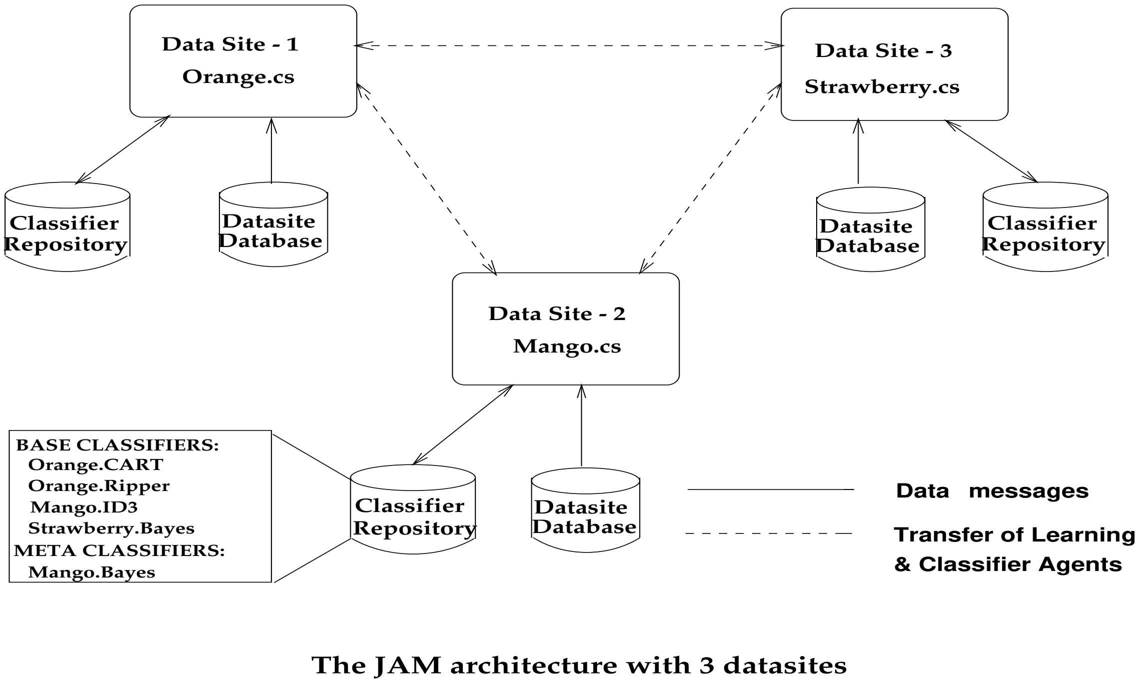

Figure 1.1: The JAM system.

Meta-learning level

Employing efficient distributed protocols, however, addresses the scalability problem only partially. The scalability of the system depends greatly on the efficiency of its components (data sites). The analysis of the dependencies among the classifiers, the management of the agents and the efficiency of the meta-classifiers within the data sites constitutes the other half (meta-learning level) of the scalability problem.

Meta-classifiers are defined recursively as collections of classifiers structured in multi-level trees [Chan & Stolfo, 1996]. Such structures, however, can be unnecessarily complex, meaning that many classifiers may be redundant, wasting resources and reducing system throughput. (Throughput here denotes the rate at which a stream of data items can be piped through and labeled by a meta-classifier.) We study the efficiency of meta-classifiers by investigating the effects of pruning (discarding certain base classifiers) on their performance [Prodromidis & Stolfo, 1998b; Prodromidis, Stolfo, & Chan, 1999; Prodromidis & Stolfo, 1999d; 1999b]. Determining the optimal set of classifiers for meta-learning is a combinatorial problem. Hence, the objective of pruning is to utilize heuristic methods to search for partially grown meta-classifiers (meta-classifiers with pruned sub-trees) that are more efficient and scalable and at the same time achieve comparable or better predictive performance results than fully grown (unpruned) meta-classifiers. To this end, we introduce two stages for pruning meta-classifiers, the *a-priori pruning* or *pre-training pruning* and the *a-posteriori pruning* or *post-training pruning* stages. Both levels are essential and complementary to each other with respect to the improvement of the accuracy and efficiency of the system.

A-priori pruning or *pre-training pruning* refers to the filtering of the classifiers before they are combined. Instead of combining classifiers in a brute force manner, with pre-training pruning we introduce a preliminary stage for analyzing the available classifiers and qualifying them for inclusion in a combined meta-classifier. Only those classifiers that appear (according to one or more pre-defined metrics) to be most "promising" participate in the final meta-classifier. Here, we adopt a "black-box" approach that evaluates the set of classifiers based only on their input and output behavior, not their internal structure. Conversely, *a-posteriori pruning* or *post-training pruning*, denotes the evaluation and pruning of constituent base classifiers after a complete meta-classifier has been constructed. In this dissertation, we examined three pre-training pruning and two post-training pruning algorithms each with different search heuristics. The first pre-training pruning algorithm ranks and selects its classifiers by evaluating each candidate classifier independently (metric-based), the second algorithm decides by examining the classifiers in correlation with each other (diversity-based), while the third relies on the independent performance of the classifiers and the manner in which they predict with respect to each other and with respect to the underlying data set (specialty/coverage-based). The first post-training pruning algorithms is based on a cost-complexity pruning technique (a technique that seeks the meta-classifier with the lowest cost (size) and the best performance) while the second algorithm is based on the correlation between the classifiers and the meta-classifier.

There are two primary objectives for the distributed protocols and the pruning techniques:

1. to acquire and combine information from multiple databases in a timely manner;

2. to generate effective and efficient meta-classifiers.

We evaluated the effectiveness of the proposed methods through experiments performed on real credit card data provided by two different financial institutions, where the target application is to compute predictive models that detect fraudulent transactions [Prodromidis & Stolfo, 1999a]. Our empirical study presents and compares the results of the different pruning techniques under three realistic evaluation metrics (accuracy, TP-FP[1], and a cost model fitted to the credit card fraud detection problem), and conducts an in-depth analysis of the strengths and weaknesses of these methods.

[1]TP stands for True Positive, i.e., percentage of actual fraud that is caught; FP stands for False Positive, i.e., percentage of false alarms.

1.2.2 Portability

A distributed data mining system should be capable of operating across multiple environments with different hardware and software configurations (e.g., across the Internet), and be able to combine multiple models with (possibly) different representations.

The JAM system presented in this thesis is a distributed computing construct designed to extend the OS environments to accommodate such requirements. As implied by its name (Java Agents for Meta-learning), portability is inherent within JAM. The "Java" part denotes that we have used Java technology to build the composing parts of the system, including the underlying infrastructure, the learning and classifier agents, and the specific operators that generate and spawn agents. The learning agents are the basic components for searching for patterns within the data and the classifier agents are the units that capture the computed models and can be shared among the data sites. The platform independence of Java makes it easy for each JAM site to delegate its agents to any participating site. "Meta-learning" refers to the system's methods for combining classifier agents. It has the advantage of integrating classifiers independently of their internal representation and the learning algorithms that were used to compute them.

1.2.3 Compatibility

Combining multiple models has been receiving increasing attention in the literature [Dietterich, 1997]. In much of the prior work on combining multiple models, it is assumed that all models originate from the same database or from databases of identical schema. This is not always the case and differences in the type and number of attributes among different data sets are not uncommon. The classifiers computed at a database depend directly on the format of the underlying data and minor differences in the schema between databases derive incompatible classifiers, i.e., a classifier cannot be applied on data of different formats. Yet, these classifiers may target the same concept. We seek to "bridge" these disparate classifiers in some principled fashion.

In the credit card fraud detection problem (also used to evaluate the efficiency of the pruning methods), for instance, the data sets were provided by two different banks. Both institutions seek to be able to exchange their classifiers and hence incorporate in their system useful information that would otherwise be inaccessible to both. Indeed, although for each credit card transaction both institutions record similar information, they also include specific fields containing important information that each has acquired separately and which provides predictive value in determining fraudulent transaction patterns. In a different scenario where databases and schemata evolve over time, it may be desirable for a single institution to combine classifiers from both past accumulated data with newly acquired data. To facilitate the exchange of knowledge and take advantage of incompatible and otherwise useless classifiers, we devised methods that bridge the differences imposed by the different schemata.

Integrating the information captured by such classifiers is a non-trivial problem that we call, *"the incompatible schema problem"* [Prodromidis & Stolfo, 1998a]. (The reader is advised not to confuse this with Schema Integration over Federated/Mediated Databases.) In this thesis, we investigate this problem systematically and we describe several approaches that allow JAM and other distributed data mining systems to cope with *incompatible* classifiers. The validity and potential utility of these methods is demonstrated experimentally on the credit card data sets obtained by the two independent financial institutions. By alleviating the conflicts, the system can combine classifiers with a somewhat different view of the classification problem that would otherwise be useless.

1.2.4 Adaptivity and Extensibility

Most data mining systems operate in environments that are almost certainly bound to change, a phenomenon known as *concept drift*. For example, medical science evolves, and with it the types of medication, the dosages and treatments, and of course the data included in the various medical database; lifestyles change over time and so do the profiles of customers included in credit card data; new security systems are introduced and new ways to commit fraud or to break into systems are devised. Most traditional data mining systems are static. To generate new classifiers to model both existing and new data sets, these systems require, in most cases, the re-application of the learning algorithms on the union of the existing and the new data.

Adaptivity in JAM, is achieved by re-applying the meta-learning principles to update the existing higher-level (meta-level) concepts as new data is collected and new patterns emerge over time. New information is treated similarly to remote information. Instead of combining classifiers from remote data sites (integration over space), the adaptive data mining system combines classifiers computed over different time periods (integration over time).

It is not only patterns that change over time. Advances in machine learning and data mining are bound to give rise to algorithms and tools that are not available at the present time as well. Unless the data mining system is flexible to accommodate existing as well as future data mining technology it will rapidly be rendered inadequate and obsolete. To ensure **extensibility**, JAM is designed using object-oriented methods and is implemented independently of any particular machine learning program or any meta-learning or classifier combining technique. JAM's extensible plug-and-play capabilities provide the means to easily incorporate any new learning algorithm and classifier program.

1.3 Thesis Contributions

In this dissertation we examine several important desiderata related to distributed data mining systems and describe JAM, a powerful, integrated and practical system for managing intelligent learning agents across multiple database sites. Here, we briefly summarize the contributions of this research:

- The design of JAM (Java agents for Meta-learning), a novel distributed data mining system that is based on meta-learning. To our knowledge, JAM is the first system to date that employs meta-learning as a means to mine distributed databases. (A commercial system based upon JAM has recently appeared.)

- The design of a scalable, distributed and asynchronous protocol that facilitates the cooperation and exchange of learning and classifier agents among multiple data sites.

- The extensible object-oriented design for decoupling the learning and meta-learning algorithms from the JAM system to provide the flexibility to snap in new learning programs at any time.

- The adaptation of current meta-learning techniques to combine models computed over data collected from different periods of time.

- The implementation of a prototype of the JAM system. JAM is built upon existing agent infrastructure available over the Internet that ensures portability across heterogenous platforms. JAM release 3.0 is available in the public domain and is used by many researchers around the world.

- The introduction of three pre-training pruning algorithms, each with different heuristics, for evaluating and selecting classification models before they are included in meta-classifiers. The three algorithms require no prior information about the models and their internal representation. Instead they adopt a "black-box" approach.

- Two novel post-training pruning algorithms for discarding redundant classification models after they are included in meta-classifiers.

- A detailed account on the similarities and differences between several meta-learning methods. The analysis examines and contrasts the applicability of the pruning methods on a number of existing techniques for combining classifiers (majority voting, SCANN [Merz, 1999]).

- A systematic approach for bridging databases with different schemata and for combining incompatible classification models.

- The application of JAM on the real-world learning task of fraud detection in financial information systems and the evaluation of their performance under three different realistic metrics (accuracy, TP-FP, and a cost model) by applying a (parallel and efficient) six-fold cross-validation technique.

- A thorough evaluation and comparison (including extensive measurements on the tradeoff between the degree of pruning and the predictive effectiveness) of the performance of the various pruning algorithms and an in-depth analysis of their strengths and weaknesses. The study reveals that the success of each method varies with the underlying data set, the quality, the characteristics and diversity of the candidate set of classifiers, and the metric used to evaluate the performance. By way of summary, we found that pruned meta-classifiers can sustain accuracy levels, increase the TP-FP spread and reduce losses due to credit card fraud at a substantially higher throughput, compared to the unpruned meta-classifiers.

- A methodical assessment of the effectiveness of the bridging algorithms by combining classification models from two financial institutions. The empirical evaluation suggests that employing bridging methods and combining models from different sources can substantially improve performance. In the credit card fraud detection domain, the bridging techniques enabled the meta-learning of incompatible fraud detectors (classifiers) obtained from the two different institutions. Meta-learning produced a meta-classifier with global view that demonstrated higher fraud detection rates and reduced losses due to fraud than any other meta-classifier computed over the data of a single institution.

- The integration of each of these methods in the design of a single coherent distributed data mining system.

1.4 Thesis Outline

The rest of this dissertation is organized as follows:

Chapter 2 provides the necessary background in machine learning and specifically in inductive learning and then gives an overview of several different meta-learning techniques. Furthermore, it introduces several basic definitions in statistics that are used in the empirical evaluation.

Chapter 3 presents the architecture of JAM and discusses the implementation aspects of the system and the management of the intelligent learning agents. The description includes details on the distributed protocols adopted and the scalability, portability, extensibility and adaptivity properties of the system.

Chapter 4 outlines the data mining task of detecting fraudulent use of credit cards and how JAM can be used to address this real-world problem. It also describes the credit card data sets and the evaluation metrics that are used during the empirical analysis.

Chapter 5 discusses the three a-priori pruning algorithms, the metric-based, the diversity-based and the specialty/coverage-based algorithms and Chapter 6 presents the cost complexity-based and correlation-based a-posteriori pruning algorithms. Then we report methods to compute effective and efficient meta-classifiers by combining both pre- and post-training pruning algorithms. An experimental analysis is performed in both chapters, the results are explained in detail and a broad overview of the related research is given.

Chapter 7 formulates the "incompatible schema problem" and details several techniques that bridge the schema differences to accommodate classifier agents that are otherwise incompatible. It also presents the experiments conducted and the performance results collected.

We conclude this dissertation in Chapter 8 by summarizing its contributions and by discussing future research directions.

Chapter 2

Background

Learning, in general, denotes the ability to acquire knowledge, skills or behavioral tendencies on one or more domains through experience, study or instruction. In *machine learning* [Mitchell, 1997a], a computer program is said to **learn** with respect to a class of tasks if its performance on these tasks, as captured by some performance measure, improves with its experience and interactions with its environment.

2.1 Machine Learning

In this thesis research, we concentrate on a particular type of machine learning called *supervised inductive learning* (also called learning from classified examples). Rather than being instructed with explicit rules, a computer may *learn* about a task or a set of tasks by *stimuli* provided from the outside. Given some labeled examples (data) obtained from the environment (supervisor/teacher), supervised inductive learning aims to discover patterns in the examples and form *concepts* that describe the examples. For instance, given some examples of fish, birds and mammals, a *machine learning algorithm* can form a concept that suggests that fish live in the sea, birds fly in the air and mammals usually live on the ground. The computer uses the concepts formed to classify new unseen instances, i.e., assign to a particular input the name of a class to which it belongs.

More formally, *inductive learning* (or *learning from examples* [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 examples, i.e., $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$, for some unknown function $y = f(\mathbf{x})$, with each $\mathbf{x_i}$ interpreted as a set of attribute (feature) vectors x_i of the form $\{x_{i1}, x_{i2}, ..., x_{ik}\}$ and with each y_i representing the class label associated with each vector ($y_i \in \{y_1, y_2, ..., y_m\}$), the task is to compute a *classifier* or *model* $\hat{f}$ that approximates f and 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* of *class* y.

As an example, Table 2.1 shows a medical database of patients examined for the thyroid disease [Merz & Murphy, 1996]. In this table, rows $\mathbf{x_1}, ..., \mathbf{x_4}$ represent patients and columns $\{x_{i1}, ..., x_{i6}\}$ and $\mathbf{y}$ correspond to their medical profiles (ID, age, sex, test results, medication administered) and their diagnoses (i.e., y_1 corresponds to normal, y_2 to hypothyroidism, and y_3 as hyperthyroidism) respectively. A classification model computed over such a data set can subsequently be used to provide, with a certain confidence, prognoses for new patients.

Table 2.1: Thyroid data set

PatientID	Age	Sex	T3 test	TSH	I131 treatment	Diagnosis
38764	56	M	107	0.9	T	Normal
38902	49	F	80	1.1	F	Hypo
39333	38	M	130	1.0	F	Normal
39454	62	M	125	1.7	T	Hyper

Some of the common representations used for the generated classifiers are decision trees, rules, version spaces, neural networks, distance functions, bit strings and probability distributions. In general, these representations are associated with different types of algorithms that extract different types of information from the database and provide alternative capabilities besides the common ability to classify unseen examples 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 able to compute a concept y to classify unseen records (examples). Decision trees are used in CART [Breiman *et al.*, 1984], ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1993], 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], CN2 [Clark & Niblett, 1989] and Ripper [Cohen, 1995] are if-then expressions, where the antecedent is a pattern expression and the consequent is a class label. Each version space learned in VS [Mitchell, 1982] defines the most general and specific description boundaries of a concept using a restricted version of first order formulae. Neural networks [Hopfield, 1982; Rumelhart & McClelland, 1986], compute separating hyperplanes in n-dimensional feature space to classify data [Lippmann, 1987]. The learned distance functions in example-based learning algorithms (or nearest neighbor algorithms) define a similarity or “closeness” measure between two instances [Stanfill & Waltz, 1986; Aha, Kibler, & Albert, 1991]. In genetic algorithms, hypotheses are usually represented by application-specific bit strings. These algorithms search for the most appropriate hypotheses [Holland, 1986; DeJong, 1988] by simulating evolution, i.e., they generate successor hypotheses by repeatedly mutating and recombining (crossover) parts of the best currently known hypothesis [Holland, 1975; DeJong, Spears, & Gordon, 1993]. Conditional probability distributions used by Bayesian classifiers are derived from the frequency distributions of attribute values and reflect the

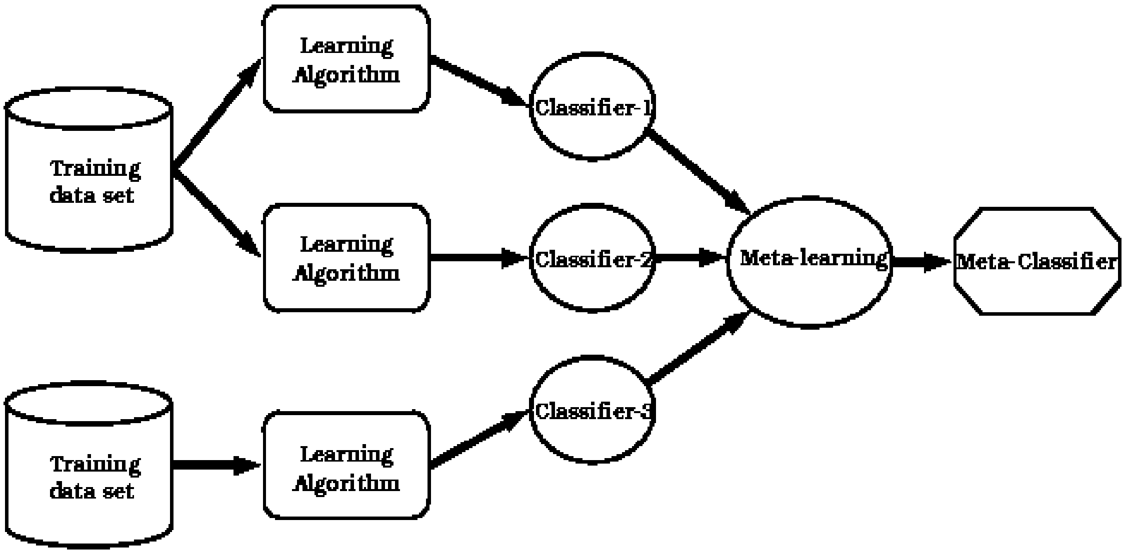

Figure 2.1: Meta-learning process.

likelihood of a certain instance belonging to a particular classification [Duda & Hart, 1973; Cheeseman *et al.*, 1988]. Implicit decision rules classify according to maximal probabilities.

Most of the current research has concentrated on determining the learning algorithm that best fits the target data mining application to compute the best possible classification model. Recently, however, there has been considerable interest in meta-learning techniques that combine or integrate an ensemble of models computed by the same or different learning algorithms over a single or multiple data subsets [Chan, Stolfo, & Wolpert, 1996; Dietterich, 1997; Provost & Kolluri, 1997].

2.2 Meta-Learning

Meta-learning is itself a "learning" process. Loosely defined, meta-learning is about learning from learned knowledge. The idea is to execute a number of concept learning processes on a number of data subsets, and combine their collective results through an extra level of learning.

A graphical representation of meta-learning three different classifiers is depicted in Figure 2.1. In this figure, two classifiers are derived from the same data set (either from different samples or from different learning algorithms, or both) while the third is induced from a separate set. The meta-learning algorithm combines the three classifiers into an ensemble meta-classifier by "learning" how they predict, i.e., by observing their input/output behavior.

Several methods for integrating ensembles of models have been studied, including techniques that combine the set of models in some linear fashion [Ali & Pazzani, 1996; Breiman, 1994; 1996; Freund & Schapire, 1995; Krogh & Vedelsby, 1995; LeBlanc & Tibshirani, 1993; Littlestone & Warmuth, 1989; Opitz & Shavlik, 1996; Perrone & Cooper, 1993; Schapire, 1990; Tresp & Taniguchi, 1995], techniques that employ referee functions to arbitrate among the predictions generated by the classifiers [Chan & Stolfo, 1993b; Jacobs *et al.*, 1991; Jordan & Xu, 1993; R. & J., 1994; Jordan & Jacobs, 1994; Kong & Dietterich, 1995; Ortega, Koppel, & Argamon-Engelson, 1999], methods that rely on principal components analysis [Merz, 1999; Merz & Pazzani, 1999] or methods that apply inductive learning techniques to learn the behavior and properties of the candidate classifiers [Wolpert, 1992; Chan & Stolfo, 1993b].

In this thesis, we describe a distributed meta-learning system that supports, in principle, any of these methods. However, we do not elaborate on all of them. Instead, we concentrate on three representative techniques: *voting, stacking* and *SCANN*.

2.3 Meta-Learning Methods

The three methods, *voting, stacking* and *SCANN*, aim to improve efficiency and scalability by executing a number of learning processes on a number of data subsets in parallel and by combining the collective results. Initially, each learning task, also called a *base learner*, computes a *base classifier*, i.e., a model of its underlying data subset or *training set*. Next, a separate task, integrates these independently computed base classifiers into a higher level classification model. The meta-learning process is illustrated in Figure 2.2.

1. *Base classifiers* are computed by the learning algorithms over separate (base-level) training sets.

2. Next, predictions are generated by applying the learned classifiers on a separate (unseen) data set, called validation set.

3. A meta-level training set is composed from these predictions and the validation set itself.

4. The final classifier (*meta-classifier*) is trained over this meta-level training set.

The various meta-learning strategies differ on the way the meta-level training set is formed and the way the final prediction of the meta-classifier is synthesized. The final meta-classifiers are also classifiers that can also be meta-learned and recursively combined into hierarchical meta-classifier in a similar manner. Before we detail the three combining strategies, however, we introduce the following notation. Let C_i, $i = 1, 2, ..., K$, denote a base classifier computed over sample D_j, $j = 1, 2, ..., N$, of training data T, and m the

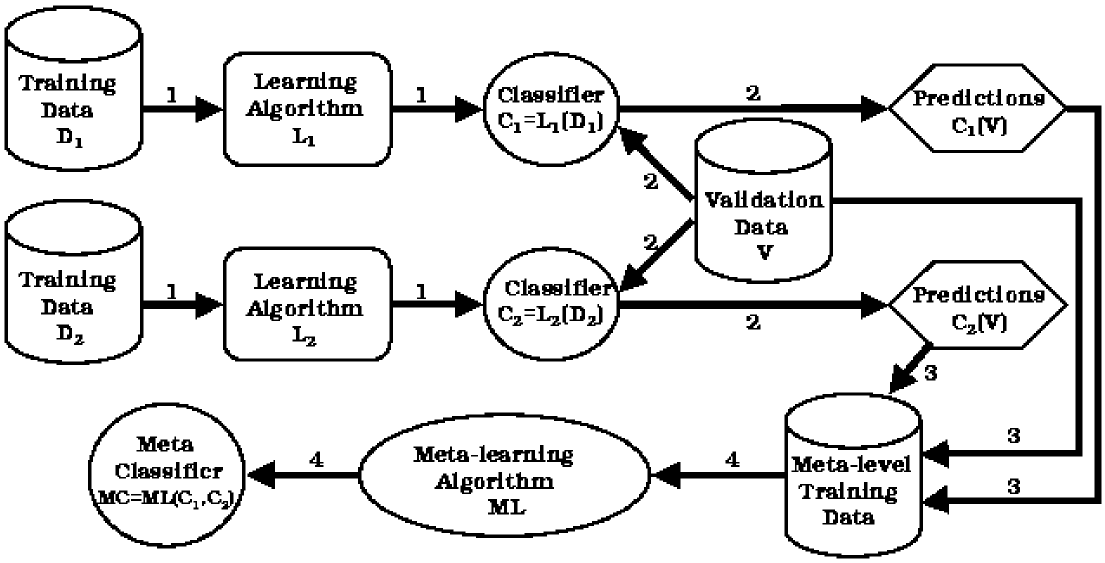

Figure 2.2: Distributed meta-learning process.

number of possible classes. Let $\mathbf{x}$ be an instance whose classification we seek, and $C_1(\mathbf{x})$, $C_2(\mathbf{x})$,..., $C_K(\mathbf{x})$, $C_i(\mathbf{x}) \in \{y_1, y_2, ..., y_m\}$, be the predicted classifications of $\mathbf{x}$ from the K base classifiers, C_i, $i = 1, 2, ..., K$. Finally, let V be a separate validation set of size n that is used to generate the *meta-level* predictions of the K base classifiers.

2.3.1 Voting

Voting denotes the simplest method of combining predictions from multiple classifiers. In its simplest form, called *plurality* or *majority* voting, each classification model contributes a single vote (its own classification). The collective prediction is decided by the majority of the votes, i.e., the class with the most votes is the final prediction. In *weighted* voting, on the other hand, the classifiers have varying degrees of influence on the collective prediction that is relative to their predictive accuracy. Each classifier is associated with a specific weight determined by its performance (e.g., accuracy, cost model) on a validation set. The final prediction is decided by summing over all weighted votes and by choosing the class with the highest aggregate. For a binary classification problem, for example, where each classifier C_i with weight w_i casts a 0 vote for class y_1 and a 1 vote for class y_2, the aggregate is given by:

$$S(\mathbf{x}) = \frac{\sum_{i=1}^{K} w_i C_i(\mathbf{x})}{\sum_{i=1}^{K} w_i} \tag{2.1}$$

If we choose 0.5 to be the threshold distinguishing classes y_1 and y_2, the weighted voting method classifies unlabeled instances $\mathbf{x}$ as y_1 if $S(\mathbf{x}) < 0.5$, as y_2 if $S(\mathbf{x}) > 0.5$ and randomly if $S(\mathbf{x}) = 0.5$.

Plurality voting can be considered as a simple case of weighted voting but with each w_i set to one, $i \in 1, 2, ..., K$.

This approach can be extended to non-binary classification problems by mapping the m-class problem into m binary classification problems and by associating each class j with a separate $S_j(\mathbf{x})$, $j \in 1, 2, ...m$. To classify an instance $\mathbf{x}$, each $S_j(\mathbf{x})$ generates a confidence value indicating the prospect of $\mathbf{x}$ being classified as j versus being classified as non-j. The final class selected corresponds to the $S_j(\mathbf{x})$, $j \in 1, 2, ...m$, with highest confidence value.

In this study, the weights w_i's are set according to the performance (with respect to a selected evaluation metric) of each classifier C_i on the separate validation set V. Other weighted majority algorithms, such as WM and its variants, described in [Littlestone & Warmuth, 1989], determine the weights by assigning, initially, the same value to all classifiers and by decreasing the weights of the wrong classifiers when the collective prediction is false.

2.3.2 Stacking

The main difference between voting and *stacking* [Wolpert, 1992] (also called *class-combiner* [Chan & Stolfo, 1993a]) is that the latter combines base classifiers in a non-linear fashion. The combining task, called a *meta learner*, integrates the independently computed base classifiers into a higher level classifier, the *meta classifier*, by learning over the *meta-level training set*. This meta-level training set is composed by using the base classifiers' predictions on the validation set as attribute values, and the true class as the target. An example of such a set on the thyroid disease [Merz & Murphy, 1996] with three base classifiers is shown in Table 2.2. The first three columns correspond to the predictions (prognosis) of the base classifiers on four patients from a validation set, while the last column represents the correct class (diagnosis). The aim of this strategy is to "correlate" the predictions of the base classifiers by learning the relationship between these predictions and the correct prediction. To classify an unlabeled instance, the base classifiers present their own predictions to the meta-classifier which then makes the final classification. It is worth noting, here, that this final classification may be entirely different from those of the constituent base classifiers.

Other meta-learning approaches that can be considered as variations of the *class-combiner* strategy, include the *class-attribute-combiner* and the *binary-class-combiner* [Chan & Stolfo, 1993a]. Again, the goal is to learn the characteristics and performance of the base classifiers and compute a meta-classifier model of the "global" data set. The two combiner strategies differ from class-combiner in that they adopt different policies to compose their meta-level training sets. The former adds to the meta-level training instances the attribute vectors of the validation set, while the latter, decomposes each prediction

Table 2.2: Meta-learning data set for Stacking

Prognoses of base classifiers			Diagnosis
Classifier-1	Classifier-2	Classifier-3	True Class
Normal	Normal	Hyper	Normal
Hypo	Normal	Hypo	Hypo
Normal	Normal	Hyper	Normal
Normal	Hyper	Hyper	Hyper

of classifier C_i of the original meta-level training set into m binary predictions, where m is the number of classes. Hence, in this case, each prediction, $C_{i_j}(\mathbf{x})$, $j \in 1, 2, ...m$, is produced from a binary classifier trained over examples that are labeled with classes j and $\neg j$. In other words, it is using more specialized base classifiers in an attempt to learn the correlation between the binary predictions and the correct prediction.

2.3.3 SCANN

The third approach considered here is Merz's SCANN (Stacking, Correspondence Analysis and Nearest Neighbor) algorithm for combining multiple models [Merz, 1999]. As with stacking, the combining task integrates the independently computed base classifiers in a non-linear fashion. The meta-level training set is composed by using the base classifiers' predictions on the validation set as attribute values, and the true class as the target. In this case, however, the predictions and the correct class are de-multiplexed and represented in a 0/1 form. In other words, the meta-level training set becomes a matrix of n rows (one per object in the validation set) and $[m \cdot (K + 1)]$ columns of 0 or 1 attributes. There are m columns assigned to each of the K classifiers and m columns for the correct class. If classifier $C_i(\mathbf{x}) = j$, then the j^{th} column of C_i will be assigned a 1 and the rest $(m - 1)$ columns a 0.

SCANN employs correspondence analysis [Greenacre, 1984] (similar to Principal Component Analysis) to geometrically explore the relationship between the validation examples, the models' predictions and the true class. In doing so, it maps the true class labels and the predictions of the classifiers onto a new scaled space that clusters similar prediction behaviors. Then, the nearest neighbor learning algorithm is applied over this new space to meta-learn the transformed predictions of the individual classifiers. To classify unlabeled instances, SCANN maps them onto the new space and assigns them the class label corresponding to the closest class point.

SCANN is a sophisticated combining method that seeks to geometrically uncover the position of the classifiers relative to the true class labels. On the other hand, it relies on singular value decomposition techniques to compute the new scaled space and capture these relationships, which can be expensive, both in space and time, as the number of examples and hence as the number of the base classifiers increases (the overall time complexity of SCANN is $O((M \cdot K)^3)$.

2.3.4 Other Meta-Learning Techniques

For completeness we briefly outline some other popular methods for forming ensembles of classifiers.

Bagging

Bagging [Breiman, 1994] employs sampling techniques to generate many training subsets of different distribution over which it computes multiple models. The method combines the models using unweighted majority voting.

Boosting

Boosting [Freund & Schapire, 1995; Schapire, 1990] learns a set of classifiers in a sequence, where each classifier concentrates on the examples of the training set that are misclassified by its predecessors. Specifically, the algorithm draws examples from the training set according to a probability distribution that reflects their difficulty to be correctly classified (examples have large weights when classifiers misclassify them). The algorithm works iteratively; in each pass it generates one classifier and then updates the weights of the examples according to the performance of that classifier. The final prediction is the weighted sum of the outputs of each classifier of the set according to their accuracy on their training set.

Refereeing

This class of algorithms combines multiple predictions by learning the area of expertise of the individual classifiers. In the hierarchical mixture of experts approach [Jacobs *et al.*, 1991; Jordan & Xu, 1993; R. & J., 1994; Jordan & Jacobs, 1994], for example, the input space is divided into a series of overlapping regions using probabilistic methods, while special *gating functions* are trained to choose between experts (neural network classifiers) over these regions. A similar approach is described in [Ortega, Koppel, & Argamon-Engelson, 1999] where *referee predictors*, in the form of decision trees, provide confidence estimates on the expertise of each base classifier on different sub-domains.

Arbitrating

Arbitration, entails the use of an "objective" judge whose own prediction is selected if the participating classifiers cannot reach a consensus decision. The *arbiter* [Chan & Stolfo, 1993b] is the result of a learning algorithm that learns to arbitrate among predictions

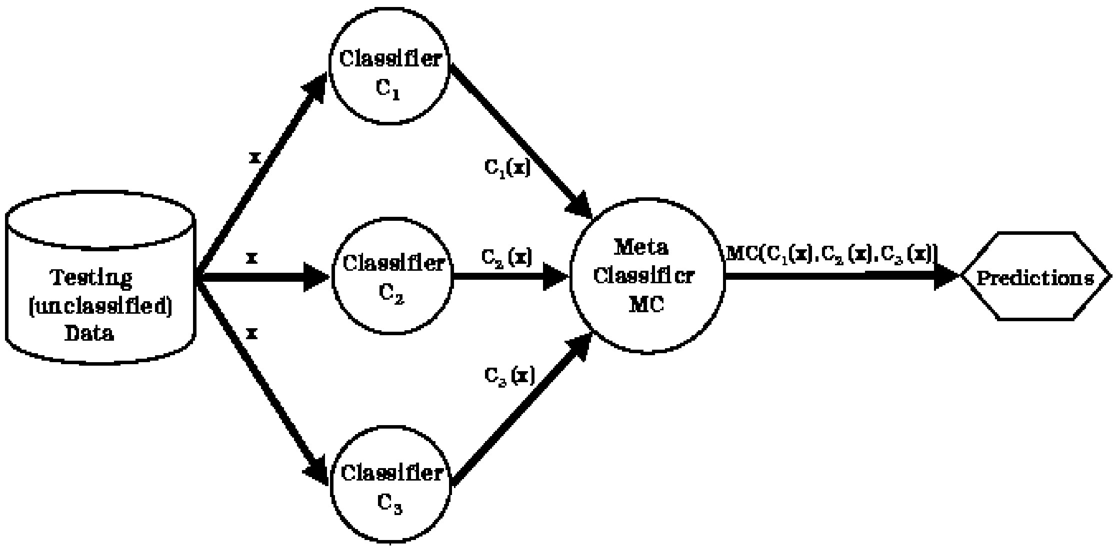

Figure 2.3: Classifying unlabeled instances using the meta-classifier.

generated by different base classifiers. This arbiter, together with an *arbitration rule*, decides a final classification outcome based upon the base predictions.

2.3.5 Meta-Classifying New Instances

The very same meta-level composition rules that are used in the construction of the meta-learning training set are used during classification as well. To generate a prediction on an unlabeled instance the meta-classifier employs its base-level classifiers and its meta-level classifier. The process is illustrated in Figure 2.3.

Initially, unclassified instances $\mathbf{x}$ are supplied as input to each of the base classifiers C_i. Next, the meta-classifier collects their outputs (predictions) $C_i(\mathbf{x})$ and applies the meta-level composition rule (e.g., the class-combiner rule, or the binary-class-combiner rule) to form the meta-level test instance. The generated meta-level test instance is subsequently provided to the meta-level classifier to compute the final classification $MC(\mathbf{x})$ of the original instance.

2.4 Benefits of Meta-Learning

An advantage of *meta-learning* is that it can produce a "higher quality" final classification model, the *meta-classifier*, by combining classifiers with different *inductive bias* (e.g., representation, search heuristics, search space) [Mitchell, 1982]. Thus, by combining separately learned concepts, meta-learning is expected to derive a higher level model that explains a large (distributed) database more accurately than any of the individual learners. Furthermore, it addresses one of the lasting challenges in machine learning, namely, the development of inductive learning techniques that effectively scale up to large and possibly physically distributed data sets. Most of the current generation of learning algorithms are computationally complex and require all data to be resident in main memory, which is clearly untenable for many realistic problems and databases. Notable exceptions include IBM's SLIQ [Mehta, Agrawal, & Rissanen, 1996] and SPRINT [Shafer, Agrawal, & Metha, 1996] decision tree algorithms and Provost and Hennessy's rule-based DRL algorithm [Provost & Hennessy, 1996] for multi-processor learning.

Meta-learning can improve both *efficiency* and *scalability* first by executing the machine learning processes *in parallel* without the time-consuming process of writing parallel programs (i.e., by using standard off-the-shelf serial code) and second, by applying the learning processes on smaller subsets of data that are properly partitioned and distributed to fit in main memory (a *data reduction* technique).

Moreover, it constitutes a unifying and scalable machine learning approach that can be applied to large amounts of data in wide area computing networks for a range of different applications. It is unifying because it is algorithm and representation independent, i.e., it does not examine the internal structure and strategies of the learning algorithms themselves, but only the outputs (predictions) of the individual classifiers, and it is scalable because it can be intuitively generalized to hierarchical multiple level meta-learning.

The literature reports an extensive collection of methods that facilitate the use of inductive learning algorithms for mining very large databases [Dietterich, 1997]. Provost and Kolluri [Provost & Kolluri, 1997] categorized the available methods into three main groups: methods that rely on the design of fast algorithms, methods that reduce the problem size by partitioning the data, and methods that employ a relational representation. Meta-learning can be considered primarily as a method that reduces the size of the data, basically due to its data reduction technique and its parallel nature. On the other hand, it is also generic, meaning that it is algorithm and representation independent, hence it can benefit from fast algorithms and efficient relational representations.

2.5 Evaluation and Confidence

We evaluate and compare the various learning and meta-learning techniques by measuring the performance of the derived classification models on separate data sets. Typically, we select a subset of the available examples S_{train} to train the models and we use the remaining examples S_{test} to evaluate them. The unbiased estimate of the generalization error of the derived classification model C_i is measured by

$$\frac{\text{Number of errors made by } C_i \text{ on } S_{test}}{|S_{test}|}$$

Two classifiers C_A and C_B with different estimates on the generalization error, however, may not necessarily exhibit different predictive performance. To evaluate the two classifiers with confidence, we apply the McNemar's test [Everitt, 1977] on their responses over the separate test set S_{test}. If n_A denotes the number of instances of S_{test} that are misclassified only by C_A and n_B denotes the number of instances of S_{test} that are misclassified only by C_B, the two classifiers would have the same error rate if $n_A = n_B$ (null hypothesis). Thus, we may reject the null hypothesis in favor of the hypothesis that the two classifiers have different performance with α confidence, if

$$\frac{(|n_A - n_B| - 1)^2}{n_A + n_B} \tag{2.2}$$

is greater than $\chi^2_{1,\alpha}$, $0 < \alpha < 1$. For 95% confidence, for example, $\chi^2_{1,0.95}$ is equal to 3.841. Here, χ^2_1 represents the χ^2 continuous distribution with 1 degree of freedom, and the -1 in the numerator corresponds to the "continuity correction" to account for the fact that n_A and n_B are discrete. It has been shown [Dietterich, 1998] that for pre-computed classifiers or for learning algorithms that can be executed only once, McNemar's test is the only test with acceptable Type I error, i.e., the probability of incorrectly detecting a difference when no difference exists.

In general, the result of a learning experiment is affected by one or more sources of variation, such as the selection of the training data, the random variation of the test data, the randomness within a learning algorithm or the random classification error (mislabeled instances). Thus, we prefer to repeat the learning experiment several times to compute reliable results. One of the most commonly used techniques for evaluating the performance of learning programs is *cross validation* [Breiman *et al.*, 1984]. In a k-fold cross validation experiment, the entire data set is divided into k disjoint subsets and k train-and-test runs are performed. At the i^{th} run, the i^{th} subset is used as the test set and the remaining $k-1$ subsets are used as the training set. In every round, the learning algorithms are trained over the training set and the generated classifiers are evaluated against the test set. Note that the two subsets are disjoint, hence the classifiers are evaluated on data not seen during the training phase. In the end, the performance of each learning program is estimated by averaging its results over all k runs. A special case of k-fold cross validation is when $k = n$ (n denotes the size of the data set). This is called *leave-one-out* or sometimes *jackknife*.

To distinguish with confidence between two learning techniques L_A and L_B, using the k-fold cross-validation method, we apply the paired t-test on the differences of the error rates $p_{A,i}$, $p_{B,i}$ of each pair of their classifiers $C_{A,i}$ and $C_{B,i}$ derived in each of the k runs, $1 \leq i \leq k$. Assuming that $p_i = p_{A,i} - p_{B,i}$ and $\bar{p} = \frac{1}{k}\sum_1^k p_i$, the Student's t-test is computed by

$$t = \frac{\bar{p} \cdot \sqrt{k}}{\sqrt{\frac{\sum_{i=1}^{k}(p_i - \bar{p})^2}{k-1}}} \tag{2.3}$$

Under the null hypothesis (the two learning algorithms compute comparable models), t follows a t distribution with $k-1$ degrees of freedom. Thus, for 2-tailed test we can reject the null hypothesis with confidence α if $|t| > t_{k-1,\frac{1+\alpha}{2}}$.

According to [Dietterich, 1998], the k-fold cross validation t-test is quite powerful and highly recommended in cases where Type II error (i.e., failure to detect a real difference between algorithms) is important. On the other hand, it was shown to exhibit a somewhat elevated probability of Type I error, which is attributed to the overlap between any pair of training sets in the k fold cross validation experiment (training sets are not independent). In Chapter 4 we describe a framework that parallels that of a k-fold cross validation experiment, but with the advantage of employing only non-overlapping training sets.

Chapter 3

The JAM System

One of the main objectives of this work is the design and implementation of a system that supports the mining of information from distributed data sets. With meta-learning to provide the means for combining information across separate data sources (by integrating individually computed classifiers), we developed a system called JAM, that facilitates the sharing of information among multiple sites without the need of exchanging or directly accessing remote data. The name JAM stands for Java Agents for Meta-learning; Agents implemented in Java [Arnold & Gosling, 1998] generate and transport the trained classifiers while Meta-learning underlines the key component of the system for combining these classifiers.

The JAM system is designed around the idea of meta-learning to benefit from its inherent parallelism and distributed nature. Recall that meta-learning improves efficiency by executing in parallel the same or different serial learning algorithms over different subsets of the training data set. In this chapter, we describe the distributed architecture of the JAM system and we detail the system design and its scalability, portability, extensibility and adaptivity properties.

3.1 JAM System Architecture

An early version of the architecture of JAM is described in [Stolfo *et al.*, 1997b]. JAM is an agent based system that supports the launching of learning, classifier and meta-learning agents to distributed database sites. It is architected as a distributed computing construct developed on top of OS environments. It can be viewed as a coarse-grain parallel application, with each constituent process running on a separate database site. Under normal operation, each JAM site (i.e., the database site) functions autonomously and (occasionally) exchanges classifiers with the rest.

JAM is implemented as a collection of distributed learning and classification programs linked together through a network of *JAM sites*. Each JAM site consists of:

- one or more local databases,
- one or more learning agents, or in other words machine learning programs that may migrate to other sites as Java objects, or be locally stored as native programs callable by Java agents,
- one or more meta-learning agents, or programs capable of combining a collection of classifier agents,
- a repository of locally computed and imported base- and meta-classifier agents,
- a local user configuration file and,
- a Graphical User Interface and Animation facilities or alternatively a Text-based User Interface.

The JAM sites have been designed to collaborate[1] with each other to exchange classifier agents computed by learning agents. When JAM is initiated, *local or imported learning agents* execute on the *local database* to compute the local classifiers. Each JAM site may then import (remote) classifiers from its peer JAM sites and combine these with its own local classifiers using the *local meta-learning agent*. Finally, once the base and meta-classifiers are generated, the JAM system manages the execution of these modules to classify new unlabeled data sets. Each JAM site stores its base- and meta-classifiers in its *classifier repository*, a special database for classifiers. These actions take place at all JAM sites simultaneously and independently.

The owner (user) of a JAM site administers the local activities via the *local user configuration file*. Through this file, he/she can specify the required and optional local parameters to perform the learning and meta-learning tasks. Such parameters include the names of the databases to be used, the policy to partition these databases into training and testing subsets, the local learning agents to be dispatched, etc. Besides the static[2] specification of the local parameters, the owner of a JAM site can also employ JAM's *graphical user interface* and *animation facilities* to supervise agent exchanges and administer dynamically the meta-learning process. Through this graphical interface, the owner can access more information such as accuracy, trends, statistics and logs and compare and analyze results in order to improve performance. Alternatively, the owner has the option of using a command-driven (text-based) interface to manage the JAM site.

The configuration of the distributed system is maintained by a logically independent module, the Configuration Manager (CM). The CM can be regarded as the equivalent of a domain name server of a system. It is responsible for providing information about the participating JAM sites and for keeping the state of the system up-to-date. The logical

[1]A JAM site may also operate independently without any changes.

[2]Before the beginning of the learning and meta-learning tasks.

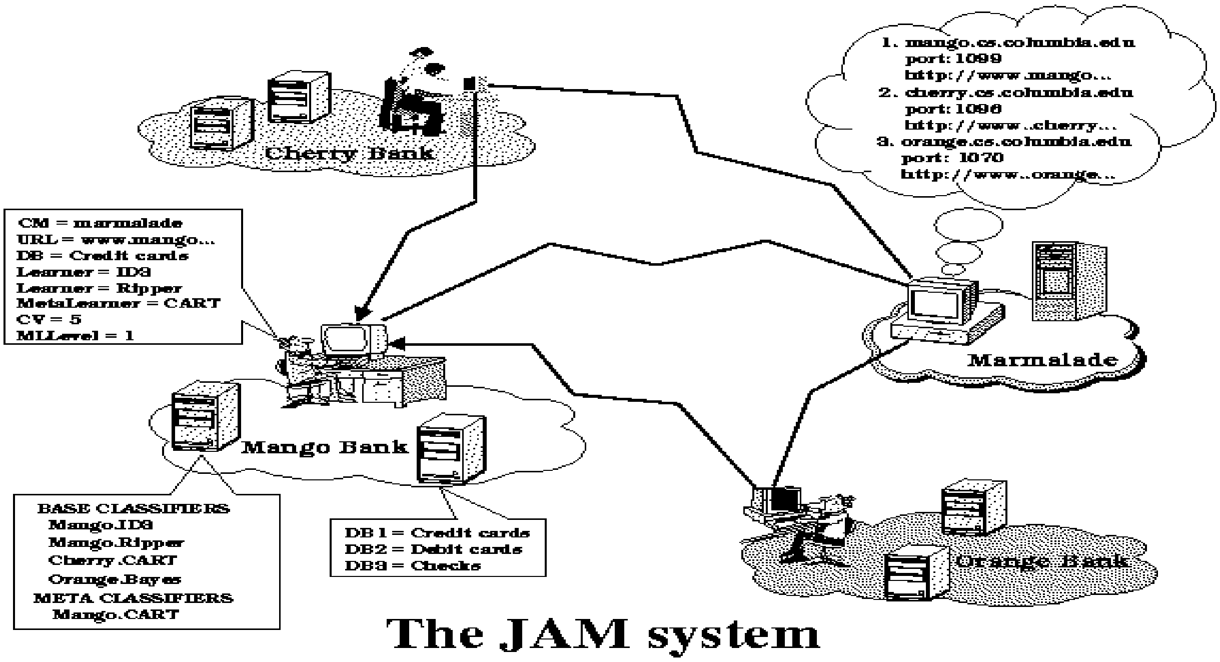

Figure 3.1: The architecture of the meta-learning system.

architecture of the JAM system is presented in Figure 3.1. Notice, the CM runs on Marmalade and three JAM sites Mango Bank, Orange Bank and Cherry Bank exchange their base classifiers to share their local view of the learning task. Mango, for example, has acquired four base classifiers (two are computed locally, one was imported from Orange and one from Cherry) that may be combined in a meta-classifier. The owner of the JAM site controls the learning task by setting the parameters of the user configuration file, i.e., the algorithms to be used, the images to be used by the animation facility, the cross validation and folding parameters, etc.

3.2 Configuration Manager

The Configuration Manager (hereinafter CM) assumes a role equivalent to that of a name server of a network system. The CM provides registration services to all JAM sites that wish to become members and participate in the distributed meta-learning activity. When the CM receives an ACTIVE request from a new JAM site, it verifies both the validity of the request and the identity of the JAM site. Upon success, it acknowledges the request and registers the JAM site as active. Similarly, the CM can receive and verify an INACTIVE request; it notes the requestor JAM site as inactive and removes it from its list of members. The CM, maintains the list of active member JAM sites that seek to establish contact and collaborate with peer JAM sites. By issuing a special QUERY request to the CM, registered JAM sites can obtain this list of active members. Apart from ACTIVE, INACTIVE and QUERY, the CM also supports UPDATE requests that allow JAM sites to change their entries in the list of active members. The complete set of the different type of messages supported by the CM are described in Table 3.1. In addition, the table includes the acknowledgment messages from the CM to the client JAM site requests.

Table 3.1: Types of messages supported by the CM

Message Header	Message body	Direction	Description
JAM_ACTIVE	Identity, contact information	incoming	Join the group
JAM_ACK_ACTIVE		outgoing	Join acknowledged
JAM_INACTIVE	Identity	incoming	Departure notification
JAM_ACK_INACTIVE		outgoing	Departure acknowledged
JAM_QUERY	Identity	incoming	Request list of sites
JAM_ACK_QUERY	List of JAM sites	outgoing	Return list
JAM_UPDATE	Identity, new information	incoming	Change JAM sits entry
JAM_ACK_UPDATE		outgoing	Update successful

Using a single CM within JAM is not a limiting factor to the scalability of the system. The bulk of the communication between the CM and the JAM sites occurs during the initialization stage of each site. On average, a JAM site is expected to issue UPDATE and QUERY requests fairly infrequently. Moreover, the overhead incurred due to the transfer of information between the sites and the CM is minimal. (Each entry in the list of active JAM sites accounts for only a few bytes.)

The CM is a logical unit. Hence, even if the number of participating data sites increases, the CM can be decomposed and distributed across several hosts in a straightforward manner. The architecture follows that of the name servers in a network environment. A single server is responsible for a limited number of network devices; if the address of a device is unknown to a name server, that server will contact another server in an attempt to resolve the name.

The current implementation of JAM defines a CM that provides registration and membership services to each JAM site. Future extensions of the CM include the support of multiple groups of sites, of varying levels of "visibility" (e.g., some sites may not be allowed to get access information about every other JAM site - a similar approach to access/capability lists between users and resources), of authentication capabilities, directory services of databases and learning and classifier agents, fault tolerance, etc.

3.3 JAM Site Architecture

Unlike the CM, which provides a passive configuration maintenance function, the JAM sites are the active components of the meta-learning system. They manage the local databases, obtain remote classifiers, build the local base and meta classifiers and interact

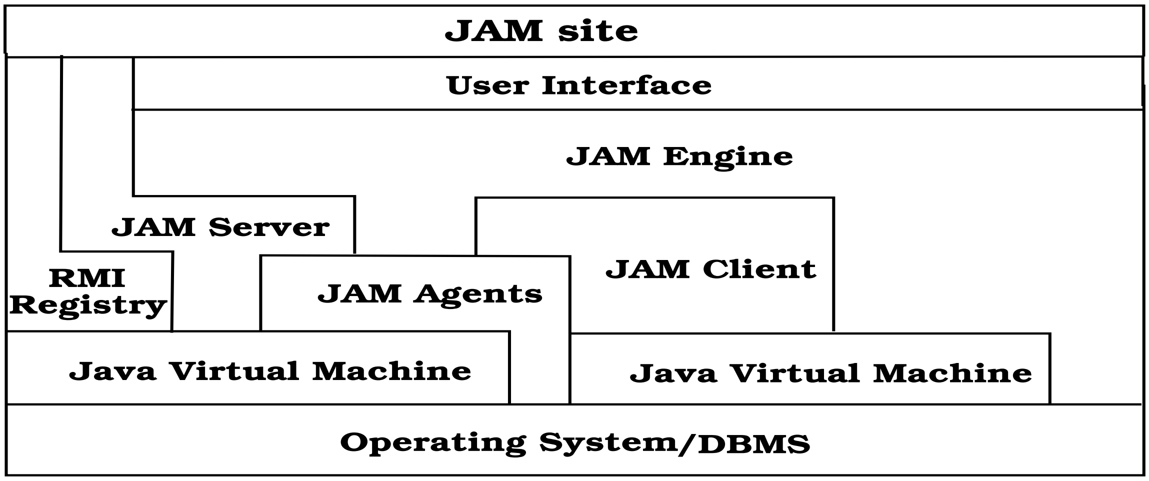

Figure 3.2: JAM site layered model.

with the JAM user. JAM sites are implemented as multi-threaded Java programs with a special GUI.

Each JAM site is organized as a layered collection of software components shown in Figure 3.2. In general, the system can be decomposed into four separate subsystems, the User Interface, the JAM Engine and the Client and Server subsystems. The User Interface (upper tier) materializes the front end of the system, through which the owner can define the data mining task and drive the JAM Engine. The JAM Engine constitutes the heart of each JAM site by managing and evaluating the local agents, by preparing/processing the local data sets and by interacting with the Database Management System (DBMS), if one exists. Finally, the Client and Server subsystems compose the network component of JAM and are responsible for interfacing with other JAM sites to coordinate the transport of their agents. Each site is developed on top of the JVM (Java Virtual Machine), with the possible exception of some agents that may be used in a native form and/or depend on an underlying DBMS. A Java agent, for instance, may be able to access a DBMS through JDBC (Java Database Connectivity). The RMI registry component displayed in Figure 3.2 corresponds to an independent Java process that is used indirectly by the JAM server component and is described later.

3.3.1 User Interface and JAM Engine Components

Upon initialization, a JAM site undertakes a series of tasks; it starts up the GUI on a separate thread; it registers with the CM; it instantiates the JAM Client and finally spawns the JAM Server thread for listening for requests/connections from the outside. The necessary information to carry out these tasks (e.g., the host name and the port number of the server socket of the CM, required URLs, the path names to local agents and data sets, etc.) is maintained in the local user configuration file and is administered by the owner of the JAM site.

JAM sites are event-driven systems; they wait for the next event to occur, either a command issued by the owner via the GUI, or a request from a peer JAM site via the JAM Server. Such tasks can be any of JAM's functions, from computing a local classifier and starting the meta-learning process to sending the local classifiers to peer JAM sites, to requesting remote classifiers from other sites or to reporting the current status and presenting computed results. GUI commands can either be single-action instructions (e.g., partition the data set into training and test sets under specific constraints) or batch-mode instructions (e.g., perform a 10-fold cross validation meta-learning experiment).[3] A GUI command activates the JAM Engine, which will subsequently translate, verify the validity of the input and execute the command. Depending on the nature of this command, the JAM Engine may, in turn, call the JAM Client. For example, on an "import and meta-learn remote classifier agents" command, the JAM Engine would rely on the JAM Client component to obtain the remote classifier agents. The status of the system and the final outcome of the actions of the JAM Engine are returned and reported to the owner through the GUI.

Figures 3.3 and 3.4 present two snapshots of the JAM system. In this example, three JAM sites, Marmalade, Strawberry and Mango collaborate in order to share and improve their performance in diagnosing thyroid-related problems [Merz & Murphy, 1996]. Both snapshots are taken from "Marmalade's point of view". Figure 3.3 shows one JAM site (Marmalade) exchanging agents with the two other JAM sites (Mango and Strawberry), while Figure 3.4 displays the system during the meta-learning phase. Notice that Marmalade has established that Strawberry and Mango are its potential peer JAM sites by acquiring information through a QUERY request to the CM.

The right side of panel of the GUI keeps information about the current stage of the system and displays the settings of several key parameters, including the Cross-Validation fold, the Meta-Learning fold (i.e., the data partitioning scheme used in the meta-learning stage), the Meta-Learning level, the names of the local learning and meta-learning agents, etc. The bottom part of the panel logs the various events, and records the current status of the system. In this instance, the Marmalade JAM site partitions the thyroid database into the thyroid.1.bld and thyroid.2.bld data subsets according to the 2-fold Cross Validation Scheme. During the learning phase of the first fold, Marmalade computes the local classifier Marmalade.1 by applying the ID3Learner agent on thyroid.1.bld. Next, it imports the remote classifiers, noted by Strawberry.1 and Mango.1 and begins the meta-learning process. In this experiment, each site contributes a single classifier agent. During the meta-learning phase of the first fold, Marmalade applies the three base classifier agents Mango.1, Marmalade.1 and Strawberry.1 on the thyroid.1.bld data subset using the 2-

[3]The Text-based user interface provides a similar, albeit more limited set of commands. A fine control of the JAM site, however, is still possible through the local user configuration file.

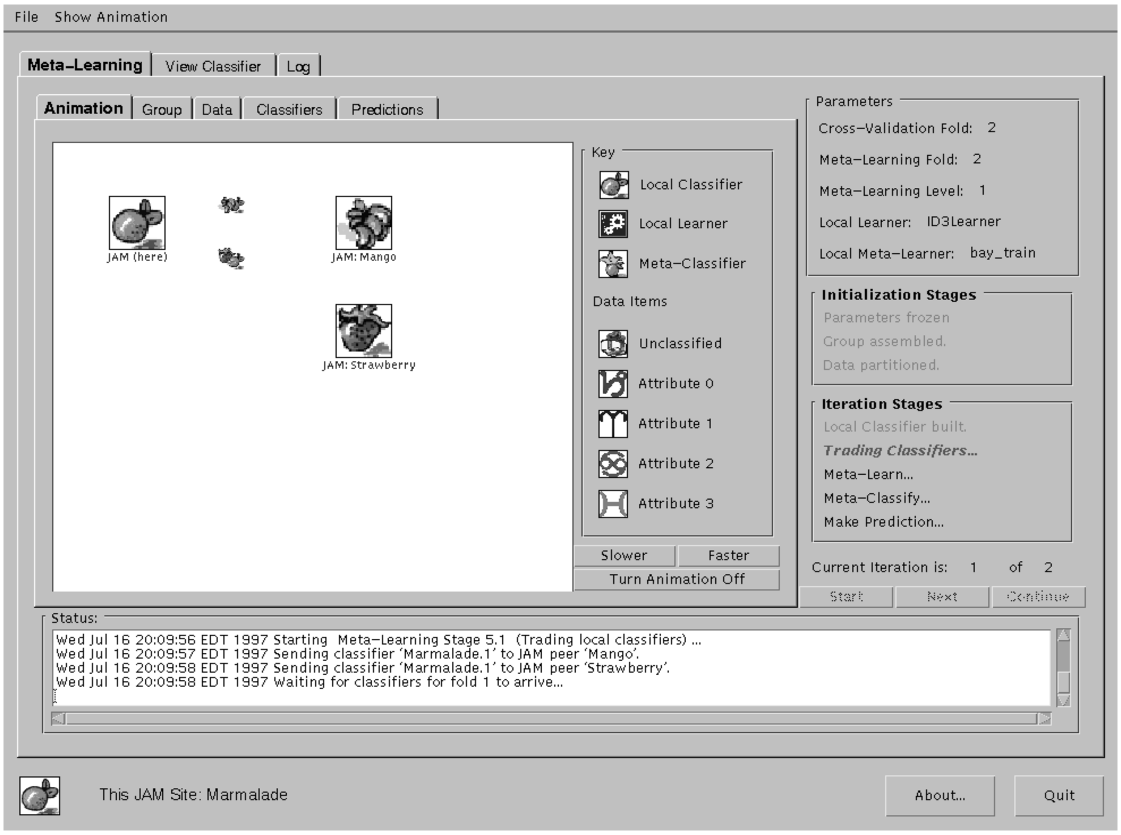

Figure 3.3: Snapshot of the JAM system in action: Marmalade is trading classifiers with the Strawberry and Mango JAM sites (trading classifiers stage).

fold meta-learning scheme, to generate the meta-level training set. The final ensemble meta-classifier, noted as Meta-Classifier.1 is computed via the stacking method using the "native" bay_train Bayesian learning algorithm over this meta-level training set.

Marmalade will employ the Meta-Classifier.1 to predict the classes of the thyroid.2.bld test set as dictated by the 2-fold Cross Validation evaluation scheme. If Cross Validation was set to one, the JAM site would use Meta-Classifier.1 to classify single data instances (in this case unlabeled medical records), or optionally label a separate test set provided by the owner.

The snapshots of Figures 3.3 and 3.4 display the system during the animated trading and meta-learning processes, where JAM's GUI moves icons within the Animation Tabbed folder of the JAM site displaying the construction of the new meta-classifier. Detailed information (not shown here) about the participating JAM sites and the local thyroid data sets are found inside the Group tabbed folder and the Data tabbed folder respectively. In addition, the User Interface provides a Classifier Tabbed folder and a Predictions Tabbed folder. The Classifier Tabbed folder (presented later in more detail)

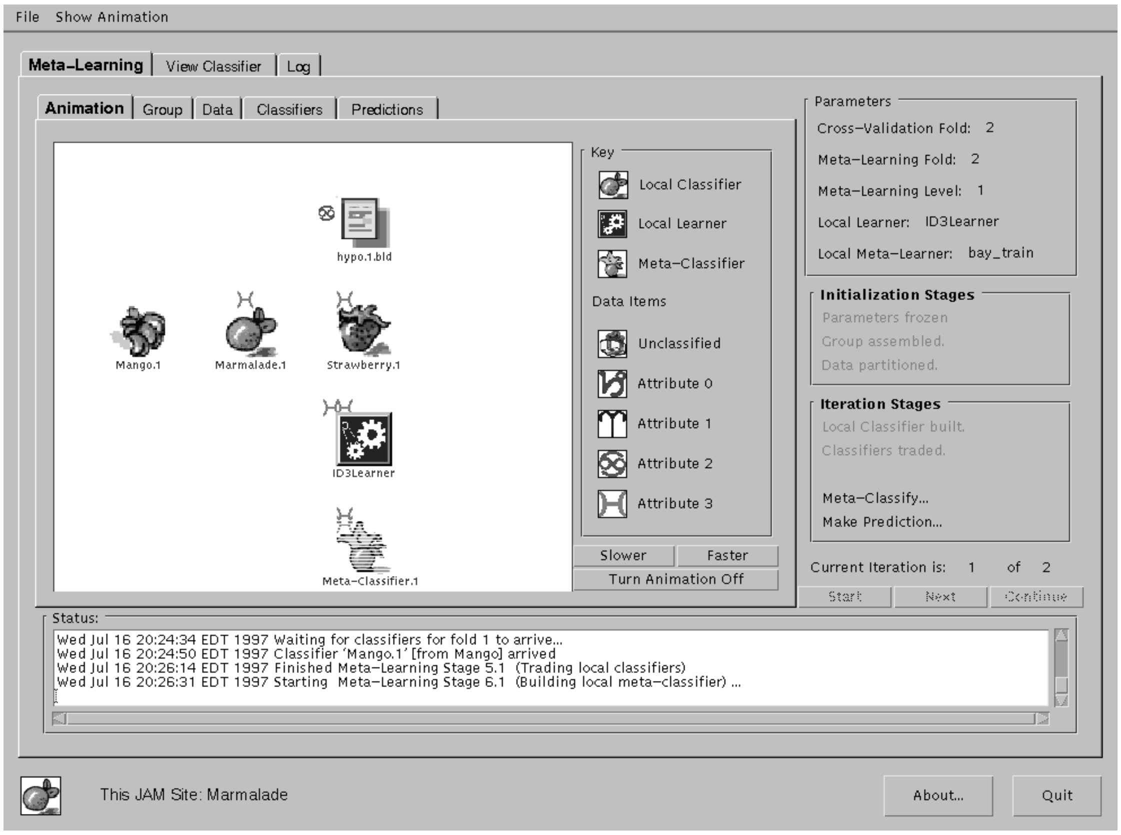

Figure 3.4: Snapshot of the JAM system in action: Marmalade is building the meta-classifier (meta-learning stage).

allows the owner of the JAM site to study the base- and meta-classifiers more closely, while the Predictions Tabbed folder lets him/her administer the test phase, e.g., subject the various models in batch testing (generate predictions on multiple test instances of a test file) or single testing (classify one example at a time) to evaluate the performance of the derived classifiers and meta-classifiers.

Animation

For demonstration and didactic purposes, the Animation Tabbed Folder panel of the JAM Graphical User Interface contains a collection of animation panels that visually illustrate the stages of meta-learning in parallel with execution. When animation is enabled, a transition into a new stage of computation or analysis triggers the start of the animation sequence corresponding to the underlying activity. The animation loops continuously until the given activity ceases.

The JAM program gives the user the option of manually initiating each distinct

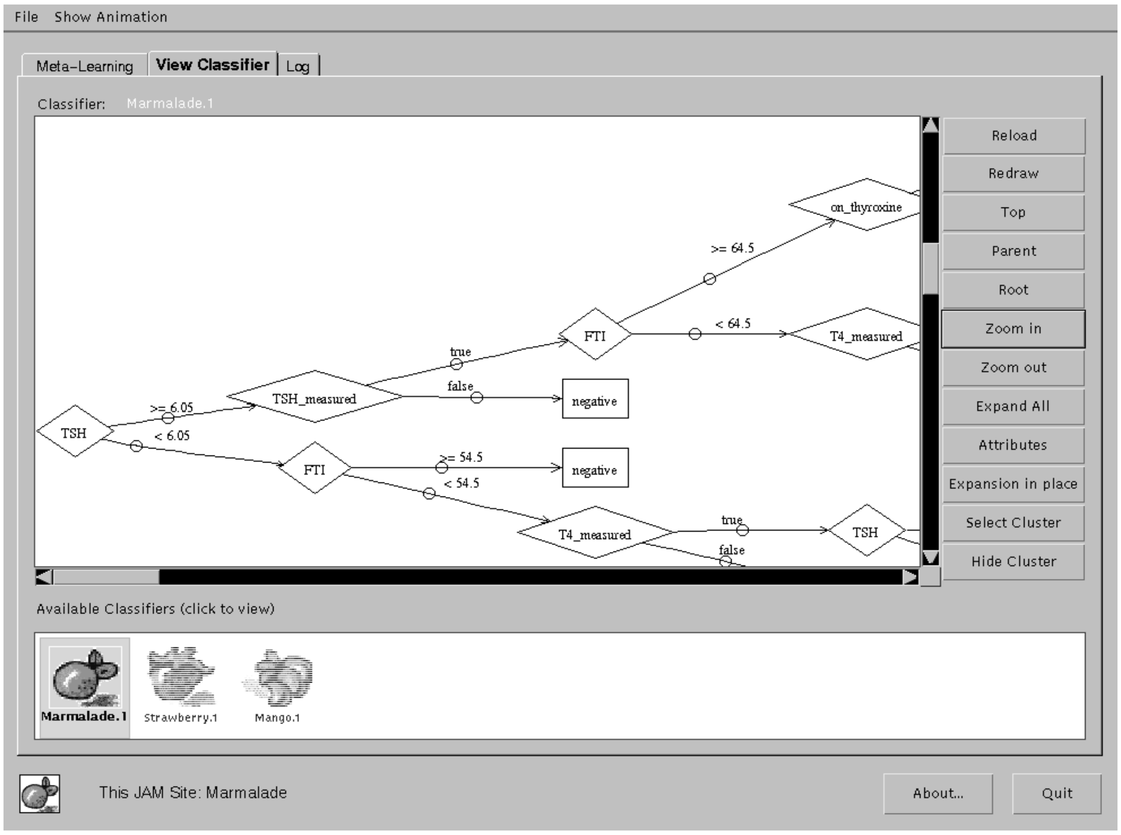

Figure 3.5: Snapshot of the JAM system in action: An ID3 tree-structured classifier is being displayed in the Classifier Visualization Panel.

meta-learning stage (by clicking the *Next* button), or sending the process into automatic execution (by clicking the *Continue* button).

Classifier Visualization

JAM provides graph drawing tools to help users understand the learned knowledge [Fayyad, Piatetsky-Shapiro, & Smyth, 1996]. There are many types of classifiers, e.g., decision trees by ID3, that can be represented as graphs. In JAM we visualize decision tree-type classifiers by employing major components of Grappa [Lee, Barghouti, & Moccenigo, 1997], an extensible visualization system that allows users to display and analyze graphs. Since each machine learning algorithm implementation has its own format to represent the learned classifier, JAM relies on algorithm-specific translators to read the classifier and generate the Grappa-graph representation.

Figures 3.5 and 3.6 show two snapshots of the JAM Classifier Tabbed folder panel (also called JAM Visualization Panel) with a decision tree, where the leaf nodes represent

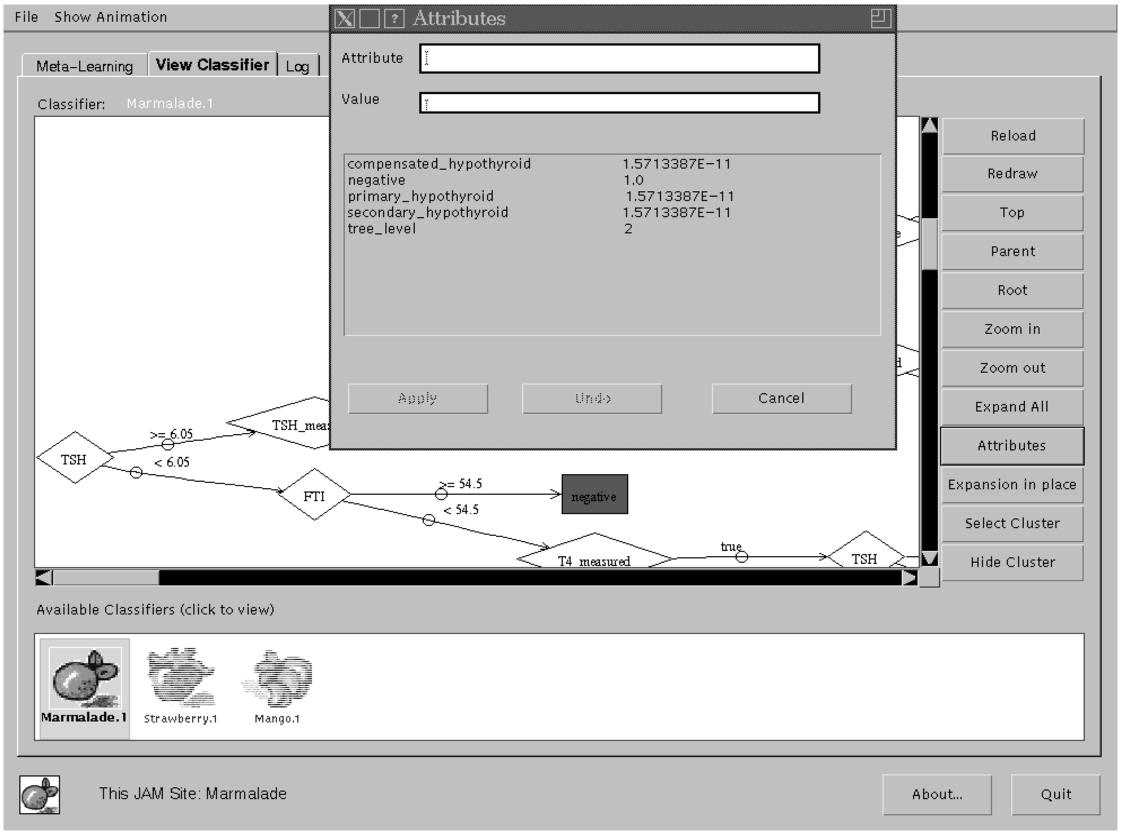

Figure 3.6: Snapshot of the JAM system in action: Classification information of a leaf of an ID3 tree-structured classifier is shown by pressing the *Attributes* button.

classes (decisions), the non-leaf nodes represent the attributes under test, and the edges represent the attribute values. The owner can click on the *Attributes* button from the menu of Figure 3.5 to see any additional information about a node or an edge (e.g., as shown in Figure 3.6). In Figure 3.6, the *Attributes* window shows the classifying information of the highlighted leaf node. In this case, we can see that for a test data item, if its "TSH" value is less that 6.05 and its "FTI" value is greater or equal to 54.5, then it belongs to class "negative" with 1.0 probability.

When it is difficult to clearly view a very large graph (that has a large number of nodes and edges) due to the limited window size, the Classifier Tabbed Folder panel provides commands for the owner to traverse and analyze parts of the graph: he/she can select a node and use the *Top* button from the *Graph* menu to make the subgraph starting from the selected node be the entire graph in display; use the *Parent* button to view the enclosing graph; and use the *Root* command to see the entire original graph.

Some machine learning algorithms generate concise and readable textual outputs, e.g., the rule sets from Ripper [Cohen, 1995]. It is thus counter-intuitive to translate the

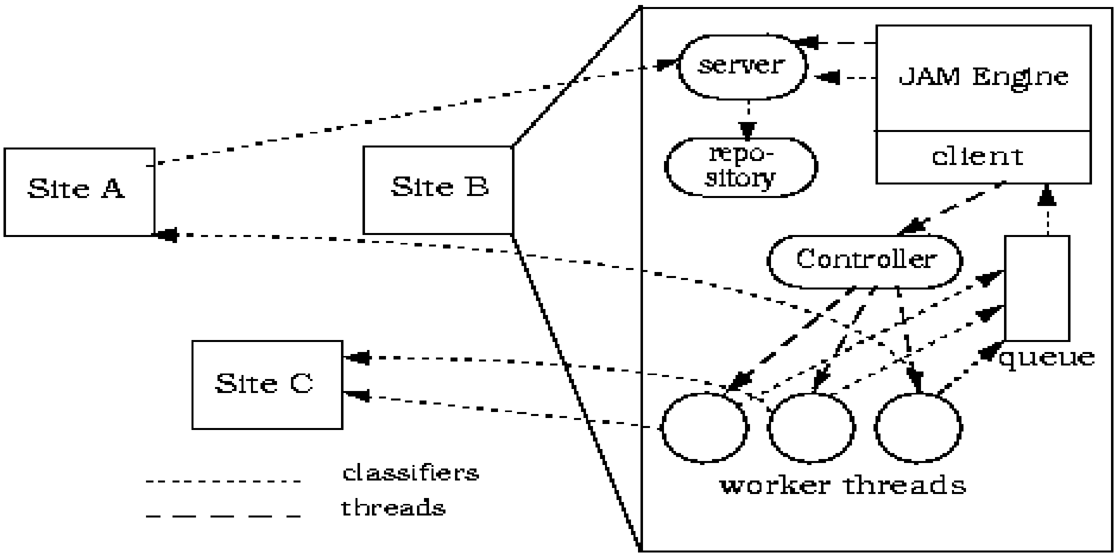

Figure 3.7: JAM as a Client-Server architecture.

text to graph form for display purposes. In such cases, the algorithm-specific translator can simply format the text output and display it in the classifier visualization panel.

3.3.2 JAM Client and JAM Server Components

The JAM sites are designed to work in parallel and autonomously. In particular, the JAM system is architected as a collection of loosely coupled processes (the JAM sites), each performing its own local data mining (in this case, learning/classification) and occasionally collaborating with its peer processes to import or export local classifiers. The design follows that of a client-server architecture. Specifically, each JAM site can operate simultaneously as a Client site requesting learning or classifier agents from remote servers and as a Server site responding to similar requests from other sites.

To avoid synchronization barriers and minimize busy-wait scenarios, both the Client and the Server components are implemented in a multi-threaded fashion. Figure 3.7 shows JAM site B acting as a Client to sites A and C and as a Server for site A. In this example, the JAM Engine of site B instructs the JAM client to obtain three remote classifier agents, one from JAM Site A and two from JAM Site C. To service the request, the Client spawns a main *Controller* thread that creates a local *Queue* (i.e., a buffer) for storing the results (e.g., the returned classifiers) and spawns, in turn, three *Worker* threads, one for each classifier agent. The benefit of this design is that the JAM sites are capable of issuing multiple requests to their peer JAM sites in parallel. Upon completion, each *Worker* thread obtains the lock of the *Queue*, inserts the result into the *Queue* and releases the lock. Every t seconds, currently set at 5 seconds, the *Controller* thread obtains the lock of the *Queue* and conveys any returned results to the JAM Client. When the complete set of results is available, the JAM Client provides it to the JAM Engine, which continues with normal execution.

Besides collecting these results, the *Controller* thread is responsible for monitoring its *Worker* threads' progress. To account for the probability of site failures and network outages, for example, the *Controller* thread imposes a hard limit as to how long it may wait for a response from its *Worker* threads. Any *Worker* thread violating this limit is deemed blocked and is killed. In such a case, the *Controller* thread and, subsequently, the JAM Client provide to the calling JAM Engine an appropriate error code along with the partial set of results.

At the opposite end, JAM Servers are responsible for satisfying requests. As with the JAM Client, the JAM Server is also multi-threaded to support multiple calls simultaneously, both local (e.g., from the JAM Engine), and remote (from other JAM Sites). The most recent version of the JAM Server is built upon the existing Remote Method Invocation (RMI) technology offered as part of Sun's Java package. As the name implies, RMI enables the invocation of methods of remote Java objects from other virtual machines, possibly on different hosts. By integrating RMI into JAM and by defining the set of object methods exported by the JAM Servers, we are able to specify the communication protocol among sites. Then we materialize it via remote object method calls from the JAM Clients to the JAM servers. Remote object methods are invoked by JAM Clients through references provided by the *RMI registry* [Arnold & Gosling, 1998].

An *RMI Registry* corresponds to a name server at the server side that allows remote clients to get a reference to server objects. Typically, there is one *RMI Registry* for every JAM site. The *RMI Registry* and the JAM site run as separate processes sharing the same host machine. Upon initialization, the JAM server uses the *RMI Registry* to bind its list of available objects to names. Subsequently, a JAM Client can access and lookup up the server objects at the *RMI Registry* based on the Uniform Resource Locator (URL), and invoke the server methods as needed. In addition to being a clean and straightforward approach, RMI provides the additional benefit of being extensible; by allowing the JAM Servers to define and export additional methods through the *RMI Registry*, the communication protocol can be extended to support new functionality. The interface (the server object methods) provided as part of the current design of the JAM server is presented in Table 3.2.

The first four rows of the table contain the necessary and sufficient methods that need to be defined by a JAM Server. The first two methods provide the means for a JAM Client to access remote database information, whereas the next two rows describe the methods for requesting the list of available agents and obtaining the desirable remote learning or classifier agents. The design of the interface of the JAM Server, however, is extended with additional methods to allow alternative, more flexible and easier use, i.e., it provides methods for requesting the names of the base-learning and meta-learning

Table 3.2: Interface published by the JAMServer

Method call	Method parameters	Return result
JAMGetDBDirectory		Vector of local database names
JAMGetDBProperties	DBName	Schema description
JAMGetAgentDirectory	DBName	Directory of local agents
JAMGetAgent	TimeStamp	Single (Learner/Classifier) agent
JAMGetBaseLearnersNames		Vector of BaseLearners' names
JAMGetBaseLearners	LearnerNames, TimeStamp	Vector of BaseLearners
JAMGetMetaLearnersNames		Vector of MetaLearner' names
JAMGetMetaLearners	MetaLearnerNames, TimeStamp	Vector of MetaLearners
JAMGetClassifiers	DBName, AlgorithmNames, IsMeta, FoldNumber, TimeStamp	Vector of Classifiers

algorithms, for acquiring the base-learning and meta-learning agents and for obtaining the needed classifier agents.

The learning and classifier agents are uniquely identified by the *TimeStamp* index, i.e., an index created at the instant each agent is inserted in the JAM Site repository (discussed later in more detail). Besides the *TimeStamp* index other information associated with each agent include:

1. the name of learning algorithm,
2. the cross validation fold number (zero for learning agents),
3. a boolean parameter distinguishing whether it is a base-level of meta-level agent and
4. the name of the database over which it is computed (only for classifier agents).

The JAM Server is designed to provide to a JAM Client all agents that match the parameters of the calling methods. For instance, if *DBName* is set to thyroid, *IsMeta* is set to false, and *FoldNumber* is set to one, and both *AlgorithmNames* and *TimeStamp* are set to null, the *JAMGetClassifiers* method will return all base classifiers computed over the thyroid database under the first cross-validation fold, independently of the learning algorithm or the time they were created. An error code and a null vector are returned in case the input parameters of a remote method call are conflicting.

To avoid exposing the wrong agents when confidentiality issues and distribution rights are of matter, we followed the conservative approach and designed the JAM Server to export only its local learning and classifier agents and not any agents acquired from other sites. Nevertheless, it is easy to relax these constraints, if required, by extending the *TimeStamp* index to include the name of the remote site from where an agent originates. This change would enable JAM Clients to index and obtain any agent that resides at a particular JAM Server, regardless of its being remote (obtained from a peer JAM Server) or local to that Server.

Each JAM Server interacts with the local *Repositories* that maintain the agents and make them available as required. The JAM Engine instantiates a separate *Repository* object for each local data set, i.e., for each *DBName.* The *Repository* consists of a database of local (introduced/installed by the owner) and remote (transferred from another site) learning agents, and local and remote classifier agents. By local classifier agents we mean the classifiers computed over a local data set by local or remote learning agents; by remote classifier agents we denote the classifiers derived over remote data either by remote learning agents or by local learning agents that migrated at the remote site. A learning agent can represent either a base-learning algorithm or a meta-learning technique. Similarly, a classifier agent can either be a single base-classifier or a meta-classifier that combines multiple classifier agents.

Table 3.3: JAM site Repository Interface

Method call	Method parameters	Description
JAMInsert	JAMsite, AlgorithmName, isMeta, FoldNumber, TimeStamp	Add an agent to the Repository
JAMDelete	JAMSite, TimeStamp	Remove an agent from the Repository
JAMGetAgent	JAMSite, TimeStamp	Return a specific agent (Learner/ Classifier) (Learner or Classifier)
JAMLoad	URL to storage location	Populate the Repository with existing agents from previous runs
JAMGetLearner	JAMSite, TimeStamp	Return a specific Learner agent
JAMGetClassifier	JAMSite, TimeStamp	Return a specific Classifier agent
JAMSelect	JAMSite, AlgorithmName, FoldNumber, isMeta	Return a vector of agents that match the input parameters

The *Repository* supports a small number of primitives for accessing and updating the available agents, as described in Table 3.3. Each entry in the database, i.e., a learning or a classifier agent, is indexed by the name of its originating JAM site and a time stamp created upon entrance into that database. Other attributes defined for each entry in the *Repository* include the name of the learning algorithm, the fold number that generated a specific classifier agent in a k-fold Cross-Validation experiment (this number is set to zero for learning agents) and the boolean attribute *isMeta* that distinguishes base-level from meta-level agents.

JAMInsert, JAMDelete and *JAMGetAgent* are the main primitives for adding, removing and retrieving agents. Similar to the JAM Server interface, however, the *Repository* provides a second set of primitives to support additional functionality; *JAMLoad* provides the means to populate the *Repository* with existing agents that were computed and stored during past executions of the JAM system; *JAMGetLearner* and *JAMGetClassifier* define alternative methods to access the learning and classifier agents respectively; and finally, *JAMSelect* returns all agents that match the parameters of the calling method.

3.4 Agents

JAM's extensible plug-and-play architecture allows snap-in learning agents. The learning and meta-learning agents are designed as objects. JAM provides the definition of an abstract parent agent class and every instance agent object (i.e., a program that implements a learning algorithm ID3, Ripper, CART [Breiman *et al.*, 1984], Bayes [Duda & Hart, 1973], Wpebls [Cost & Salzberg, 1993], CN2 [Clark & Niblett, 1989], etc.) is then defined as a subclass of this parent class. Through the variables and methods inherited by all agent subclasses, the parent agent class describes a simple and minimal interface that all subclasses have to comply to. As long as a learning or meta-learning agent conforms to this interface, it can be introduced and used immediately as part of the JAM system. To be more specific, a JAM learning agent needs to implement the following methods:

1. A *constructor* method with no arguments. The JAM Engine calls this method to instantiate the agent, provided it knows its name (it is supplied by the owner of the JAM site through the local user configuration file or the GUI).

2. An *initialize* method. In most of the cases, the sub-classed agents inherit this method from the parent agent class. Through this method, the JAM Engine supplies the necessary arguments to the agent including the name of the training data set, the name of the dictionary file (also known as attribute file), and the file name of the output classifier, if required.

3. A *buildClassifier* method. The JAM Engine calls this method to trigger the agent to learn (or meta-learn) a classifier from the training data set.

4. A *getCopyOfClassifier* method. This method is used by the JAM Engine to obtain the newly built classifier. The derived Classifier, a Java object itself, can be subsequently transferred and "snapped-in" at any participating JAM site. Hence, remote agent dispatch is easily accomplished.

5. Additional methods, such as *getAlgorithmName*, *getDBName*, *getDictionaryExtension*, etc. that facilitate the access of agent-specific and task-specific information. These methods are defined at the Learner class level and inherited by the sub-classed agents.

The class hierarchy (only methods are shown) for five different learning agents is presented in Figure 3.8. ID3, Bayes, Wpebls, CART and Ripper re-define the *buildClassifier* and *getCopyOfClassifier* methods but inherit the *initialize* method from their parent learning agent class as well as the methods for acquiring specific information (e.g., *getAlgorithmName*.

Due to this design, no task- or algorithm-specific information (such as the name of the algorithm, program options, input and output parameters, etc) is present in the

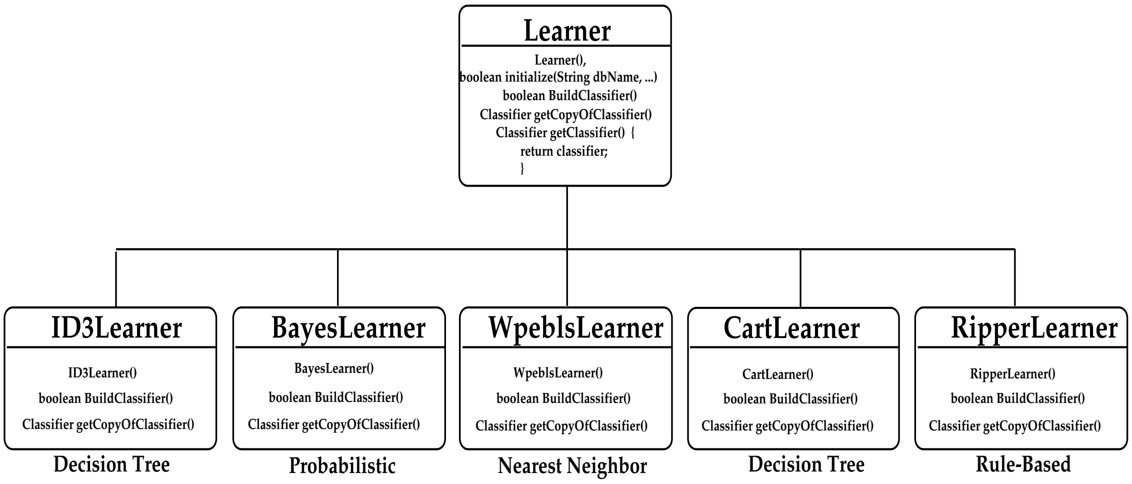

Figure 3.8: The class hierarchy of Learning agents.

source code of the JAM Engine. As a result, the system need not be re-compiled if a new algorithm is introduced. Instead, the JAM Engine can access an agent by calling the redefined methods of the instantiated sub-classed objects of the abstract parent Learner class. The additional methods (e.g., *getAlgorithmName*, etc) described earlier, are defined as a means to expose information that is specific to each sub-classed object (e.g., the name of the learning algorithm).

The abstract MetaLearner class follows the Learner class design, but with the addition of the *prepareMLSet* method.[4] This method implements the generic function of composing the meta-level training set based on the predictions of the base classifiers and the validation set (as described in Chapter 2). Different meta-learning schemes, such as Stacking, Voting, SCANN, etc, can be introduced in JAM by sub-classing the MetaLearner class and by defining the *buildMetaClassifier* method and inheriting or redefining (if needed) the *prepareMLSet* method.

Base- and meta-classifiers are defined as Java objects as well. JAM provides the definition of the abstract parent Classifier agent class and every instance agent object (base-classifier of meta-classifier) is defined as a subclass of this parent class. A Classifier agent is the product of a Learner or MetaLearner agent when applied to a data set. As with the Learner and MetaLearner classes, as long as a Classifier agent conforms to the specific interface, it can be introduced and used immediately as part of the JAM system. Specifically, a JAM Classifier agent needs to implement the following methods:

1. A *constructor* method. A sub-classed object of the Learner class calls this method to instantiate an object of the corresponding Classifier subclass.

[4]Besides the *prepareMLSet*, the MetaLearner class defines an extra *baseClassifiers* data field that corresponds to the vector of classifiers combined by the meta-learning algorithm.

2. A *getClassifierEngine* method. It returns an object of the ClassifierEngine class, that is subsequently used to classify new examples. More specifically, the ClassifierEngine class provides the *classifyFile* method for generating batch predictions on a test set and the *classifyItem* method to classify a single instance.

   The ClassifierEngine object is made part of Classifier to accommodate a number of existing learning programs of the public domain that require that a data dictionary accompanies each training or test set. This requirement compels Classifier agents to read the data dictionary multiple times when classifying multiple single instances. The ClassifierEngine object, allows the decoupling of the parsing of the data dictionary information and classification process, thus making it possible to read data dictionaries only once.

3. A *displayClassifier* method. It is defined by each sub-classed Classifier agent and is tailored to the specific representation of the learning algorithm and the particular implementation. The method is called from within the Classifier Visualization panel when the owner seeks to study the internal of the Classifier agent.

4. An *isMetaClassifier* method. It is used to distinguish between base-classifiers from meta-classifier agents.

5. A *setBaseClassifiers* and a *getBaseClassifiers* methods for populating and retrieving the base Classifier agents from the baseClassifiers vector of the meta-classifiers. For base classifiers, both methods return null values.

6. Additional methods such as *getOriginatingJAMSite*, *getDBName*, *getCVFold*, etc. that provide detailed information regarding the origin and the conditions a classifier was computed.

The class hierarchy (only methods are shown) for five different classifier agents (base- and meta-classifiers) is presented in Figure 3.9. ID3 and Bayes, represent base-classifier objects while Stacking, Voting and SCANN correspond to meta-classifier objects. All subclasses re-define their constructors and the algorithm-specific *getClassifyEngine* and *displayClassifier* methods, but inherit other methods such as *isMetaClassifier*, *getBaseClassifiers*, *getOriginatingJAMSite*, etc. The definitions of the ClassifierEngine class that is used by Classifier objects follow a similar approach. For each Classifier subclass, a ClassifierEngine subclass tailors its *classifyFile* and *classifyItem* methods to execute its own base- or meta-classification scheme.

The Learning and Classifier agents are transferred among the various data sites using Java's Object Serialization capabilities [Arnold & Gosling, 1998]. Object Serialization extends Java's Input and Output classes with support for objects by marshaling and unmarshaling them to and from a stream of bytes, respectively. To efficiently transport

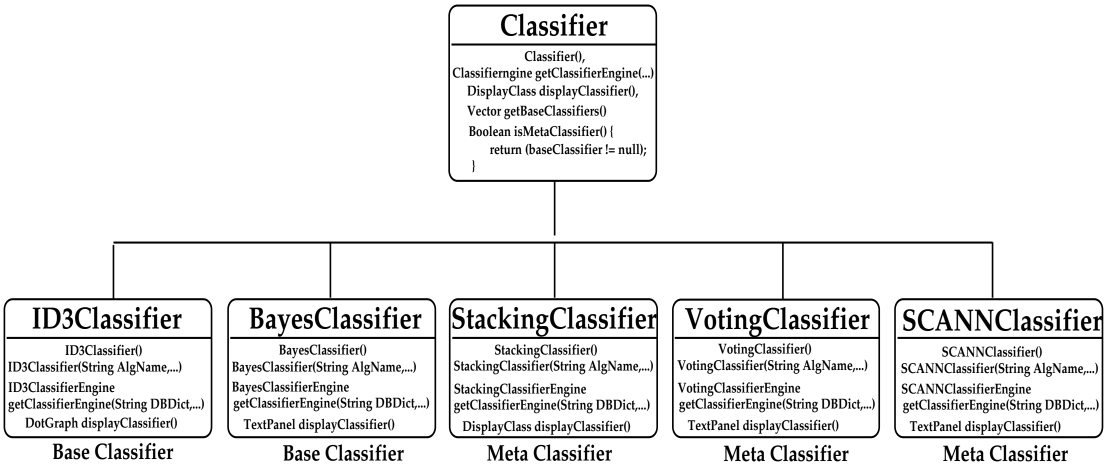

Figure 3.9: The class hierarchy of (base- and meta-) Classifier agents.

Classifier agents across JAM sites, we overrode the default object serialization mechanism by customizing the *writeObject* and *readObject* methods for each agent subclass. Methods *writeObject* and *readObject* are part of Java's ObjectOutputStream and ObjectInputStream class definitions respectively for serializing and de-serializing a given object through an RMI or socket connection.

3.5 Portability

We have used Java technology to build the infrastructure and the various components of the JAM system, including the specific *agent operators* that compose and spawn new agents from existing learning agents, the implementation of the User Interface (Graphical and Text-based), the animation facilities and most of the machine learning algorithms and the classifier and meta-learning agents.

Java provides the means to develop a system that is capable of operating under different hardware and software configurations (e.g., across the Internet), as long as the Java Virtual Machine (JVM) [Lindholm & Yellin, 1999] is installed on these environments. Moreover, by adopting the meta-learning framework as the unifying machine learning approach, JAM constitutes an algorithm independent data mining system. Meta-learning has the advantage of not being constrained to any specific representation, internal structures or strategies of the learning algorithms, but only to the output (predictions) of the individual classifiers. The learning agents are the basic components for searching for patterns within the data and the classifier agents are the units that capture the computed models and can be shared among the data sites. The platform independence of Java and the algorithm independence of meta-learning make it easy to port JAM and delegate agents to participating sites. As a result, JAM has been successfully tested on the most popular platforms including Solaris, Windows and Linux simultaneously, i.e., JAM sites can import and utilize classifiers that are computed over different platforms.

In cases where Java's computational speed is of concern, JAM is designed to also support the use of native (e.g., C or C++) learning algorithms to substitute slower Java implementations, a benefit stemming from JAM's extensible design. Native learning programs can be embedded within appropriate Java wrappers to interface with the JAM system and can subsequently be transfered and executed at a different site, provided, of course, that both the receiving site and the native program are compatible.

3.6 Extensibility

The independence of JAM from any particular learning or meta-learning method, in conjunction with the object oriented design ensure the system's capability to incorporate and use new algorithms and tools. As discussed in Section 3.4 introducing a new technique requires the sub-classing of the appropriate abstract class and the encapsulation of the tool within an object that adheres to the minimal interface. In fact, most of the existing implemented algorithms have similar interfaces already.

This plug-and-play characteristic makes JAM a powerful and extensible data mining facility. It is exactly this feature that allows users to employ native programs within Java agents if computational speed is of concern. For faster prototype development and proof of concept, for example, we implemented the ID3 and CART learning algorithms as full Java agents and imported and used the Bayes, Wpebls, Ripper and CN2 learning programs in their native (C++) form. For the latter cases, we developed program-specific Java wrappers that define the abstract methods of the parent classes and are responsible for invoking the executables of these algorithms. Furthermore, to support the transfer of native classifiers across multiple sites, we overrode the default *writeObject* and *readObject* methods to transport files instead of objects. Contrary to the Java classifiers that are represented as objects with the ability to execute, native classifiers are, in their majority, passive constructs. By storing these native classifiers into conventional files and by re-defining the *writeObject* and *readObject* methods to transport files we achieve transparency between Java and native programs.

3.7 Adaptivity

The design of JAM provides an alternative approach to a phenomenon known as *concept drift*. Most classification systems operate under the hypothesis of a static environment, or in other words, an environment where the data distribution is fixed. The assumption is that the test data and all unseen instances are drawn from the same probability distribution as the training set. As we have already discussed with many examples in Chapter 1, however, this is rarely the case. In reality, the patterns will shift and the environment will change as time passes. Hence, the problem is to design a classification system that can evolve in case a new database becomes available.

The classifiers deployed in the traditional classification systems are obtained by applying machine learning programs over historical databases. One way to address *concept drift* in such systems is to merge the old and new databases into a larger database and re-apply the machine learning programs to generate new classifiers. This, however, may not constitute a viable solution. First, learning programs do not scale very well with large databases and second, the main memory requirement by the majority of learning programs poses a physical limitation to the size of the training databases.

A second alternative would be to employ *incremental* machine learning programs, (e.g., ID5 [Utgoff, 1988; 1989; 1994], an incremental version of ID3) or nearest neighbor algorithms. Incremental machine learning programs are not constrained to retain all training examples in main memory; instead they examine one instance at a time and tune the model accordingly. Hence, the classifiers initially trained over an existing database can be updated later by resuming their training on the new database once it becomes available. On the other hand, these algorithms do not provide a means for removing irrelevant knowledge gathered in the past. Furthermore, updating a model on every instance may not be accurate in a noisy domain. This shortcoming can be avoided by employing *incremental* batch learning methods [Clearwater *et al.*, 1989; Domingos, 1996; Wu & Lo, 1998], i.e., methods that update models using subsets of data. The problem with these approaches is that they are not general enough; instead they rely on specific algorithms, model representation and implementations. To our knowledge there are not many incremental machine learning algorithms and incremental batch learning methods implemented to date.

The architecture of JAM system introduces a third alternative, which is orthogonal to batch learning. New information is treated in a fashion similar to the information imported from remote data sites in JAM. Instead of combining classifiers from remote data sites (integration over space), JAM provides the means to combine classifiers acquired over different time periods (integration over time). The meta-learning component of JAM enables the incorporation of the new classifiers that capture emerging patterns learned over new data sources into its accumulated information (existing classifiers).

Let C_{new} be the set of classifiers generated from the latest batch of data and $C_{current}$ be the set of classifiers currently in use. The union of the $C_{current}$ and the C_{new} classifiers constitutes the new set of classifiers. An apparent disadvantage of this approach is the accumulation of multiple classifiers. Incorporating continuously new classifiers in the meta-classifier hierarchy would eventually result in a large and inefficient hierarchical meta-learning structure. In the framework of JAM, we address this drawback as part of the *efficiency* desideratum discussed later in Chapters 5 and 6. Briefly, the two chapters define and investigate several *pruning* techniques that aim to reduce the complexity and size of the meta-classifiers by evaluating the base classifiers over a validation set and by discarding those that are deemed redundant or less "relevant" to a particular meta-classifier.

After the pruning process over the validation set, a new meta-classifier is computed via meta-learning. The classifiers from $C_{current}$ that survived the pruning stage represent the existing knowledge, while the remaining classifiers from C_{new} denote the newly acquired information. A key point in this process is the selection of the validation set that is used during the pruning and meta-learning stages. A simple and straight-forward approach is to include in the validation set both old and new data and in proportions that reflect the speed of pattern changes (which can be approximated by monitoring the performance of $C_{current}$ over time). A more sophisticated approach would also weigh the data according to recency — weights decay over time.

In addition to solving the problem of how to make a learning system evolve and adjust according to its changing environment, this meta-learning-based solution has other advantages that make it even more desirable:

1. It is simple. Different classifiers capture the characteristics and patterns that surfaced over different period of times and meta-learning combines them in a straight-forward manner.

2. It integrates uniformly with the existing approaches of combining classifiers and information acquired over remote sources.

3. It is easy to implement and test. In fact, all the necessary components for building classifiers and combining them with older classifiers are similar or identical to the components used in standard meta-learning and can be re-used without modification.

4. It is modular and efficient. The meta-learning based system need not repeat the entire training process in order to create models that integrate new information. Instead, it can build independent models that can plug-in to the meta-learning hierarchy. In other words, we only need to train base classifiers from the new data and employ meta-learning techniques to combine them with other existing classifiers. In this way, the overhead for incremental learning is limited to the meta-learning phase.

5. It can be used in conjunction with existing pruning techniques. The new classifiers are not different in nature from the "traditional" classifiers, hence, in JAM we can adopt the same pruning methods to analyze and compare the collective set of classifiers and keep only those that contribute to the overall accuracy and not over-burden the meta-classification process. For example, JAM can decide to collapse or

substitute a sub-tree of the meta-classifier hierarchy with newly obtained classifier(s) that capture the same or new patterns.

This strategy is compatible with the JAM system and at the same time is scalable and generic, meaning that it can deal with many large databases that become available over time, and can support different machine learning algorithms respectively. The strategy allows JAM to extend and incorporate new information without discarding or depreciating the knowledge it has accumulated over time from previous data mining.

3.8 Summary

In this chapter we described the architecture of the JAM system, a distributed, scalable, portable, extensible and adaptive agent-based system that supports the launching of learning and meta-learning agents to distributed database sites. JAM consists of a set of similar and collaborating JAM sites in a network configuration maintained by the JAM Configuration Manager.

JAM is scalable in that it is designed with asynchronous, distributed communication protocols that enable the participating database sites to operate independently and collaborate with other peer sites as necessary, thus eliminating centralized control and synchronization points. JAM is portable because it is built upon existing agent infrastructure available over the Internet using Java technology and algorithm-independent meta-learning techniques. Adaptivity is attained by extending the meta-learning methods to combine both existing and new classifiers while extensibility is ensured by decoupling JAM from the learning algorithms and by introducing modular plug-and-play capabilities though a well-developed object-oriented design.

The objective of JAM is to integrate distributed knowledge and boost overall predictive accuracy of a number of independently learned classifiers through meta-learning agents. The next chapter describes the application of JAM on a real, practical and important problem. In collaboration with the FSTC (Financial Services Technology Consortium) we have populated multiple database sites with records of credit card transactions, provided by different banks, in an attempt to detect and prevent fraud by combining learned patterns and behaviors from independent sources.

Chapter 4

Applying JAM in Fraud Detection

As discussed in the previous chapter, JAM is an agent-based distributed data mining system that provides the means of dispatching and executing learning agents at remote database sites, with each learning agent being a Java-encapsulated machine learning program. One of JAM's key features is meta-learning, a general technique that combines multiple classification models, each of which may have been computed over distributed sites. In this chapter we describe the application of JAM in fraud detection in network-based information systems by detailing a comprehensive set of experiments in the real-world application of credit card fraud. We begin with a brief overview of the fraud detection application to highlight the advantages of JAM in a distributed learning environment.

4.1 Fraud Detection

A secured and trusted inter-banking network for electronic commerce requires high speed verification and authentication mechanisms that allow legitimate users easy access to conduct their business, while thwarting fraudulent transaction attempts by others. Fraudulent electronic transactions are a significant problem, one that grows in importance as the number of access points increase and more services are provided.

The traditional way to defend financial information system has been to protect the routers and network infrastructure. Furthermore, to intercept intrusions and fraudulent transactions that inevitably leak through, financial institutions have developed custom fraud detection systems targeted to their own asset bases. Recently however, banks have come to realize that a unified, global approach that involves the periodic sharing of information regarding fraudulent practices is required. Here, we employ the JAM system as an alternative approach that supports the cooperation among different institutions and consists of *pattern-directed inference systems* that use models of anomalous or errant transaction behaviors to forewarn of fraudulent practices. This approach requires the analysis of large and inherently distributed databases of information about transaction behaviors to produce models of "probably fraudulent" transactions. An orthogonal approach to modeling transactions would be to model user behavior. An application of this method, but in cellular phone fraud detection has been examined in [Fawcett & Provost, 1997].

The key difficulties in our strategy are: financial companies do not share their data for a number of (competitive and legal) reasons; the databases that companies maintain on transaction behavior are huge and growing rapidly; real-time analysis is highly desirable to update models when new events are detected and easy distribution of models in a networked environment is essential for up-to-date detection.

To address these difficulties and thereby protect against electronic fraud our approach has two key component technologies, both provided by JAM: *local fraud detection agents* that learn how to detect fraud within a single information system, and an integrated *meta-learning mechanism* that combines the collective knowledge acquired by the individual local agents. The fraud detection agents consist of *classification models* computed by machine learning programs at one or more sites, while meta-learning provides the means to combine a number of separately learned classifiers. Thus, meta-learning allows financial institutions to share their models of fraudulent transactions without disclosing their proprietary data. This way their competitive and legal restrictions can be met, but they can still share information. Furthermore, by supporting the training of classifiers over distributed databases, JAM can substantially reduce the total learning time (parallel learning of classifiers over (smaller) subsets of data). The final meta-classifiers (the combined ensemble of fraud detectors) can be used as sentries forewarning of possible fraud by inspecting and classifying each incoming transaction.

To validate the applicability of this approach in the security of financial information systems we experimented with two data sets of credit card transactions supplied by two different financial institutions. The task was to compute classification models that accurately discern fraudulent credit card transactions. In this chapter we apply several machine learning agents on different subsets of data from both banks to establish the potential of inductive learning methods in fraud detection. Then, we evaluate the utility of meta-learning by combining the fraud detectors of each bank. In later chapters we describe additional experiments aiming at computing superior meta-classifiers and we detail the exchange of classifiers between the two banks as a means to assess the validity and merit of this approach. By way of summary, we find that JAM, as a pattern-directed inference system coupled with meta-learning methods, constitutes a protective shield against fraud with the potential to exceed the performance of existing fraud detection techniques.

4.2 Experimental Setting

Before we discuss the various experiments and results of the empirical evaluation of JAM on the credit card fraud detection domain, we detail the learning algorithms and tasks.

4.2.1 Learning Algorithms

Five inductive learning algorithms are used in our experiments, Bayes, C4.5, ID3, CART and Ripper. Bayes implements a naive Bayesian learning algorithm described in [Minksy & Papert, 1969], CART [Breiman *et al.*, 1984], ID3 [Quinlan, 1986] and its successor C4.5 [Quinlan, 1993] are decision tree based algorithms, and Ripper [Cohen, 1995] is a rule induction algorithm.

4.2.2 Meta-Learning Algorithms

We employed eight different meta-learning techniques, based on the Voting, Stacking and SCANN methods described in Chapter 2. Specifically, we applied the two variations of voting, majority and weighted, the five learning algorithms (Bayes, C4.5, ID3, CART, Ripper) as meta-learning algorithms for stacking and the SCANN meta-learning method.

4.2.3 Data Sets

We obtained two large databases (70MB approximately) from Chase and First Union banks, both members of FSTC (Financial Services Technology Consortium), each with 500,000 records of credit card transaction data spanning one year (form October 1995 to September 1996). Chase bank data consisted, on average, of 42,000 sampled credit card transactions records per month with a 20% fraud and 80% legitimate distribution, whereas First Union data were sampled in a non-uniform (many records from some months, very few from others, very skewed fraud distributions for some months) manner with a total of 15% fraud versus 85% legitimate distribution. The database schemata were developed over years of experience and continuous analysis by bank personnel to capture important information for fraud detection. The records have a fixed length of 137 bytes each and about 30 numeric and categorical attributes including the binary class label (fraud/legitimate transaction). Among other data, each transaction included:

- A (jumbled) account number (no real identifiers)
- Scores produced by a COTS authorization/detection system
- Date/Time of transaction
- Past payment information of the transactor
- Amount of transaction

- Geographic information: where the transaction was initiated, the location of the merchant and transactor
- Codes for validity and manner of entry of the transaction
- An industry standard code for the type of merchant
- A code for other recent "non-monetary" transaction types by the transactor
- The age of the account and the card
- Other card/account information
- Confidential/Proprietary Fields (other potential indicators)
- Fraud Label (0/1)

The first step in this data mining process involves the arduous process of cleaning and preprocessing the given data sets. In this case, dealing with real-world data entailed missing data fields (records with fewer attributes), invalid entries (e.g., real values out of bounds), legacy systems remains (e.g., in some cases, letters were used instead of signed numbers for compactness), undefined classes for certain categorical attributes, conflicting semantics (e.g., in some cases for the same attribute, a zero denoted both a missing value, and the value 0), etc. Furthermore, we simplified the learning task by removing insignificant data (e.g., the last four digits of the nine digit zip codes), by discretizing some real values (e.g., the time a transaction took place) and by transforming attributes to more informative representations (e.g., we replaced the date of the last payment with the number of days passed till the day of the transaction).

Although preprocessing is an early task in the data mining process, we had to backtrack (sometimes even after learning and meta-learning) and repeat it several times until we settle on the final format for each data set.

4.2.4 Learning Tasks

Our task was to compute effective classifiers that correctly discern fraudulent from legitimate transactions. Contrary to most studies on comparing different classification models and systems, however, effectiveness does not mean overall accuracy (or minimal error rate). Other measures of interest include the True Positive (TP) and False Positive (FP) rates for (binary) classification problems, the ROC analysis (Receiver Operating Characteristics — used in signal theory for depicting and comparing hits versus false alarms) and problem specific cost models. A detailed study against the use of accuracy estimation for comparing induction algorithms can be found in [Provost, Fawcett, & Kohavi, 1998]. In the credit card fraud domain, overall predictive accuracy is inappropriate as the single measure of predictive performance. If 1% of the transactions are fraudulent, then a model that always predicts "legitimate" will be 99% accurate. Hence, TP rate is more important. Of the 1% fraudulent transactions, we wish to compute models that predict 100% of these, while producing no false alarms (i.e., predict no legitimate transactions to be fraudulent). Hence, maximizing the TP-FP spread may be the right measure of a successful model. Yet, one may find a model with TP rate of 90%, i.e., it correctly predicts 90% of the fraudulent transactions, but here it may correctly predict the lowest cost transactions, being entirely wrong about the top 10% most expensive frauds. Therefore, a cost model criteria may be the best judge of success, i.e., a classifier whose TP rate is 10% may be the best cost performer.

To evaluate and compare our fraud predictors, we adopted three metrics: the overall accuracy, the TP-FP spread[1] and a cost model fit to the credit card fraud detection problem. Overall accuracy expresses the ability of a classifier to provide correct predictions, TP-FP denotes the ability of a classifier to catch fraudulent transactions while minimizing false alarms, and finally, the cost model captures the performance of a classifier with respect to the goal of the target application (stop loss due to fraud).

Credit card companies have a fixed overhead that serves as a threshold value for challenging the legitimacy of a credit card transaction. If the transaction amount *transamt*, is below this threshold, they choose to authorize the transaction automatically. Each transaction predicted as fraudulent requires an "overhead" referral fee for authorization personnel to decide the final disposition. This "overhead" cost is typically a "fixed fee" that we call Y. Therefore, even if we could accurately predict and identify all fraudulent transactions, those whose *transamt* is less than Y would produce $\$(Y - transamt)$ in losses anyway. To calculate the savings each fraud detector contributes due to stopping fraudulent transactions, we use the following cost model for each transaction:

- If prediction is "legitimate" or ($transamt \leq Y$), authorize the transaction ($savings = 0$);
- Otherwise investigate the transaction:
  - If transaction is "fraudulent", $savings = transamt - Y$;
  - otherwise $savings = -Y$;

With this cost model per transaction, we seek to generate classification models C_i that maximize the savings $S(C_i, Y)$:

$$S(C_j, Y) = \sum_{i=1}^{n} [TF(C_j, \mathbf{x_i}) \cdot (transamt(\mathbf{x_i}) - Y) - FA(C_j, \mathbf{x_i}) \cdot Y] \cdot I(\mathbf{x_i}, Y)$$

where $TF(C_j, \mathbf{x_i})$ (TF: True Fraud) returns one when classifier C_j classifies correctly a fraudulent transaction $\mathbf{x_i}$ and $FA(C_j, \mathbf{x_i})$ (FA: False Alarm) returns one when classifier

[1]In comparing the classifiers, one can replace the TP-FP spread, which defines a certain family of curves in the ROC plot, with a different metric or even with a complete analysis [Provost & Fawcett, 1997; 1998] in the ROC space.

C_j misclassifies a legitimate transaction $\mathbf{x_i}$. $I(\mathbf{x_i}, Y)$ inspects the transaction amount *transamt* of transaction x_i, and returns one if it greater than the overhead \$Y and zero otherwise, while n denotes that number of examples in the data set used in the evaluation.

4.3 Data Partitioning

The size of the training set constitutes a significant factor in the overall performance of the trained classifier. In general, the quality of the computed model improves as the size of the training set increases. At the same time, however, the size of the training set is limited by main memory constraints and the time complexity of the learning algorithm.

Partitioning the data in small subsets may have a negative impact on the performance of the final classification model. An empirical study by Chan and Stolfo [Chan & Stolfo, 1995] that compares the robustness of meta-learning and voting as the number of data partitions increases while the size of the separate training sets decreases, shows training set size to be a significant factor in the overall predictive performance of the combining method. That study cautions that data partitioning as a means to improve scalability can have a negative impact on the overall accuracy, especially if the sizes of the training sets are too small. Here, we re-examine the effects of data partitioning on the predictive performance of learning, granted that an abundance of data is available. Given a large data set, the objective is to empirically determine a "good" scheme for partitioning the selected data set into subsets that are neither too large for the available system resources nor too small to yield inferior classification models.

To determine a good data partitioning scheme for the credit card data sets we applied the five learning algorithms over training sets of varying size. Specifically, we constructed training sets of varying size, starting from 100 examples to 200,000 examples for both credit card data sets. The accuracy (top row), the TP-FP spread (middle row) and the savings (bottom row) results reported in Figure 4.1 are the average results of each learning algorithm after a 10-fold cross-validation experiment. The left graphs correspond to classifiers trained over the Chase data set and the right graphs correspond to classifiers trained over the First Union data set. Training sets larger than 250,000 examples were too large to fit in the main memory of a 300 MHz PC with 128MB capacity running Solaris 2.6 for some of the learning programs. Moreover, the training of base classifiers from 200,000 examples required several CPU hours anyway, which prevented us from experimenting with larger sets.

According to these figures, different learning algorithms perform best for different evaluation metrics and for different training set sizes. Some notable examples include the ID3 algorithm that computes good decision trees over the First Union data and bad decision trees over the Chase data, and Naive Bayes that generates classifiers that are very effective according to the TP-FP spread and the cost model over the Chase data, but do not perform as well with respect to overall accuracy. The Ripper algorithm ranks

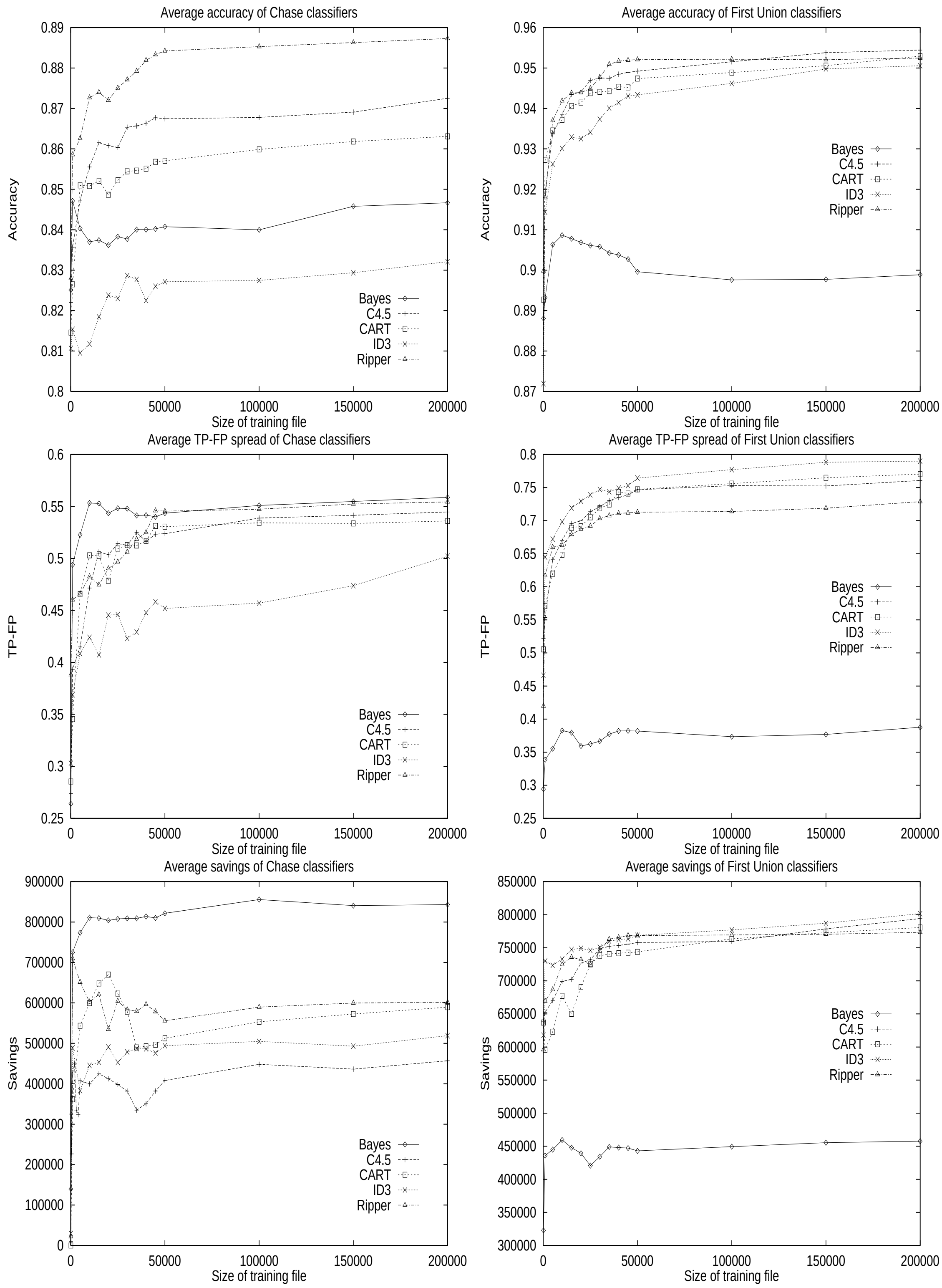

Figure 4.1: Accuracy (top), TP-FP spread (middle) and total savings (bottom) of the Chase (left) and First Union (right) classifiers as a function of the size of the training set.

among the best performers.

The graphs show that larger training sets result in better classification models, thus verifying Catlett's results [Catlett, 1991; 1992] pertaining to the negative impact of sampling on the accuracy of learning algorithms. On the other hand, they also show that performance curves converge, thus indicating reduced benefits as more data is used for learning. Increasing the amount of training data beyond a certain point may not necessarily provide performance improvements that are significant enough to justify the use of additional system resources to learn over larger data sets. In a related study, Oates and Jensen [T. Oates, 1998] find that increasing the amount of training data to build classification models often results in a linear increase in the size of the model with no significant improvement in their accuracy.

The empirical evidence suggests that there is a tradeoff between the efficient training and the effective training of classifiers. Recall that the objective was to compute base classifiers with good performance in a "reasonable" amount of time. Based on these curves for this data and learning task, the turning point lies between 40,000 and 50,000 examples (i.e., above this point, predictive performance improves very slowly). We decided to partition the credit card data in 12 sets of 42,000 records each, a scheme that (roughly) corresponds to partitioning transaction data by month.

4.4 Computing Base Classifiers

To generate our classification models we distributed each data set across six different data sites (each site storing two months of data) and we applied the five learning algorithms on each month of data, therefore creating 60 classifiers (10 classifiers per data site). This "month-dependent" data partitioning scheme was used only on the Chase bank data set. The very skewed nature of the First Union data forced us to equi-partition the entire data set randomly into 12 subsets and assign two subsets in each data site. Next, we had each data site import all "remote" classifier agents (50 in total) to test them against its "local" data. In essence, each classifier agent was evaluated on five different (and unseen) subsets.

Figure 4.2 presents the averaged results for the Chase (left plots) and First Union (right plots) credit card data. The top row shows the accuracy, the middle row depicts the TP-FP spread and bottom row displays the savings of each fraud detector. The x-axes for Chase plots represent the "months of data" used to train the classifier agents, starting in October 1995 and ending in September 1996 (the one year span is repeated for each of the five learning algorithms). For First Union, the x-axes correspond to the different subsets of data used to train the classifier agents (each tick represents one out of 12 subsets; the 12 subsets repeat five times, once for each of the five learning algorithms). Each vertical bar represents a specific classifier agent, the result of the application of a learning algorithm on a data subset. The first set of 12 bars denotes Bayesian classifiers

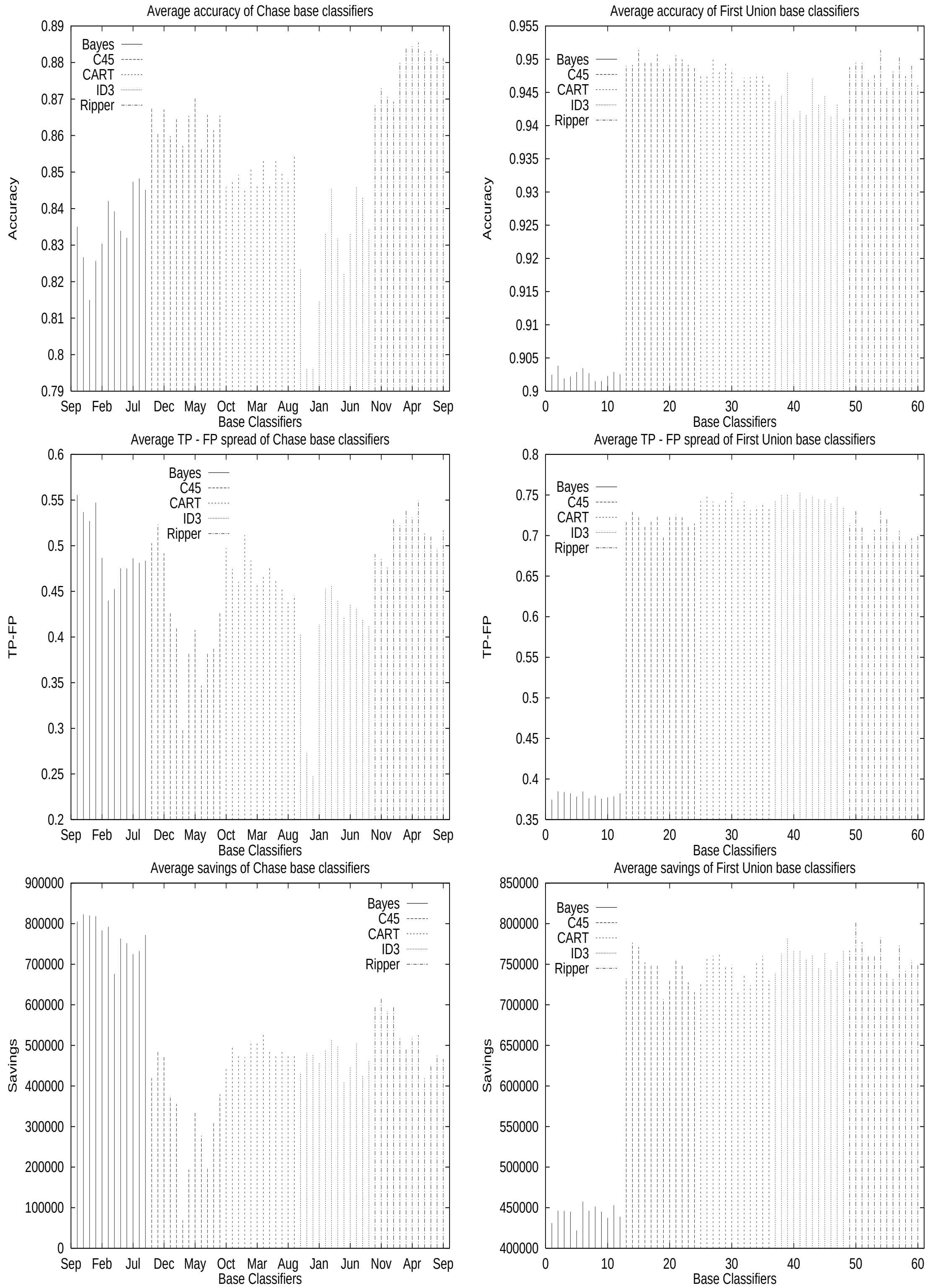

Figure 4.2: Accuracy (top), TP-FP spread (middle) and savings (bottom) of Chase (left) and First Union (right) classifiers on Chase and First Union credit card data respectively.

trained over the 12 different data subsets, the second set of 12 bars denotes the C4.5 classifiers etc. In other words, every 13^{th} bar corresponds to a classifier trained over the same data subset by a different learning agent. For example, the first bar of the left plot of the top row of Figure 4.2 represents the accuracy (83.5%) of the Bayesian classifier agent that was trained on the October 1995 data, the second bar represents the accuracy (82.7%) of the Bayesian classifier that was trained on the November 1995 data and the 13^{th} bar of the same plot (the first of the second group of the 12 adjacent bars) represents the accuracy 86.8% of the C4.5 classifier that was trained on the October 1995.

The maximum achievable savings for the perfect classifier, with respect to our cost model, is $1,470K for the Chase and $1,085K for the First Union data sets. According to the figures, some learning algorithms are more suitable for one problem for one evaluation metric (e.g., naive Bayes on Chase data is more effective in savings) than for another metric (e.g., accuracy of naive Bayes classifier on Chase data) or another problem (e.g., naive Bayes on First Union data), even though the two sets are very similar in nature. The figures also reveal that the classifier agents computed over the Chase data exhibit a larger variance in their performance, compared to those computed over the First Union data.

4.4.1 Discussion

The diversity among the classifiers is attributed, first, to the use of disparate learning algorithms (each with different search space, evaluation criteria, model representation and bias), and second, on the degree the training sets are different. Although the first factor is the same for both Chase and First Union datasets, we postulate that this is not the case with the second. Recall, that the First Union classifiers are trained over subsets of data of equal size and class distribution while the Chase classifiers are trained on subsets of data divided according to the date of the credit card transaction. The latter led to variations in the size of the training sets and the class distributions, thus explaining the increased variance within the set of Chase classifiers, even when comparing classifiers computed by the same learning algorithm. Conversely, the First Union classifiers, with the exception of the significantly inferior Bayesian classifiers, lack this diversity especially when comparing classifiers computed by the same learning algorithm.

To substantiate this conjecture,[2] we plotted Figure 4.3 as a means to visualize the diversity among the base classifiers. Each cell in the plot displays the diversity of a pair of base classifiers, with bright cells denoting a high degree of similarity and dark cells high degree of diversity. The bottom right half is allocated to Chase base classifiers while the top left half represents the First Union base classifiers. Each base classifier is identical to itself as shown by the white diagonal cells. The Bayesian First Union base classifiers,

[2]After all, similarity in performance does not necessarily imply high correlation between classifiers, e.g., it is possible for two classifiers to have different predictive behavior and exhibit similar accuracy.

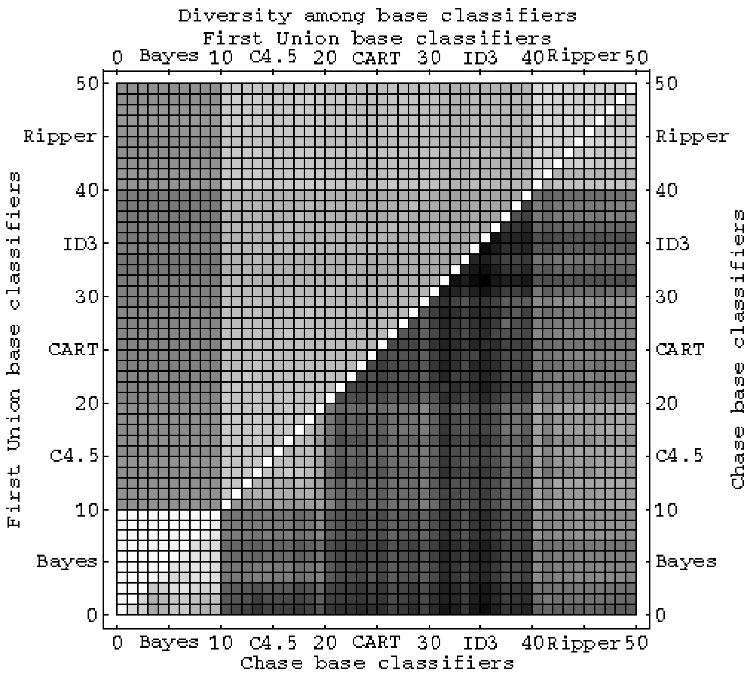

Figure 4.3: Base classifier diversity plot between all pairs of Chase base classifiers (bottom right half) and all pairs of First Union base classifiers (top left half).

for example, are very similar to each other (light color cells at the bottom left corner of the plot and above the diagonal) and fairly different from the rest (10 dark columns at the left side of the figure), an observation that confirms the results from the right side plots of Figure 4.2. The diversity plot clearly demonstrates that Chase base classifiers are more diverse than First Union base classifiers.

Overall, it appears that all learning algorithms performed better on the First Union data set than on the Chase data set. On the other hand, note that there are fewer fraudulent transactions in the First Union data and this causes a higher baseline accuracy. In all cases, classifiers are successful in detecting fraudulent transactions. Moreover, by combining these separately learned classifiers, it is possible to generate meta-classifiers (higher level classification models) with improved fraud detection capabilities. Next, we detail the experiment and present the results from combining the classifier agents under the three different meta-learning strategies, voting, stacking and SCANN.

Table 4.1: Performance of the meta-classifiers.

	Chase			First Union		
Algorithm	Accuracy	TP-FP	Savings	Accuracy	TP-FP	Savings
Majority	89.58%	0.556	$ 596K	96.16%	0.753	$ 798K
Weighted	89.59%	0.560	$ 664K	96.19%	0.737	$ 823K
Bayes	88.65%	**0.621**	**$ 818K**	96.21%	**0.831**	**$935K**
C4.5	89.30%	0.566	$ 588K	96.25%	0.791	$ 878K
CART	88.67%	0.552	$ 594K	96.24%	0.798	$ 871K
ID3	87.19%	0.532	$ 561K	95.72%	0.790	$ 858K
Ripper	89.66%	0.585	$ 640K	**96.53%**	0.817	$ 899K
SCANN	**89.74%**	0.574	$ 632K	96.44%	0.774	$ 855K

4.5 Combining Base Classifiers

Although the sites are populated with 60 classification models, in our experiments we combine only the 50 "imported" base classifiers. The 10 "local" base classifiers are left out to ensure that no classifiers predict on their own training data. (Meta-learning and testing is performed over the local data — the first subset is used as the validation set and the second subset as the test set). We applied eight different meta-learning methods at each site: the two voting methods (majority and weighted), the five stacking methods corresponding to the five learning algorithms each used as a meta-learner, and the SCANN method. Since all sites meta-learn the base classifiers independently, the setting of this experiment corresponds to a six-fold cross validation with each fold executed in parallel.

The performance results of these meta-classifiers averaged over the six sites are reported in Table 4.1. As with the base classifiers, none of the meta-learning strategies outperform the rest in all cases. It is possible, however, to identify SCANN and Ripper as the most accurate meta-classifiers and Bayes as the best performer according to the TP-FP spread and the cost model. (The best result in every category is depicted in bold.) On the other hand, the advantage of combining classifiers is evident. For all the evaluation metrics we were able to compute ensemble classification models that are capable of identifying a substantial portion of the fraudulent transactions and exhibit superior performance to that of the base level models.

4.5.1 Existing Fraud Detection Techniques

To compare with existing fraud detection techniques, we measured the performance of Chase's own COTS[3] authorization/detection system. (First Union's COTS authorization/detection system was not made available to us.) Such systems are trained to inspect and evaluate incoming transactions and produce scores in the [0-1000] range. Bank personnel determine a threshold value based on the target evaluation metric, and all trans-

[3]Commercial off the shelf fraud detection system.

actions scored above this threshold are considered suspicious. In Figure 4.4, we plot the accuracy, the TP-FP spread and savings of the COTS system as a function of its output. Notice, that the same results holds for Chase's system as well: there is no single best threshold; depending on the evaluation metric targeted different thresholds are better — 700 for accuracy (85.7%), 100 for the TP-FP spread (0.523) and 250 for savings ($682K). In contrast, the best results obtained by our own Chase meta-classifiers attained 89.75% in accuracy, 0.621 in the TP-FP spread, and $818K in savings.

4.5.2 Discussion

An unexpected outcome of the meta-learning experiments is the inferior performance of the weighted voting strategy with respect to the TP-FP spread and the cost model. Stacking and SCANN are at a disadvantage since their combining method is ill-defined: training classifiers to distinguish fraudulent transactions using standard learning methods is not a direct approach to maximizing savings (or the TP-FP spread). Standard learning algorithms are unaware of the adopted cost model and the actual value (in dollars) of the fraud/legitimate label; instead they are designed to reduce misclassification error. Hence, the most accurate classifiers are not necessarily the most cost effective. This can be demonstrated from the left plots of Figure 4.2. Although the Bayesian base classifiers are less accurate than the Ripper and C4.5 base classifiers, they are by far the best under the cost model. Similarly, the meta-classifiers in stacking and SCANN are trained to maximize the overall accuracy not by examining the savings in dollars but by relying on the predictions of the base-classifiers. In fact, the left plot of the bottom row of Figure 4.2 reveals that with only a few exceptions, Chase base classifiers are inclined towards catching "cheap" fraudulent transactions and for this they exhibit low savings scores. Naturally, the meta-classifiers are trained to trust the wrong base-classifiers for the wrong reasons, i.e., they trust the base classifiers that are most accurate instead of the classifiers that accrue highest savings. The same, but to a lesser degree, holds for the TP-FP spread.

Although weighted voting combines classifiers that too are unaware of the cost model, its meta-learning stage is independent of learning algorithms and hence it is not ill-defined. Instead, it assigns a degree of influence to each base classifier that is proportional to its performance (accuracy, TP-FP spread, savings). Hence, the collective prediction is generated by trusting the best classifiers as determined by the chosen evaluation metric over the validation set.

One way to deal with such ill-defined problems is to use cost-sensitive algorithms, i.e., algorithms that employ cost models to guide the learning strategy [Turney, 1995]. On the other hand, this approach has the disadvantage of requiring significant change to generic algorithms. An alternative, but (probably) less effective technique is to alter the class distribution in the training set [Breiman *et al.*, 1984; Chan & Stolfo, 1998] or to

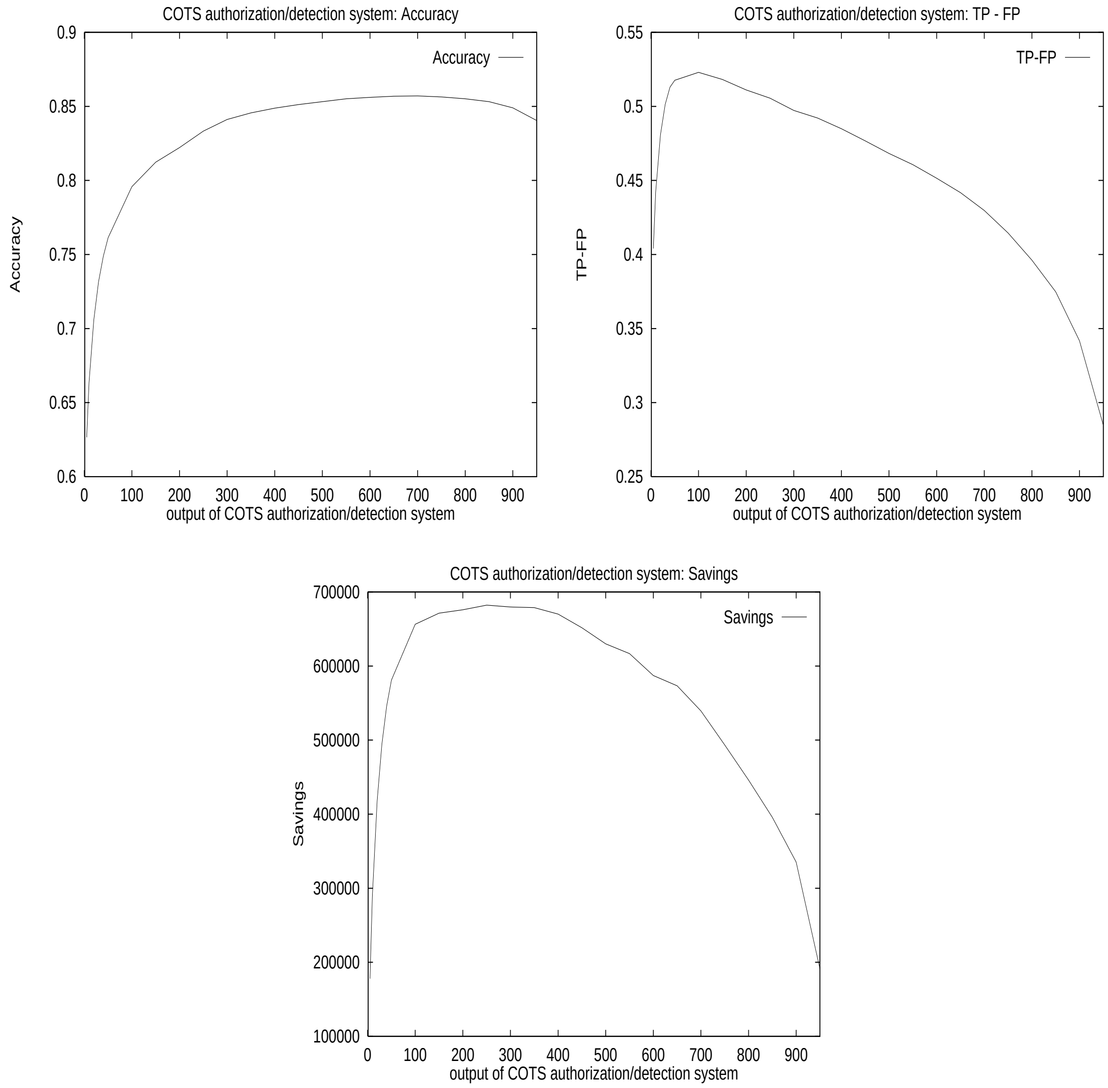

Figure 4.4: Performance of existing authorization/detection system on Chase's data.

tune the learning problem according to the adopted cost model. In the credit card fraud domain, for example, we can transform the binary classification problem into a multi-class problem by multiplexing the binary class with the continuous *tranamt* attribute (properly quantized into several "bins"). The classifiers derived from the modified problem would perhaps fit better to the specification of the cost model and ultimately achieve better results. A third option, complementary to the other two, is to have the meta-classifier pruned [Prodromidis & Stolfo, 1998c], i.e., discard the base classifiers that do not exhibit the desired property.

To improve the performance of our meta-classifiers, we followed the latter approach (see Chapter 5), even though it addresses the cost-model problem at a late stage, after base classifiers are generated. On the other hand it has the advantage of fitting better to the requirements of this problem since it treats classifiers as black boxes (financial institutions import pre-computed classification models), and also reduces the size of the meta-classifier, thus allowing for faster predictions and better use of system resources.

4.6 Summary

This chapter described a general method to protect financial information systems against fraudulent practices. The main advantages of this approach are: its flexibility to allow financial institutions to share their models of fraudulent transactions without disclosing their proprietary data and its ability to scale as the number and size of databases grow.

We applied this approach using the JAM system on actual credit card transaction data sets provided by two separate financial institutions. The experiments presented in this chapter involve the training of multiple fraud detectors (classifier agents) and meta-detectors within each data set. Through an extensive empirical evaluation we showed that, for the given data sets, meta-detectors exhibit far superior fraud detection capabilities comparing to single model approaches and traditional authorization/detection systems. Additional experiments in a later chapter (Chapter 7) will describe the meta-learning of meta-detectors across the two data sets.

We believe meta-learning systems deployed as intelligent agents will be an important contributing technology to deploy intrusion detection facilities in global-scale, integrated information systems.

Chapter 5

A-priori Pruning of Meta-Classifiers

Scalability and efficiency account for two necessary properties of distributed data mining systems. As discussed in Chapter 1 combining scalability and efficiency without sacrificing accuracy performance is one of the most intricate problems in the design of a distributed data mining system. The goal is dual:

1. to acquire and combine information from multiple databases in a timely manner and

2. to generate efficient and effective meta-classifiers.

In this thesis we addressed the first goal at the *system architecture level* by developing the JAM system. JAM is a distributed data mining system that supports meta-learning (Chapter 2) as a means to combine or integrate an ensemble of models computed in parallel by the same or different learning algorithms over multiple distributed data subsets. The system is designed with distributed protocols that support the dynamic reconfiguration of the system architecture (in case more data sites become available), with scalable protocols to allow data sites to participate in large numbers and with asynchronous protocols to avoid the overheads of synchronization barriers (Chapter 3). Moreover, we showed that by employing meta-learning techniques, JAM can potentially generate "higher quality" final classification models, comparing to the traditional single learning algorithm approaches (Chapter 4).

These gains, however, come at the expense of an increased demand for run-time system resources. The final ensemble meta-classifier may consist of a large collection of base classifiers that require increased memory resources while also slowing down classification throughput. In this chapter we address the second goal (efficiency and effectiveness) by examining the JAM system at the *data site* or *meta-learning level*.

5.1 Run-time Efficiency and Effectiveness

Numerous studies and our prior work posit that meta-learning approaches provide the means to efficiently scale learning to large datasets, while also boosting the accuracy over individual classifiers. Meta-learning techniques provide the advantage of improving the *scalability* of **learning** by executing the machine learning processes in parallel and on (possible disjoint) subsets of the data (a data reduction technique). On the other hand, meta-learning may not necessarily produce *efficient* models for **run-time** classification.

The main challenge in data mining and machine learning is to deal with large problems in a "reasonable" amount of time and at an acceptable cost. Constructing ensembles of classifiers is not cheap and produces a final outcome that is expensive due to the increased complexity of the generated meta-classifier. As the number of data sites, the size of data subsets and the number of deployed learning algorithms increases, more base classifiers are made available to each site and meta-learning and meta-classifying are bound to strain system resources. Furthermore, the meta-classifier hierarchy can be rebuilt and grown in breadth and depth to adapt to changes of patterns as new information is collected and new classifiers are generated as discussed in Section 3.7.

In general, meta-classifiers combine all their constituent base classifiers. Hence, to classify unlabeled instances, predictions need to be generated from all base classifiers before the meta-classifier can produce its final classification. This results in significant decrease in classification throughput (the speed with which an unknown datum can be classified) and increased demand for system resources (including memory to store base classifiers).

From experiments conducted on a Personal Computer with a 200MHz Pentium processor running Solaris 2.5.1 where base- and meta-classifiers were trained to detect credit card fraud, we measured a decrease of 50%, 70% and 80% credit card transaction processing throughput for meta-classifiers composed of 13, 20 and 30 base-classifiers, respectively. Meta-classifier throughput is crucial in real-time systems, such as e-commerce or intrusion detection systems. Memory constraints are equally important. For the same problem, a single ID3 decision tree may require more than 850KBytes of main memory, while a C4.5 decision tree may need 100KBytes. Retaining a large number of base classifiers and meta-classifiers may not be practical nor feasible.

Next, we describe fast methods, called *pruning techniques*, that aim to reduce the complexity and cost of ensemble meta-classifiers by evaluating the base classifiers, by filtering out (pruning) redundant models and by deploying only the most essential classifiers. In other words, the objective of pruning is to build partially grown meta-classifiers (meta-classifiers with pruned subtrees) to achieve comparable or better performance (accuracy) results than fully grown meta-classifiers. In particular, this chapter examines the *a-priori* pruning of *pre-training* pruning methods that filter the classifiers before they are used in the training of a meta-classifier. Instead of combining classifiers in a brute force manner, with pre-training pruning we introduce a preliminary stage for analyzing the available classifiers and qualifying them for inclusion in a meta-classifier. Only those classifiers that appear (according to one or more pre-defined metrics) to be most "promising" participate in the final meta-classifier. Conversely, *a-posteriori* pruning or *post-training* pruning methods, examined in Chapter 6, denotes the evaluation and the pruning of the constituent base classifiers after the complete meta-classifier has been constructed.

Besides accelerating the *run-time* classification process, pruning can prove invaluable to computationally and memory expensive meta-learning algorithms such as SCANN (i.e., it facilitates the meta-learning process). To meta-learn 50 classifiers over a validation set of 42,000 instances, for example, SCANN exhausts the available resources of the 300MHz PC with 128MB main memory and 465MB swap space. By discarding classifiers prior to meta-learning, pruning reduces the size of the SCANN meta-learning matrix and naturally simplifies the problem.

Last but not least, pruning can be used as a means to improve the predictive performance of the final classification model (meta-classifier). As discussed in Chapter 4, the effectiveness of classification models may not be measured only by their accuracy or error rate, but by other measures as well, such as the True Positive (TP) and False Positive (FP) rates or by ROC analysis and problem specific cost models. In general, unless the learning algorithm's target function is aligned with the evaluation metric, the resulting base- and meta-classifiers will not be able to solve the classification problem as best as possible except perhaps by chance. In such cases, it is often preferable to discard from the ensemble the base classifiers that do not exhibit the desired property, and hence "kill two birds with one stone"; address efficiency and effectiveness at the same time.

5.2 Evaluation Metrics for Pruning

First, we need to define several "heuristic" measures that are fast to compute in the hope of choosing the "best" classifiers to combine. Furthermore, these measures need to select base classifiers in a manner that reflects the alternative criteria that evaluate the ensemble meta-classifiers (e.g., accuracy, TP, FP, ROC analysis, TP-FP, cost model).

One may posit the view that the combining technique can be an arbitrarily expensive off-line computation. In the context we consider here, classifiers can be computed at remote sites at arbitrary times (even in a continuous fashion) over very large data sets that change rapidly. We seek to compute a meta-classifier as fast as possible on commodity hardware. Techniques that are quadratic in the training set size or cubic in the number of models are prohibitively expensive.

We therefore approach the problem considering fast search techniques to compute a meta-classifier. A greedy search method is described whereby we iteratively apply measures to the set of classifiers and a "best" classifier is chosen for inclusion until a termination condition is met. The resulting set of chosen classifiers are then combined into a meta-classifier. To compare classification models, we (naturally) rely on the most direct metrics (e.g., the accuracy, the TP-FP spread and the cost-model savings). However, we also investigate the utility of the *diversity*, the *coverage* and the *class specialty* properties of a candidate set of classifiers as alternate metrics that capture additional information for the better analysis and understanding of the characteristics of that set of classifiers.

5.2.1 Diversity

Brodley [C.Brodley, 1993] defines diversity by measuring the *classification overlap* of a pair of classifiers, i.e., the percentage of the instances classified the same way by two classifiers while Chan [Chan, 1996] associates it with the entropy in the set of predictions of the base classifiers. (When the predictions of the classifiers are distributed evenly across the possible classes, the entropy is higher and the set of classifiers is more diverse.) Other metrics studying diversity include the *KL-divergence* and the *kappa-statistic*. For example, Cover and Thomas [Cover & Thomas, 1991] use the Kullback-Leibler (KL) divergence to measure the distance between the probability distributions of the training sets of two classifiers while Margineantu and Dietterich [Margineantu & Dietterich, 1997] employ the kappa-statistic [Cohen, 1960; Bishop, Fienberg, & Holland, 1975; Agresti, 1990] to measure the agreement (or disagreement) between classifiers.

On the same subject, Krogh and Vedelsby [Krogh & Vedelsby, 1995] measure the diversity, they call it *ambiguity*, of an ensemble of base classifiers, by calculating the mean square difference between the prediction made by the ensemble and the base classifiers. They proved that increasing ambiguity decreases overall error. A similar conclusion is reached by Kwok and Carter in [Kwok & Carter, 1990]. Their study shows that ensembles with decision trees that were more syntactically diverse achieved lower error rates than ensembles consisting of less diverse decision trees, while Ali and Pazzani [Ali & Pazzani, 1996] suggest that the larger the number of gain ties,[1] the greater the ensemble's syntactic diversity is, which may lead to less correlated errors among the classifiers and hence lower error rates. However, they also cautioned that syntactic diversity may not be enough and members of the ensemble should also be competent (accurate). In the same study, Ali and Pazzani defined as *correlation error* the fraction of instances for which a pair of base classifiers make the same incorrect prediction and showed that there is a substantial (linear) negative correlation between the amount of error reduction due to the use of multiple models and the degree to which the errors made by individual models are correlated.

Here, we measure the diversity $\mathcal{D}$ within a set of classifiers (not just within a pair

[1] The information gain of an attribute captures the "ability" of that attribute to classify an arbitrary instance. The information gain measure favors the attribute whose addition as the next split-node in a decision tree (or as the next clause to the body of a rule) would result in a tree (rule) that would separate as many examples as possible into the different classes.

of classifiers) $\mathcal{H}$ by using a more direct and general approach. We calculate diversity $\mathcal{D}$ to be the average diversity of all possible pairs of classifiers in that set $\mathcal{H}$:

$$\mathcal{D} = \frac{\sum_{i=1}^{|\mathcal{H}|-1} \sum_{j=i+1}^{|\mathcal{H}|} \sum_{k=1}^{n} Dif\ (C_i(y_k),\ C_j(y_k))}{\frac{(|\mathcal{H}|-1)\cdot|\mathcal{H}|}{2} \cdot n} \tag{5.1}$$

where $C_j(y_i)$ denotes the classification of the y_i instance by the C_j classifier and $Dif(a, b)$ returns zero if a and b are equal, and one if they are different. Intuitively, the more diverse the classifiers are, the more room a meta classifier will have to improve performance. Freund and Schapire's [Freund & Schapire, 1996] boosting algorithm, for example, is designed to benefit from diverse classifiers obtained from a single learning program. The algorithm generates and combines multiple models, by purposely resampling the initial data set to artificially generate diverse training subsets and by applying the same learning program on each of those training subsets (see Chapter 2).

5.2.2 Coverage

Brodley and Lane [Brodley & Lane, 1996] defined as *coverage* the fraction of instances for which at least one of the classifiers produces the correct prediction. Under this definition, an instance is not covered if and only if all classifiers generate an incorrect prediction for that instance. In other words, if a meta-learning method (e.g., voting) is designed to only choose among the predictions of all the constituent classifiers (i.e., it cannot produce a different classification), then its coverage also signifies an upper bound of the accuracy it can attain. As a result, high coverage for such meta-learning methods is particularly important. On the other hand, as Brodley and Lane illustrate in that study, increasing coverage through diversity is not enough to ensure increased prediction accuracy; they argued that if the integration method does not utilize the coverage, then no benefit arises from integrating multiple classifiers.

5.2.3 Class Specialty

The above metrics are designed to measure the degree the classifiers correlate with each other and to quantify their overall accuracy potential. But they fail to evaluate classifiers with respect to cost models or take into account that classes have varying significance and are associated with different costs. The *class specialty* metric is created to address this limitation.

The term *specialty* was first defined by Chan [Chan, 1996] to be equal to one minus the average normalized entropy over K classifiers:

$$specialty = 1 - \frac{1}{K} \sum_{j}^{K} \frac{1}{\log m} \sum_{k}^{m} -p_{ji} \log(p_{ji}) \tag{5.2}$$

where m represents the number of classes and p_{ji} denotes the normalized accuracy of the j^{th} base-classifier on the i^{th} class. In essence, the larger the value of specialty the more specialized the base-classifiers are to certain classes. This metric, however, although informative, fails to distinguish among the classifiers and the particular classes they specialize.

Hence, in this thesis, we introduce the *class specialty* term to define a family of evaluation metrics that concentrate on the "bias" of a classifier towards certain classes. A classifier specializing in one class, should exhibit, for that class, both a high True Positive (TP) and a low False Positive (FP) rate. The TP rate is a measure of how often the classifier predicts the class correctly, while FP is a measure of how often the classifier predicts that class by mistake.

For concreteness, given a classifier C_j and a data set with n examples, we construct a two dimensional contingency table T^j where each cell T^j_{kl} contains the number of examples $\mathbf{x}$ for which the true label $L(\mathbf{x}) = k^{th}$ class and $C_j(\mathbf{x}) = l^{th}$ class. Thus, cell T^j_{kk} contains the number of examples classifier C_j classifies correctly as class k. If the classifier C_j is capable of 100% accuracy on the given data set, then all non-zero counts appear along the diagonal. The sum of all the cells in T^j is n. Then, the TP and FP rates are defined as:

$$TP(C_j, k) = \frac{T^j_{kk}}{\sum_{i=1}^{m} T^j_{ki}} \qquad FP(C_j, k) = \frac{\sum_{i \neq k} T^j_{ik}}{\sum_{i \neq k} \sum_{l=1}^{m} T^j_{il}} \tag{5.3}$$

In essence, $TP(C_j, k)$ measures the examples classified correctly to be in class-k versus the total number of examples of that class. Analogously, $FP(C_j, k)$ calculates the number of examples that classifier C_j misclassified as class k versus the total number of examples that belong in different classes (m is the number of classes). With the *class specialty* metric we attempt to quantify the bias of a classifier towards a certain class. In particular, a classifier C_j is highly biased/specialized for class k when its $TP(C_j, k)$ is high and its $FP(C_j, k)$ is low. With the class specialty defined, we can evaluate the candidate (base-) classifiers and combine those with the highest "bias" per class, or in other words, those with the most specialized and accurate view of each class, in the hope that a meta-classifier trained on these "expert" (base-) classifiers will be able to uncover and learn their bias and take advantage of their properties.

Naive Class Specialty Metric

The naive class specialty metric is defined by Equation 5.3 and constitutes the most straight forward metric of the class specialty family. It captures the TP and FP rates of a classifier for each class over the validation set and is used for the selection of the candidate base classifiers to be included in the final meta-classifier. For example, a simple pruning algorithm that is based on the naive class specialty metric that also takes into account the different costs associated with each class would be the one that chooses to integrate in the meta-classifier the base classifiers that exhibit high TP rates for each class with the classifiers that exhibit low FP rates for that class in a number proportional to the cost associated for each class.

Combined Class Specialty Metric

The problem with the naive class specialty metric is that it may qualify poor classifiers. Assume, for instance, the extreme case of a classifier that always predicts the class k. This classifier is highly biased and pruning algorithms such as the one outlined above will select it. So, we introduce two new metrics that are more balanced, the *positive combined specialty* $PCS(C_j, k)$ and the *negative combined specialty* $NCS(C_j, k)$ metrics, that take into account both the TP and FP rates of a classifier for a particular class. The former is biased towards TP rates, while the latter is biased towards FP rates:

$$PCS(C_j,k) = \frac{TP(C_j,k) - FP(C_j,k)}{1 - TP(C_j,k)} \qquad NCS(C_j,k) = \frac{TP(C_j,k) - FP(C_j,k)}{FP(C_j,k)} \tag{5.4}$$

According to the definition, the PCS metric ranks higher (with high values) classifiers with high TP rate while penalizing classifiers with high FP rates. In contrast, the NCS metric assigns higher values to classifiers with reduced FP rates and low or even negative scores for classifiers with high FP rates. The PCS and NCS rates are undefined for the extreme classifiers that always predict a single class. The two new metrics can be used to select classifiers in a manner similar to the naive class specialty metrics. In the context of the pruning algorithm examined earlier as an example, this would mean that meta-classifiers will be composed of classifiers that exhibit high PCS rates for each class with classifiers that exhibit high NCS rates for that class.

Balanced Class Specialty Metric

A third alternative is to define a metric that combines the TP and FP rates of a classifier for a particular class into a single formula. Such a metric has the advantage of distinguishing the single best classifier for each class with respect to some predefined criteria (e.g., misclassification costs). The *balanced class specialty* metric, or $BCS(C_j, k)$, is defined as:

$$BCS(C_j, k) = f_{TP}(k) \cdot TP(C_j, k) + f_{FP}(k) \cdot FP(C_j, k) \tag{5.5}$$

where $-1 \leq f_{TP}(k), f_{FP}(k) \leq 1, \forall k \in$ {class 1, class 2,..., class m}. Coefficient functions f_{TP} and f_{FP} are single variable functions quantifying the importance of each class according to the needs of the learning task and the distribution of each class in the entire data set. Note that the total *accuracy* of a classifier C_j is a special case of this metric; it can be computed by calculating the $\sum_{k=1}^{m} BCS(C_j, k)$, with each $f_{TP}(k)$ set to the distribution of the class k in the testing set and each $f_{FP}(k)$ set to zero.

In many real world problems, e.g., medical diagnosis, credit card fraud, etc., the plain $TP(C_j,k)$ and $FP(C_j,k)$ rates fail to capture the entire story. The distribution of the classes in the data set may not be uniform and maximizing the TP rate of one class may be more important than maximizing total accuracy. As discussed in Chapter 4, for instance, in the credit card fraud detection domain, catching expensive fraudulent transactions is more vital than eliminating false alarms. The *balanced class specialty metric* provides the means to associate a misclassification cost for each class to evaluate the classifiers from a more realistic perspective.

A more general and elaborate specialty metric can take into account the individual instances as well. For the balanced class specialty metric this is materialized by defining dynamic coefficient functions f_{TP} and f_{FP}, i.e., functions where individual instances are also a parameter.

$$\begin{aligned} BCS(C_j,k,(\mathbf{x_i},y_i)) &= f_{TP}(k,(\mathbf{x_i},y_i)) \cdot TP(C_j,k,(\mathbf{x_i},y_i)) + \\ & \quad f_{FP}(k,(\mathbf{x_i},y_i)) \cdot FP(C_j,k,(\mathbf{x_i},y_i)) \end{aligned} \tag{5.6}$$

It is this variation of the balanced class specialty metric that is used to introduce complex cost models in the evaluation and selection process of the candidate classifiers. The cost model of the credit card fraud detection problem that was introduced in Chapter 4, is as specific case of the new BCS metric. The $f_{TP}(fraud,(\mathbf{x_i},fraud))$ coefficient function of $BCS(C_j,fraud,(\mathbf{x_i},y_i))$ is defined to return zero if the $transamt$ of transaction $\mathbf{x_i}$ is less than the overhead Y and $transamt - Y$ if the $transamt$ is equal or greater than Y. Similarly, the $f_{FP}(legitimate,(\mathbf{x_i},fraud))$ coefficient function is defined to return zero if the $transamt$ of transaction $\mathbf{x_i}$ is less than the overhead Y and $-Y$ if the $transamt$ is equal to or greater than Y. The $f_{TP}(legitimate,(\mathbf{x_i},legitimate))$ and $f_{FP}(fraud,(\mathbf{x_i},legitimate))$ coefficients functions are set to always return zero to be consistent with the definition of the cost model that measures the savings incurred due to timely fraud detection.

Aggregate Specialty

For completeness, we describe another metric of the *class specialty* family that is called *aggregate specialty*. However, we will not consider it further since it suffers from the same disadvantage as the specialty metric defined by Chan [Chan, 1996] (see Equation 5.2), namely the inability to distinguish among the classes in which each classifier specializes. The *aggregate specialty* metric characterizes classifiers by measuring their "total" specialty, that is, their specialty when taking into account all classes together. Formally, the *Aggregate Specialty* $AS(C_j)$ metric of a classifier C_j is defined as:

$$AS(C_j) = \sqrt[m]{\prod_{i=1}^{m} TP(C_j,i)} \tag{5.7}$$

which is, basically, the geometric mean of the accuracies of a classifier measured on each class separately. The geometric mean of m quantities reaches high values only if all m values are high enough and in balance. In these cases, $AS(C_j)$ will have a high value when classifier C_j performs relatively well on all m classes. A highly specialized classifier, on the other hand, will exhibit a lower $AS(C_j)$ value. This metric can be very useful with skewed data sets [Kubat & Matwin, 1997] in which some classes appear much more frequently that others. In this cases, the aggregate specialty metric distinguishes the classifiers that can focus on the sparse examples.

5.2.4 Combining Metrics

Instead of relying just on one criterion to choose the "best" (base-) classifiers, the pruning algorithms can employ several metrics simultaneously. Different metrics capture different properties and qualify different classifiers as "best". By combining the various "best" classifiers into a meta-classifier we can presumably form meta-classifiers of higher accuracy and efficiency, without searching exhaustively the entire space of the possible meta-classifiers. For instance, one possible approach would be to combine the (base-) classifiers with high *coverage* and low *correlation error*. In another study [Stolfo *et al.*, 1997a] concerning the same credit card fraud detection problem, for example, we investigated evaluation formulas for selecting classifiers that linearly combine multiple characteristics such as *diversity*, *coverage* and *correlated error* or their combinations, e.g., the weighted combination of *True Positive rate* and *diversity*.

Next, we introduce three pre-training algorithms that are based on different evaluation metrics and search heuristics.

5.3 Pruning Algorithms

As we have already discussed, *a-priori* pruning or *pre-training* pruning refers to the evaluation and elimination of classifiers before they are used for the training of the meta-classifier. A pre-training algorithm is given a set of pre-computed classifiers $\mathcal{H}$ (obtained from one or more databases by one or more machine learning algorithms) and a validation set $\mathcal{V}$ (a separate subset of data, disjoint from the training and test sets). Its result is a set of classifiers $\mathcal{C} \subseteq \mathcal{H}$ to be combined in a meta-classifier. A pictorial description of this pruning process is shown in Figure 5.1. Determining the optimal meta-classifier, however, is a combinatorial problem. Wolpert, for example, considers forming effective combinations as a "black art" [Wolpert, 1992]. Here, we employ the accuracy, diversity, coverage and specialty metrics to guide the greedy search. More specifically, we implemented three instances of a metric-based pruning algorithm, a diversity-based pruning algorithm and three instances of a combined specialty/coverage-based pruning algorithm.

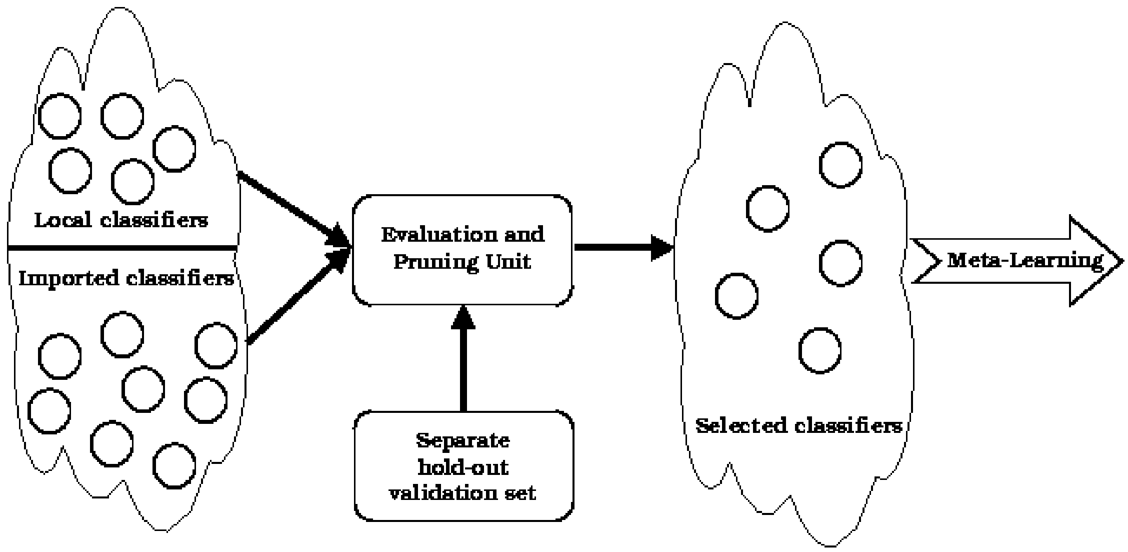

Figure 5.1: Pre-training pruning.

5.3.1 Metric-Based Pruning Algorithm

The most obvious approach is to combine the best classifiers according to their performance. In the credit card fraud detection problem, for example, the metric for evaluating the candidate classifiers can be one of the accuracy, the TP-FP spread or the savings measures. Thus, a particular instance of this pruning algorithm would rank the available classifiers independently over a separate validation set using one of these metrics and then select the best $|\mathcal{C}|$ models. To study the effectiveness of this approach on the credit card data sets, we implemented all three instances of the metric-based pruning algorithm, namely, the accuracy-based, the TP-FP spread-based and the savings-based pruning algorithms.

5.3.2 Diversity-Based Pruning Algorithm

The formal description of the algorithm is displayed in the left side of Figure 5.2. The diversity-based algorithm works iteratively selecting one classifier each time starting with the most accurate (base) classifier. Initially it computes the diversity matrix d where each cell d_{ij} contains the ratio of the instances of the validation set for which classifiers C_i and C_j give different predictions. In each round, the algorithm adds to the list of selected classifiers $\mathcal{C}$ the classifier C_k that is most diverse to the classifiers chosen so far, i.e., the C_k that maximizes $\mathcal{D}$ (defined in Equation 5.1) over $\mathcal{C} \cup C_k, \forall k \in \{1, 2, \ldots |\mathcal{H}|\}$. The selection process ends when the K most diverse classifiers are selected. K is a parameter that depends on factors such as minimum classification throughput, memory constraints

```
Let H := Initial set of classifiers
Let C := ∅, K := maximum number of classifiers
For i := 1, 2, ... , |H| - 1 do
    For j := i, i+1, ... , |H| do
        Let d_ij := the ratio of the instances the C_i
                    and C_j give different predictions
Let C' := the classifier with the highest accuracy
C := C ∪ {C'}, H := H − {C'}
For i := 1, 2, ... , K do
    For j := 1, 2, ... , |H| do
        Let D_j := 2 · (Σ_{k=1}^{|C|} d_jk)/|C| · (|C| + 1)
    Let C' := the classifier from H with the highest D_j
    C := C ∪ {C'}, H := H − {C'}
```

```
Let C := ∅
For all target classes k, k = 1, 2, ... , m, do
    Let E := V
    Let C_k := ∅, H_k := |H|
    Until no other examples in E can be covered
          or H_k = ∅
        Let C' := the classifier with the best class
                  specialty on target class k for E
        C_k := C_k ∪ {C'}, H_k := H_k − {C'}
        E := E - examples covered by C'
    C := C ∪ C_k
```

Figure 5.2: The Diversity-based (left) and Specialty/Coverage-based (right) pruning algorithms.

or diversity thresholds. (The diversity $\mathcal{D}$ of a set of classifiers $\mathcal{C}$ decreases monotonically as the size of the set increases. By introducing a threshold, one can avoid including redundant classifiers in the final outcome.)

The algorithm is independent of the number of attributes of the data set and its complexity is bounded by $O(n \cdot |\mathcal{H}|^2)$ (where n denotes the number of examples) due to the computation of the diversity matrix. For many practical problems, however, $|\mathcal{H}|$ is much smaller than n and the overheads are minor.

5.3.3 Specialty/Coverage-Based Pruning Algorithm

A formal description of the algorithm is displayed in the right side of Figure 5.2. The algorithm combines the *coverage* metric and one of the instances of the *class specialty metric*. Initially, the algorithm starts by choosing the (base-) classifier with the best performance with respect to the specialty metric for a particular target class on the validation set. Then, it iteratively selects classifiers based on their performance on the examples the previously chosen classifiers failed to cover. The cycle ends when there are no other examples to cover. The algorithm repeats the selection process for a different target class. The complexity of the algorithm is bounded by $O(n \cdot m \cdot |\mathcal{H}|^2)$. For each target class (m is the total number) it considers at most $|\mathcal{H}|$ classifiers. Each time it compares each classifier with all remaining classifiers (bounded by $|\mathcal{H}|$)) on all misclassified examples (bounded by n).

Even though the algorithm performs a greedy search, it combines classifiers that are diverse (they classify correctly different subsets of data), accurate (they exhibit the best performance on the data set used for evaluation with respect to the class specialty) and with high coverage. The three instances of this algorithm that we study here, combine coverage with $PCS(C_j, k)$, coverage with a specific $BCS(C_j, k)$ and coverage with the elaborate version of $BCS(C_j, k)$ that incorporates the specific cost model that is tailored to the credit card fraud detection problem.

5.3.4 Related Work

Provost and Fawcett [Provost & Fawcett, 1997; 1998] introduced the ROC convex hull method as a means to manage, analyze and compare classifiers. The ROC convex hull method is intuitive in that it provides clear visual comparisons and is flexible in the sense that it allows classifier evaluation under different metrics (e.g., accuracy, true positive/false negative rates, error cost, etc.). The method operates by mapping the candidate classifiers onto a *True Positive/False Positive* plane and by identifying the potentially optimal classifiers under specific conditions. An additional benefit of the ROC convex hull method is that it retains a small subset of the available models, and thus it reduces the amount of extra resources required to manage the sub-optimal classifiers and their performance data.

On the other hand, this method provides no information with respect to the interdependencies among the base classifiers when combined into higher level meta-classifiers. The intent of that work is to discover single classifiers that perform optimally under certain conditions and prune away the inferior models, while here we explore evaluation metrics that focus on the relations between classifiers and their potential when forming ensembles. In fact, as it will be shown, the performance of sub-optimal yet diverse models can substantially improve and even surpass that of the best single model when combined together.

In other related work, Margineantu and Dietterich [Margineantu & Dietterich, 1997] studied the problem of pruning the ensemble of classifiers obtained by the boosting algorithm AdaBoost [Freund & Schapire, 1996], described in Chapter 2. In that paper, the authors acknowledge the importance of reducing the number of classifiers and discuss five different selection methods, namely *early stopping, KL-divergence pruning, Kappa pruning, Kappa-Error Convex Hall Pruning* and *Reduce-Error Pruning with Backfitting.* According to their findings, it is possible for a subset of classifiers to retain a percentage of the performance gains achieved by the entire set.

Briefly, early stopping refers to the blind approach of keeping the first M classifiers obtained, whereas KL-divergence pruning attempts to detect the most diverse classifiers by examining the probability distributions of their respective training set. Both methods, however, fail to produce results of practical use. The Kappa-Error Convex Hull Pruning method, on the other hand, appears more promising, but it is also more restrictive, in that it can select only a fixed number of classifiers. It maps all classifiers in an accuracy-diversity plane and chooses those that form the accuracy-diversity convex hull of the available classifiers. Overall the best pruning method found in that study is Reduce-Error Pruning with Backfitting and then Kappa Pruning. The latter relies on the discovery of the most diverse classifiers by evaluating their predictions on the training set, while the former takes a more direct approach and selects the subset of the classifiers that gives the best voted performance on a separate pruning (validation) data set. The most apparent disadvantage of the best method (reduced error pruning with backfitting) is that it relies on a computationally expensive algorithm. It performs an extensive search for the subset of classifiers that gives the best voted performance on a separate pruning (validation) data set.

Although related to our work, Margineantu and Dietterich have restricted their research in the boosting algorithm that derives all classifiers by applying the same learning algorithm on many different subsets of the same training sets. In this thesis, we are considering the same problem but with several additional dimensions. We consider the general setting where ensembles of classifiers can be obtained by applying potentialy different learning algorithms over (potentially) distinct databases. Furthermore, instead of voting (AdaBoost) over the predictions of classifiers for the final classification, we adopt meta-learning as a more general approach to combine predictions of the individual classifiers. Margineantu and Dietterich's best algorithm (reduced error pruning with backfitting), for example, depends on the computation of a large number of intermediate meta-classifiers before it converges to its final set of classifiers, a possibly prohibitive factor when other meta-learning techniques are employed. By relying on the heuristic metrics described in this chapter and following greedy search methods, our pre-training pruning algorithms avoid computing such intermediate meta-classifiers.

5.4 Incorporating Pruning Algorithms in JAM

To integrate the various techniques within JAM and at the same time be consistent with the system's objectives, we followed an object-oriented design for pruning as well. As with the Learner and Classifier classes (Chapter 3), JAM provides the definition of the abstract parent Prune class and every pruning technique can be subsequently defined as a subclass of this parent class. To deploy one of the pruning methods discussed earlier (i.e., the metric-based, the diversity-based or the specialty/coverage-based) or any other new or tailored method, JAM simply needs to instantiate the implemented subclasses with the appropriate arguments (the vector of candidate base classifiers agents, the meta-learning agent, the stopping criteria, etc.) prior to meta-learning, and invoke its redefined *selectClassifiers* method. This method is responsible for evaluating the candidate classifiers and for returning the new vector of the selected classifiers; different implementations of this method, materialize different pruning algorithms. As long as a pruning object conforms to the interface defined by the abstract parent Prune class, it can be introduced and used immediately as part of the JAM system.

Table 5.1: Performance of the best pruned meta-classifiers.

	Chase						First Union					
Algorithm	Accur.	K	TP-FP	K	Savings	K	Accur.	K	TP-FP	K	Savings	K
Majority	89.60%	47	**0.577**	11	**$ 902K**	3	**96.55%**	15	**0.780**	15	$ **854K**	13
Weighted	89.60%	48	**0.577**	11	$ **905K**	4	**96.59%**	12	**0.789**	12	$ **862K**	10
Bayes	**89.33%**	16	**0.632**	32	$ **903K**	5	**96.57%**	13	**0.848**	12	$ 944K	12
C4.5	**89.58%**	14	0.572	27	$ **822K**	5	**96.51%**	12	0.799	25	$880K	42
CART	**89.49%**	9	**0.571**	18	$ **826K**	5	**96.48%**	12	0.801	29	$ 884K	37
ID3	**89.40%**	8	**0.568**	1	$ **811K**	1	**96.45%**	8	0.795	30	$ 872K	40
Ripper	89.70%	46	**0.595**	47	$ **858K**	3	96.59%	30	0.821	36	$ 902K	44
SCANN	89.76%	46	0.581	32	$ **880K**	5	96.45%	49	0.794	12	$ **900K**	12

5.5 Empirical Evaluation

To test the performance of the pruning algorithms we continued with the experiment described in Chapter 4. Recall that we used the JAM system to apply five learning agents over 12 disjoint subsets of credit card data distributed across six JAM sites to generate 60 base classifier agents. Then each JAM site imported the 50 other remote classifier agents and combined them using eight different meta-learning methods, effectively computing eight different meta-classifiers. These meta-classifiers (eight per site) were evaluated against separate unseen test data and their results were averaged.

Determining the optimal set of classifiers for meta-learning is a combinatorial problem. With 50 base classifiers per data site, there are 2^{50} combinations of base classifiers that can be selected. To search the space of the potentially most promising meta-classifiers we applied the pruning algorithms introduced earlier. Again, we had each site use half of its local data (first subset) to test, prune and meta-learn the base classifiers and the other half (second subset) to evaluate the overall performance of the pruned meta-classifier.

Combining eight meta-learning algorithms with multiple pruning algorithms while varying the pruning requirements (number of classifiers to be discarded) generates a large number of meta-classifiers of different sizes. Table 5.1 presents a summary result of the best averaged pruned meta-classifiers and their size (number of constituent base classifiers, denoted as K). The detailed results are presented in the following sections. In this Table, the meta-classifiers are evaluated according to their accuracy (denoted as "accur."), their TP-FP spread (denoted as "TP-FP") and their performance with respect to the cost model (denoted as "savings"). Entries in bold indicate a statistically significant performance improvement comparing to that of the unpruned meta-classifiers. To be more specific, the error rates of the two meta-classifiers are different (rejection of the null hypothesis) with 99% confidence according to the paired t test. (The improvement on the accuracy of the Ripper meta-classifier on the First Union data set is statistically significant with 95% confidence). As expected, however, there is no single best meta-classifier; depending on the evaluation criteria, different meta-classifiers of different sizes perform better. Overall, it is possible, to identify Ripper as the meta-learning algorithm that computes the most accurate meta-classifiers and Bayes as the best performer according to the TP-FP spread and the cost model.

5.5.1 Comparing the Pruning Algorithms

To better understand how the selection process affects the performance of the meta-learning algorithm, we present the results of the experiments involving the various pruning algorithms when applied to the best meta-learning algorithms on the Chase and First Union data. The detailed graphs are displayed in Figure 5.3. The top row shows the overall accuracies, the middle row depicts the TP-FP spreads and the bottom row plots the savings (in dollars). The left-side graphs correspond to the results from the Chase bank data while the right-side graphs represent the results from the First Union bank data. Each figure contrasts one *metric-based* pruning method, two *specialty/coverage-based*, one *diversity-based* method, and an additional classifier selection method denoted here as *arbitrary*. As the name indicates, *arbitrary* uses no particular strategy to evaluate base classifiers; instead it combines them in a "random" order, e.g., when they become available. The first specialty/coverage pruning algorithm that implements the PCS/coverage-based algorithm and the diversity-based and arbitrary pruning algorithms are included in all figures. The metric-based pruning algorithm, on the other hand, is different for each figure depending on the evaluation metric. Specifically, the overall accuracy graphs display the plot of the accuracy-based pruning algorithm, the TP-FP spread graphs display the plot of the TP-FP spread-based pruning algorithm and the cost model-based graphs display the plot of the cost-based pruning algorithm. A similar approach is followed for the second of the two specialty/coverage-based pruning algorithms; the instance of the specialty/coverage-based pruning algorithm that is examined in each figure depends on the particular evaluation metric of that figure. In total we implemented three instances of the BCS specialty metric, namely the accuracy, the TP-FP,[2] and the cost model, thus we compared the accuracy/coverage-based pruning algorithm, the TP-FP/coverage-based pruning algorithm and the cost model/coverage-based pruning algorithm.

The vertical lines in the figures denote the number of base classifiers integrated in the final meta-classifier as determined by the specialty/coverage algorithms. The final Chase meta-classifier for the TP-FP/coverage algorithm, for example, combines 33 base classifiers (denoted by the TP-FP vertical line), while the final First Union meta-classifier for the accuracy/coverage algorithm consists of 27 base classifiers (denoted by the Accuracy vertical line). In these graphs we have included the intermediate performance results (i.e., the accuracy, TP-FP rates and the savings of the partially built meta-classifiers) as

[2]Recall that the BCS/coverage pruning algorithm is defined to iterate through all k classes and select the classifiers that maximize the $BCS(k)$, for each k. The fraud detection problem, however, is a binary classification problem, hence the BCS/coverage algorithm is, initially, reduced to select the classifiers that maximize BCS (fraud), (i.e., f_{TP} (fraud) $\cdot$ TP(C_j,fraud) - f_{FP}(fraud) $\cdot$ FP(C_j, fraud)), and furthermore reduced to TP(C_j,fraud) - FP(C_j, fraud) or simply TP-FP to match the evaluation metric.

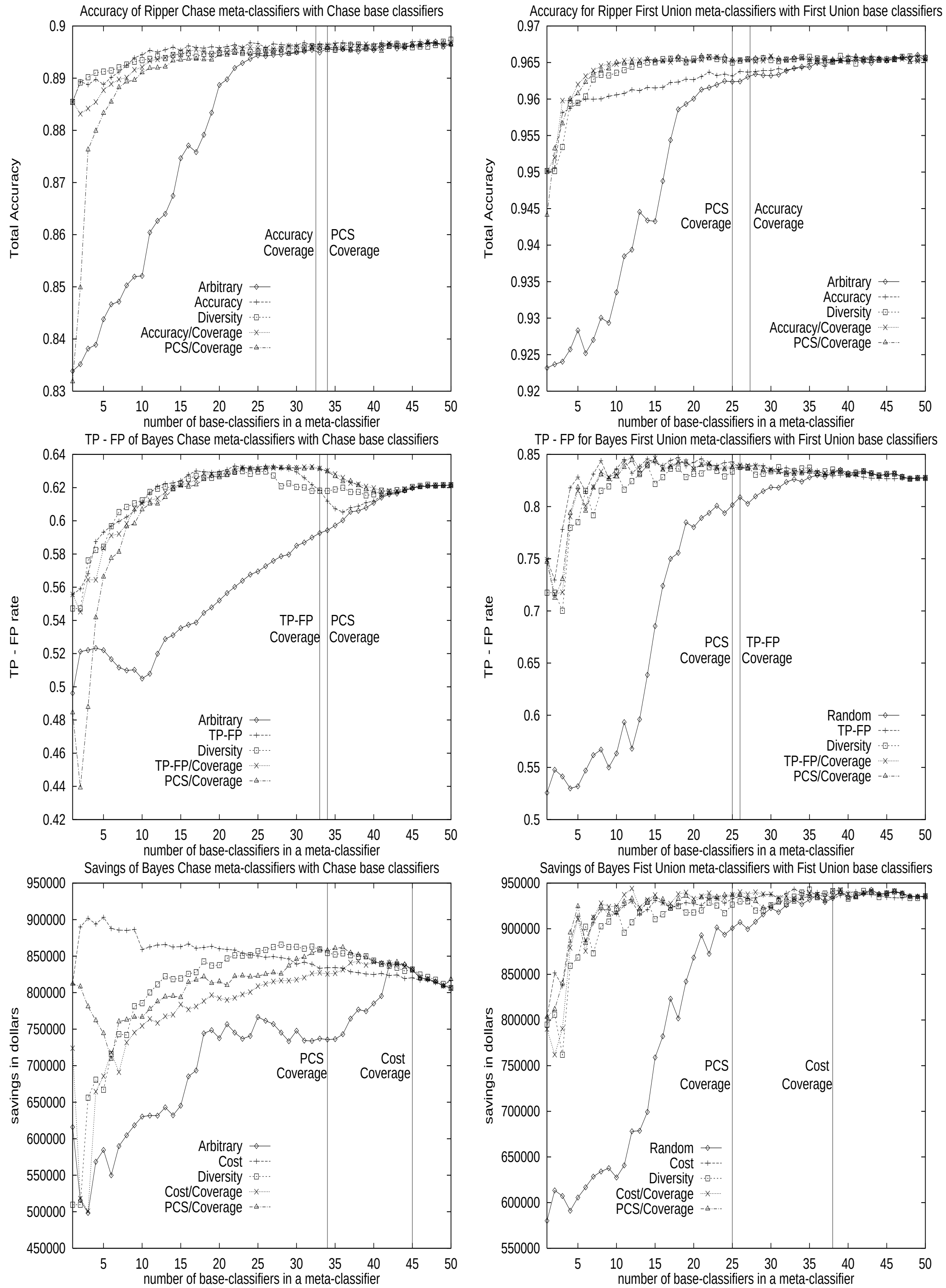

Figure 5.3: Average accuracy (top), TP-FP (middle) and savings (bottom) of the best meta-classifiers computed over Chase (left) and First Union (right) credit card data.

well as the performance results the redundant meta-classifiers would have had, had we used more base-classifiers or not introduced the pruning phase. Vertical lines for the diversity-based and metric-specific pruning algorithms are not shown in these figures as they depend on real-time constraints and available resources as discussed in Section 5.3.

Figure 5.3 reinforces the results of Table 5.1. The experiments demonstrate these algorithms to be successful in computing "good" combinations of base classifiers, at least, with respect, to the three evaluation metrics. Furthermore, these results establish that not all base classifiers are necessary in forming effective meta-classifiers. In all these cases the pruned meta-classifiers are more efficient (fewer base classifiers are retained) and at least as effective (accuracy, TP-FP, savings) as the unpruned meta-classifiers and certainly superior to those of the "arbitrary" pruning method. Using more base-classifiers than selected (denoted by the vertical lines) has no positive impact on the performance of the meta-classifiers. Overall, the pruning methods composed meta-classifiers with 1.2% higher accuracy, 7.8% larger TP-FP spread and $90K/month additional savings over the best single classifier for the Chase data and 1.65% higher accuracy, 10% larger TP-FP spread and $144K/month additional savings over the best single classifier for the First Union data. These pruned meta-classifiers also achieve 60% better throughput, 1% larger TP-FP spread and $90K/month additional savings than the unpruned meta-classifier for the Chase data and 100% better throughput, 1.7% larger TP-FP spread and $9K/month additional savings for the First Union data. Finally, pruning succeeded in further improving the performance of meta-classifiers over the single classifiers trained over the large data set of 200,000 examples (Chapter 4). These meta-classifiers increased the accuracy by 1.0%, the TP-FP rate by 7.4% and the savings by $60K/month for the Chase data set and 1.3%, 5.9% and $141K/month for the First Union data set, respectively.

Discussion

Since the performance of a meta-classifier is directly related to the properties and characteristics of its constituent base classifiers, by selecting the base classifiers, pruning helps address the cost-model problem as well as the TP-FP spread problem. Recall (from the left plot of the bottom row of Figure 4.2) that very few base classifiers from Chase have the ability to catch "expensive" fraudulent transactions. By combining only these classifiers, the meta-classifiers exhibit substantially improved performance (see the savings column of the Chase data in Table 5.1). This characteristic is not as apparent for the First Union data set since the majority of the First Union base classifiers happened to catch the "expensive" fraudulent transactions anyway (right plot of the bottom row of Figure 4.2). A similar phenomenon, although not as apparent, holds for the TP-FP spread as well.

A head to head comparison between the various pruning algorithms seems to point to a contradiction. The simple metric-specific pruning methods choose better combinations of classifiers for the Chase data, while the specialty/coverage-based and diversity-based pruning methods perform better on classifiers for the First Union data. This inconsistency, however, can be explained by examining the properties of the composing base classifiers. As we have already noted, the more diverse the set of base-classifiers is, the more room for improvement the meta-classifier has. In Chapter 4 we established that Chase classifiers are more diverse than the First Union classifiers. As a result, in the Chase bank case, the simple metric-specific pruning algorithm combines the best base classifiers that are already diverse and hence achieves superior results while the specialty/coverage pruning algorithms combine diverse base classifiers that are not necessarily the best. On the other hand, in the First Union case, the specialty/coverage pruning algorithms are more effective, since the best base-classifiers are not as diverse.

Furthermore, observe that in the First Union case the various pruning algorithms are comparably successful and their performance plots are less distinguishable. A closer inspection of the classifiers composing the pruned sets (C) revealed that the sets of selected classifiers were more "similar" (there were more common members) for First Union than for Chase. Actually, the inspection showed that for the First Union meta-classifiers, the specialty/coverage based and diversity-based pruning algorithms tended to select mainly the ID3 base-classifiers for being more specialized/diverse,[3] a fact that further substantiates the conjecture that the primary source of diversity for First Union meta-classifiers is the use of different learning algorithms. If the training sets for First Union were more diverse, there would have been more diversity among the other base-classifiers and presumably more variety in the pruned set. In any event, all pruning methods tend to converge after a certain point. After all, as they add classifiers, their pool of selected classifiers is bound to converge to the same set.

5.5.2 Comparing the Meta-Learning Algorithms

In the second part of the empirical evaluation of the pruning algorithms we investigate the effects of the best pruning algorithm on the performance of the meta-classifiers computed by the eight meta-learning algorithms as we vary the number of the pruned base classifiers.

Figure 5.4 displays the accuracy (top graphs), the TP-FP spread (middle graphs) and savings (bottom graphs) of the partially grown meta-classifiers for the Chase (left side graphs) and First Union (right side graphs) data sets respectively. The x-axis represents the number of base classifiers included in the meta-classifiers. The curves show the Ripper meta-classifier to be the most accurate and the Bayesian meta-classifiers to achieve the best performance with respect to the TP-FP spread and the cost model for both sets. At the opposite end, ID3 and in some cases the voting methods (i.e., in the TP-FP spread

[3]The ID3 learning algorithm is known to overfit its training sets. In fact, small changes in the learning sets can force ID3 to compute significantly different classifiers.

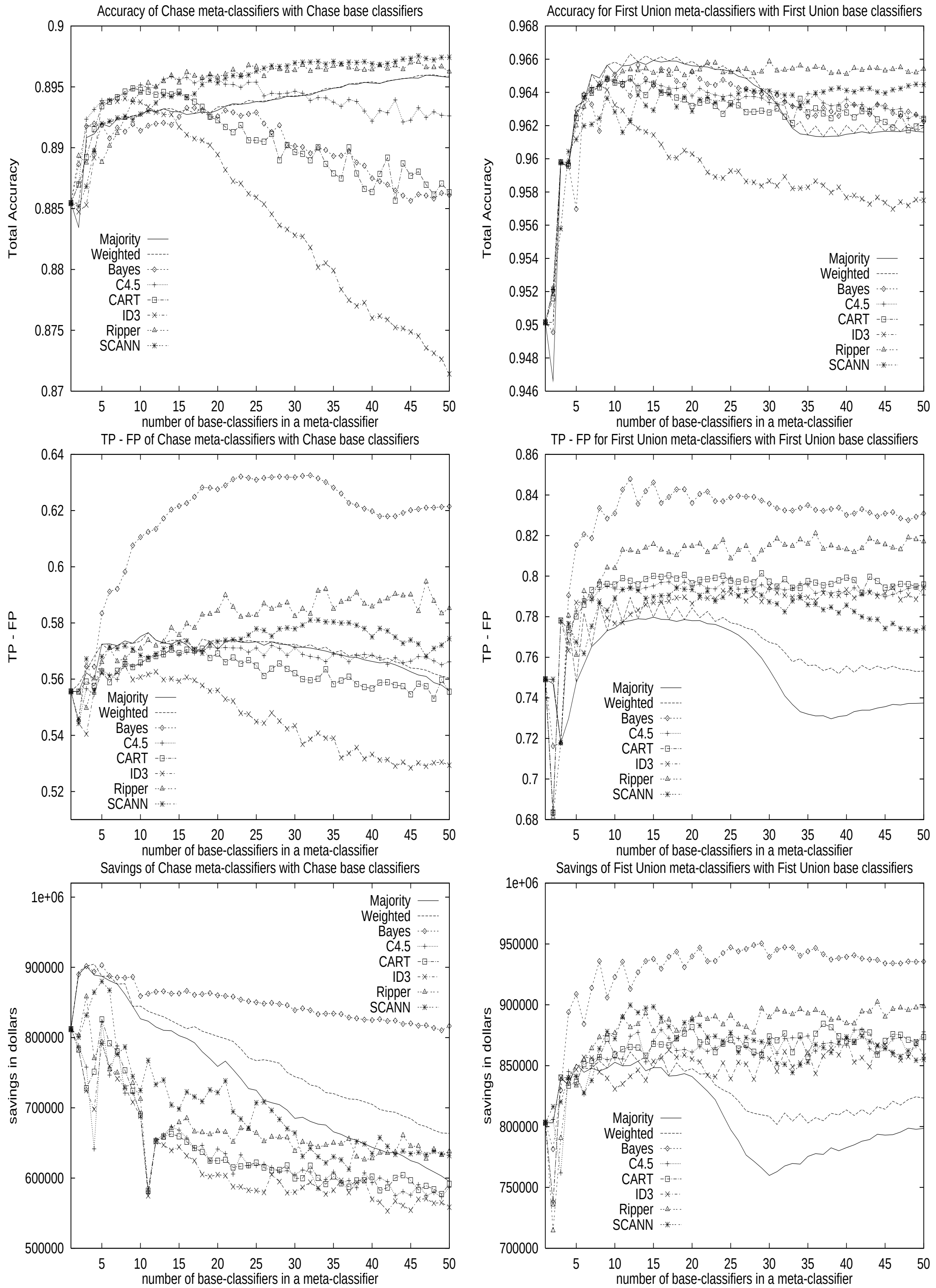

Figure 5.4: Best Pruning algorithm: Accuracy (top), TP-FP spread (middle) and savings (bottom) of Chase (left) and First Union (right) meta-classifiers.

and savings in First Union) are found to be the overall worst performers.

The figures indicate that, with only few exceptions, meta-learners tend to overfit their meta-learning training set (composed from the validation set) as the number of base-classifiers grows. Hence, in addition to computing most efficient meta-classifiers, pruning can lead to further improvements in the meta-classifiers' fraud detection capabilities. Moreover, by selecting the base classifiers, pruning helps address the cost-model problem as well. As we have already argued, the performance of a meta-classifier is directly related to the properties and characteristics of the constituent base classifiers. The conclusions from the discussion of Section 5.5.1 hold here as well. While the meta-classifiers consist of base classifiers with good cost model performance, they exhibit substantially improved performance as well (see the left plot of the bottom graph of Figure 5.4). The same, but to a lesser degree, holds for the TP-FP spread.

5.6 Conclusions

The efficiency and scalability of a distributed data mining system, such as the JAM meta-learning system, has to be addressed at two levels, the system architecture level and the data site level. In this chapter, we concentrated on the latter; we delved inside the data sites to explore the types, characteristics and properties of the available classifiers and deploy only the most essential classifiers. The goal was to reduce complex, redundant, inefficient and sizeable meta-learning hierarchies, while minimizing unnecessary system overheads.

Specifically, we examined three pre-training pruning algorithms each with different search heuristics. The first pre-training pruning algorithm ranks and selects its classifiers by evaluating each candidate classifier independently (metric-based), the second algorithm decides by examining the classifiers in correlation with each other (diversity-based), while the third relies on the independent performance of the classifiers and the manner in which they predict with respect to each other and with respect to the underlying data set (specialty/coverage-based).

The combination of various pruning algorithms with the multiple meta-learning techniques produces a large number of possible pruned meta-classifiers. Through an exhaustive experiment performed over the credit card data, adopting several performance measures (i.e., the overall accuracy, the TP-FP spread and the cost model) we evaluated the usefulness and effectiveness of our pruning methods, and at the same time we exposed their limitations and dependencies.

The empirical evaluation revealed that no specific and definitive strategy works best in all cases. Thus, for the time being, it appears scalable machine learning by meta-learning, remains an experimental art. (Wolpert's remark that stacking is a "black art" is probably correct [Wolpert, 1992].) The experiments, however, suggest that pruning base classifiers in meta-learning can achieve similar or better performance results than the brute-force assembled meta-classifiers in a much more cost effective way, especially when the learning algorithm's target function is different from the evaluation metric and only a small number of classifiers is appropriate. Moreover, the good news is, we demonstrated that a properly engineered meta-learning system does scale and does consistently outperform a single learning algorithm.

Chapter 6

A-posteriori Pruning of Meta-Classifiers

In the previous chapters we addressed the scalability and efficiency of JAM first by employing distributed and asynchronous protocols at the *architectural* level for managing the learning and classifier agents across the JAM sites of the system, and second by introducing special pre-training pruning algorithms at the *data site* (or *meta-learning*) level to evaluate and combine only the most essential classifiers. This chapter can be considered as a continuation of the latter task; we examine additional pruning algorithms, called *a-posteriori* pruning or *post-training* pruning algorithms, that aim to build smaller and faster meta-classifiers that achieve comparable performance (e.g., accuracy) as the unpruned meta-classifiers.

A-posteriori pruning or post-training pruning refers to the pruning of a meta-classifier after it is constructed. In contrast with the a-priori pruning, which uses greedy forward-search methods to choose classifiers, i.e., it starts with zero classifiers and iteratively adds more classifiers), post-training pruning is considered a backwards selection method. It starts with all available classifiers (or with the classifiers pre-training pruning selected) and iteratively removes one at a time. Again, the objective is to perform a search on the space of meta-classifiers and further prune the meta-classifier without degrading predictive performance.

To be more specific, this chapter describes two algorithms for pruning the ensemble meta-classifier as a means to reduce its size while preserving its accuracy. Both are independent of the meta-learning technique that computes the initial meta-classifier. The first is based on decision tree pruning methods and the mapping of an arbitrary ensemble meta-classifier to a decision tree model, while the second depends on the correlation between the meta-classifier and its constituent base classifiers. Through an extensive empirical study on meta-classifiers computed over the credit card data sets, we compare the two methods and illustrate the former post-training pruning algorithm to be a robust approach to discarding classification models while preserving the overall predictive performance of the ensemble meta-classifier.

6.1 Pruning Algorithms

The deployment and effectiveness of post-training pruning methods depends highly upon the availability of unseen labeled (validation) data. Post-training pruning can be facilitated if there is an abundance of data, since a separate labeled subset can be used to estimate the effects of discarding specific base classifiers and thus guide the backwards elimination process. A hill climbing pruning algorithm, for example, would employ the separate validation data to evaluate and select the best (out of K possible) meta-classifier with $K-1$ base classifiers, then evaluate and select the best meta-classifier with $K-2$ base classifiers (out of $K-1$ possible) and so on. In the event that additional data is not available, standard cross validation techniques can be used to estimate the performance of the pruned meta-classifier, at the expense of increased complexity.

A disadvantage of the above algorithm, besides the need for a separate data set and its vulnerability to the horizon effect (it only looks one step ahead), is the overhead of constructing and evaluating all the intermediate meta-classifiers ($O(K^2)$ in the example). In fact, depending on the combining methods (e.g., the learning algorithm in stacking) and the data set size, this overhead can be prohibitive. Next, we describe an efficient post-training pruning algorithm that does not require intermediate meta-classifiers or separate validation data. Instead, it extracts information from the ensemble meta-classifier and employs the meta-classifier training set seeking to minimize (meta-level) training error. Furthermore, it complies with the general requirement adhered by all our other pruning algorithms examined earlier in Chapter 5; namely to be compatible with any meta-learning technique.

6.1.1 Cost Complexity-Based Pruning

The algorithm is based on the *minimal cost complexity pruning* method of the CART decision tree learning algorithm [Breiman *et al.*, 1984]. CART computes a large decision tree T_0 that fits the training data, by allowing the splitting process to continue until all terminal nodes are either small, or pure (i.e., all instances belong to the same class) or contain only instances with identical attribute-value vectors. Next, it applies the cost complexity pruning method to compute a set of consecutive nested subtrees $T_i, i \in \{1, 2, ..., R\}$ of decreasing size from the original large tree T_0 by progressively pruning upwards to its root node (T_R corresponds to the subtree that consists of the root node only).

To compute the set of these nested subtrees $T_i, i \in \{1, 2, ..., R\}$, the cost complexity pruning method associates a complexity measure $C(T)$ with the number of terminal nodes of decision tree T. The method prunes decision trees by seeking to minimize a cost complexity metric $R_\alpha(T)$ that combines two factors, the size (complexity) and the accuracy (or, equivalently, the error rate) of the tree. Specifically, $R_\alpha(T)$ is defined as

$$R_\alpha(T) = R(T) + \alpha \cdot C(T) \tag{6.1}$$

where $R(T)$ denotes the misclassification cost (error rate[1]) of the decision tree T and α represents a complexity parameter ($\alpha \geq 0$).

The degree of pruning of the initial tree T_0 can be controlled by adjusting the complexity parameter α, which, according to Equation 6.1, corresponds to the weight of the complexity factor $C(T)$. If α is small, the penalty for having a large number of terminal nodes is small; as the penalty α per terminal node increases, the pruning algorithm removes an increasing number of terminal nodes in an attempt to compensate, thus generating the nested subtrees $T_i, i \in \{1, 2, ..., R\}$. The *minimal cost complexity pruning* method guarantees to find the "best" (according to the misclassification cost) pruned decision tree T_r, $r \in \{1, 2, ..., R\}$, of the original tree T_0 of a specific size (as dictated by the complexity parameter.)

Pruning Meta-Classifiers

The post-training pruning algorithm employs the minimal cost complexity method as a means to reduce the size (number of base-classifiers) of the meta-classifiers. In cases where the meta-classifier is built via a decision tree learning algorithm, the use of the cost complexity pruning method is straightforward. The ensemble meta-classifier is in a decision tree form with each node corresponding to a single base-classifier. Thus, by determining which nodes to remove, the post-training pruning algorithm discovers and discards the base classifiers that are least important.

To apply this method on meta-classifiers of arbitrary representation (i.e., non decision-tree), the post-training pruning algorithm maps the meta-classifier to its "decision-tree equivalent". In general, the algorithm has three phases. Phase one seeks to model the arbitrary meta-classifier as an equivalent decision tree meta-classifier that imitates its behavior. Phase two removes as many base-classifiers as needed using the minimal cost complexity pruning method on the derived decision tree model, and phase three re-combines the remaining base classifiers using the original meta-learning algorithm. Hereinafter, the term "decision tree model" will refer to the decision tree trained to imitate the behavior of the initial arbitrary meta-classifier.

These three phases are graphically illustrated in six steps in Figure 6.1. For completeness, a detailed description of the post-training pruning algorithm is provided in Figure 6.2.

[1]Estimated over a separate pruning subset of the training set or using cross validation methods.

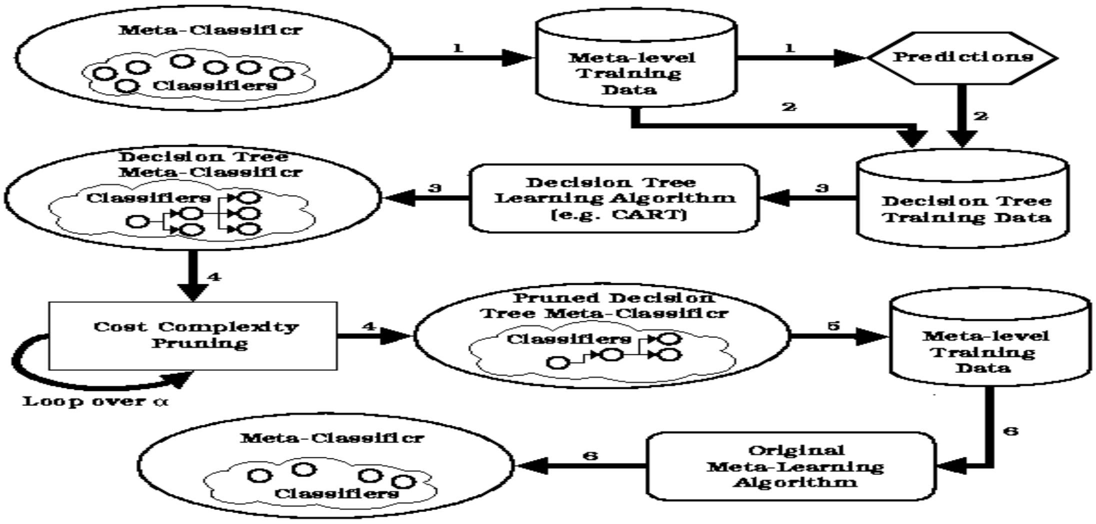

Figure 6.1: The six steps of the Cost Complexity-based Post-Training Pruning Algorithm.

1. The meta-classifier is applied to its own (meta-learning) training set. (Recall that the attributes of the meta-level training set correspond to the predictions of the base classifiers on the validation set and that the true class labels correspond to the correct classes of the validation set.)

2. A new training set, called decision tree training set, is composed by using the meta-level training set (without the true class labels) as attributes, and the predictions of the meta-classifier on the meta-level training set as the true class target.

3. A decision-tree-based algorithm, e.g., CART, computes the "decision-tree equivalent" of the original meta-classifier by learning its input/output behavior recorded in the decision-tree training set. The resultant decision-tree meta-classifier is trained to imitate the behavior of the original meta-classifier and discover the manner in which it combines its constituent base classifiers. Furthermore, this decision tree reveals the base classifiers that do not participate in the splitting criteria and hence are irrelevant to the meta-classifier. Those base classifiers that are deemed irrelevant are pruned in order to meet the performance objective.

4. The next stage aims to further reduce the number of selected base classifiers, if necessary, according to the restrictions imposed by the available system resources and/or the runtime constraints. The post-training pruning algorithm applies the minimal cost complexity pruning method to reduce the size of the decision tree and thus prune away additional base classifiers. The degree of pruning can be controlled by the complexity parameter α, as described in Equation 6.1. (The loop

Input: Set of base classifiers $\mathcal{C} = \{C_1, C_2, ..., C_K\}$, validation set $\mathcal{V} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$, meta learning algorithm $\mathbf{A_{ML}}$, meta classifier **MC**, decision tree algorithm $\mathbf{A_{DT}}$, throughput requirement (classifications/sec) T, convergence parameter $\delta(\delta > 0)$, stopping criteria $\epsilon(0 < \epsilon < \delta)$
Output: Pruned meta classifier $\mathbf{MC}^*$
Begin
$\mathcal{MLT} = \{< \vec{a}_1, \vec{a}_2, ..., \vec{a}_K, \vec{tc} >\}$, $\vec{a}_i[j] = C_i(\mathbf{x_j}) \forall i, \vec{tc}[j]$ = TrueClass($\mathbf{x_j}$), j=1, 2, ...,n, $x_j \in \mathcal{V}$;
$\mathcal{DTT} = \{< \vec{a}_1, \vec{a}_2, ...\vec{a}_K, \vec{pc} >\}$, predicted class $\vec{pc}[j] = \mathbf{MC}(\mathbf{x_j})$;
$\mathbf{DTM} = \mathbf{A_{DT}}(\mathcal{DDT})$, the decision tree model trained with $\alpha = 0$;
$K^* = K$; $\mathbf{DTM}^* = \mathbf{DTM}$; /* initialization of running variables */
$T^* = 1/$t, $t = max\{t_i | t_i = time(\mathbf{MC}(\mathbf{x_i}))$, i = 1, 2, ...,n }; /* Throughput estimate of **MC** */
While $(T^* < T)$ **do**
$\delta^* = \delta$; $K^* = K^* - 1$; /* remove one classifiers at a time */
L_1: **While** $\mathbf{DTM}^*$ has more than K^* classifiers **do**
$\alpha = \alpha + \delta^*$; /* increase α to remove classifiers */
$\mathbf{DTM}^* = \underset{i}{argmin}\{[R_\alpha(\mathbf{DTM_i}) + \alpha \cdot C(\mathbf{DTM_i})] | \mathbf{DTM_i}$ = subtree of $\mathbf{DTM}\}$;
end while
if ($\mathbf{DTM}^*$ has too many classifiers pruned (i.e., less than K^*) and $\delta^* > \epsilon$)
reset $\alpha = \alpha - \delta^*$; adjust $\delta^* = \delta^*/2$; $\mathbf{DTM}^* = \mathbf{DTM}$; **goto** L_1;
$\mathcal{C} = \mathcal{C} - \hat{C}$, $\hat{C}$ = the classifier that is not included in $\mathbf{DTM}^*$;
$\mathcal{MLT}^* = \{< \vec{a}_{j_1}, \vec{a}_{j_2}, ..., \vec{a}_{j_{K^*}}, \vec{tc} >\}$, $C_{j_1}, C_{j_2}, ..., C_{j_{K^*}} \in \mathcal{C}$ (retained classifiers);
$\mathbf{MC}^* = \mathbf{A_{ML}}(\mathcal{MLT}^*)$;
$T^* = 1/t$, $t = max\{t_i | t_i = time(\mathbf{MC}^*(\mathbf{x_i}))$, i = 1, 2, ...,n };
end while
$\mathcal{MLT}^* = \{< \vec{a}_{j_1}, \vec{a}_{j_2}, ..., \vec{a}_{j_{K^*}}, \vec{tc} >\}$, $C_{j_1}, C_{j_2}, ..., C_{j_{K^*}} \in \mathcal{C}$;
$\mathbf{MC}^* = \mathbf{A_{ML}}(\mathcal{MLT}^*)$;
End

Figure 6.2: Cost Complexity-based Post Training Pruning Algorithm.

in the figure corresponds to the search for the proper value of the α parameter). Since minimal cost complexity pruning eliminates first the branches of the tree with the least contribution to the decision tree's performance, the base classifiers pruned during this phase will also be the least "important" base classifiers.

5. A new meta-level training set is composed by using the predictions of the remaining base classifiers on the original validation set as attributes and the true class labels as target.

6. Finally, the original meta-learning algorithm is trained over this new meta-level training set to re-compute the pruned ensemble meta-classifier.

Remarks

The algorithm is guaranteed to terminate due to the finite structure of the decision tree and the bounded number of times ($\lceil \log_2(\delta/\epsilon) \rceil$) the inner loop of Figure 6.2 will be executed. Moreover, Theorem 3.10 of [Breiman *et al.*, 1984] proves the monotonic relation between the decreasing size of the decision tree as the complexity parameter α increases.

The success of the post-training pruning algorithm depends on the degree the decision tree learning algorithm "overfits" the original meta-classifier. We choose to train the model of the original meta-classifier on the very same data that was used to compute it in the first place. In some sense this is equivalent to generating test suits of test sets to exercise all program execution paths. By modeling the original meta-classifier based on its responses to its own training set we ensure that the decision tree learning algorithm has access to the most inclusive information regarding the meta-classifier's behavior. Furthermore, we do not require a separate *pruning* data set.

Another benefit of the post-training pruning algorithm comes from the fact that these intermediate decision tree models are also meta-classifiers and hence can too be used to classify unlabeled instances. Recall, that they are also trained over the predictions of the available base classifiers, albeit for a different target class. As a result, their predictive accuracy may be inferior to that of the original meta-classifier. On the other hand, in a distributed data mining system, such as JAM, where classifiers and meta-classifiers can be exchanged and used as black boxes, it may not be possible to prune the imported meta-classifiers to adhere to local constraints (e.g., if the original meta-learning algorithm is not known or not accessible). In this case, it may be preferable to trade some of their predictive accuracy for the more efficient pruned decision tree meta-classifiers.

Computing decision tree models as part of the post training pruning algorithm in not only useful for pruning or classifying unknown instances. By modeling the original meta-classifier decision tree models can also be used to explain the behavior of the meta-classifier. In general, an ensemble meta-classifier may have an internal representation that we cannot easily view or parse (except, of course, for its constituent base classifiers). By inducing a "decision tree equivalent" model, the algorithm generates an alternative and more declarative representation that we can inspect. This decision tree model, combined with certain *correlation* measurements, described next, can be used as a tool for studying and understanding the various pruning methods and for interpreting meta-classifiers. Other related methods for describing and computing comprehensible models of ensemble meta-classifiers have been studied in the contexts of Knowledge Acquisition [Domingos, 1997] and Knowledge Probing [Guo & Sutiwaraphun, 1998].

6.1.2 Correlation Metric and Pruning

Contrary to the diversity metric of Section 5.2.1, the correlation metric measures the degree of "similarity" between a pair of classifiers. Given $K+1$ classifiers C_1, C_2, ... C_K and C' and a data set of n examples mapped onto m classes, we can construct a two dimensional $m \times K$ contingency matrix $M^{C'}$ where each cell $M^{C'}_{ij}$ contains the number of examples classified as i by both classifiers C' and C_j. This means that if C' and C_j agree only when predicting class i, the $M^{C'}_{ij}$ cell would be the only non-zero cell of the j^{th} column of the matrix. If two classifiers C_i and C_j generate identical predictions on the data set, the i^{th} and j^{th} columns would be identical. And for the same reason, the cells of the j^{th} column would sum up to n only if C' and C_j produce identical predictions.

We call matrix $M^{C'}$ the *correlation matrix* of C' since it captures the correlation information of the C' classifier with all the other classifiers C_1...C_K. In particular, we define:

$$Corr(C', C_j) = \frac{\sum_{i=1}^{m} M^{C'}_{ij}}{n} \tag{6.2}$$

as the *correlation* between the two classifiers C' and C_j. *Correlation* measures the ratio of the instances in which the two classifiers agree, i.e., yield the same predictions.

In the meta-learning context, the correlation matrix M^{MC}_{ij} of the meta-classifier MC and the base-classifiers C_j can provide valuable information on the internals of the meta-classifier. If, n_i denotes the total number of examples predicted as i by the meta-classifier, then the ratio $\frac{M^{MC}_{ij}}{n_i}$ can be considered as a measure of the degree the meta-classifier agrees with the base classifier C_j about the i^{th} class. Furthermore, by comparing these ratios to the specialties of the base classifiers (calculated during the pre-training pruning phase), a human expert may discover that the meta-classifier has underrated a certain base classifier in favor of another and may decide to remove the latter and re-combine the rest. The correlations $Corr(MC, C_j)$ between the meta-classifier MC and the base classifier C_j, $1 \leq j \leq K$, can also be used as an indication of the degree the meta-classifier relies on that base classifier for its predictions. For example, it may reveal that a meta-classifier depends almost exclusively on one base classifier, hence the meta-classifier can be at least replaced by that single base classifier, or it may reveal that one or more of the base classifiers are trusted very little and hence the meta-classifier is at least inefficient. In fact, the correlation metric can be used as an additional metric for post-training pruning and for searching the meta-classifier space backwards.

A correlation-based post-training pruning algorithm that we consider as part of our experiments, removes the base classifiers that are least correlated to the meta-classifier. It begins by identifying and removing the base classifier that is least correlated to the initial meta-classifier, and continues iteratively, by computing a new reduced meta-classifier (with the remaining base classifiers), then it searches for the next least correlated base classifier and removes it. The method stops when enough base-classifiers are pruned. The advantage of this method is that it employs a different metric to search the space of possible meta-classifiers, so it evaluates different combinations of base classifiers not considered during the pre-training pruning phase. On the other hand, the least correlated base classifiers are not always the least important base classifiers and pruning them away may force the search process from considering "good" combinations.

6.2 Empirical Evaluation

This section describes a comprehensive set of experiments that evaluates our cost complexity-based post training pruning algorithm and then compares it against the correlation-based pruning algorithm. Specifically, we address the following questions:

- How accurately can a decision tree-based algorithm learn the behavior of a meta-classifier of arbitrary representation?
- What is the penalty of using the decision-tree meta-classifiers instead of the original meta-classifiers?
- How does the cost-complexity based pruning method compare to the correlation-based pruning algorithm?
- How robust is post-training pruning? What is the tradeoff between the predictive accuracy and the classification throughput of the meta-classifier as we increase the degree of pruning?

As with the pre-training pruning experiments, we compared the post-training pruning algorithms on meta-classifiers trained to predict credit card transactions as legitimate or fraudulent. Their effectiveness of the various classification models is evaluated from two perspectives, their predictive performance (accuracy, TP-FP spread and the cost model) and their efficiency. To measure the efficiencies of the pruned ensemble meta-classifiers we measured classification throughput. Throughput here denotes the rate at which a stream of data items can be piped through and labeled by a meta-classifier.

As before, the pruning algorithms are evaluated on their ability to choose the best set of base classifiers among the 50 that are available per data site. The difference in these tests is that the post-training pruning algorithms start with pre-computed (complete or pruned) meta-classifiers and search for less expensive but effective combinations of base classifiers. Furthermore, since it is trivial to learn a decision tree model of a decision tree meta-classifier, our starting set of unpruned meta-classifiers is limited to the meta-level classification models that can be derived from non-decision tree meta-learning techniques, i.e., from voting, from the Bayes and Ripper stacking methods and from SCANN.

6.2.1 Accuracy of Decision Tree Models

The first two phases of the cost-complexity based post-training pruning algorithm seek to learn and prune a decision tree model of the original meta-classifier. Recall that the performance of the base-level classification models and the meta-level ensemble classifiers that are under consideration, are shown in Figure 4.2 and Table 4.1 respectively.

To measure how accurately a decision tree-based algorithm can learn the behavior of a meta-classifier of arbitrary representation we compared the derived decision tree models against the original meta-classifiers. We began the experiment by applying the CART learning algorithm on the decision tree training set of each meta-classifier. The initial decision-tree models were generated with the complexity parameter α set to zero to allow the growth of trees that are as accurate and as "close" to the original meta-classifiers as possible. Then, by increasing the α parameter (i.e., the degree of pruning), we gradually derived trees with fewer nodes (and fewer base classifiers). Next, we quantified the effectiveness of these decision trees by comparing their predictions against the predictions of their corresponding original meta-classifiers when tested on separate data sets.

The accuracy of the decision tree models and the impact of pruning on their performance is shown in Figure 6.3. Since we employ decision tree algorithms to learn classification behavior, the emphasis in this experiment is on the accuracy results; the TP-FP spread and savings are of limited importance.

Each plot (left for Chase, right for First Union) displays the average accuracy (y-axis) of the decision tree algorithm when imitating the behavior of the 5 different methods for constructing ensembles. The x-axis represents decision-tree models of progressively smaller sizes. The initial (unpruned) model corresponds to the left-most points of each curve (i.e., degree of pruning is 0%). Naturally, an accuracy result of 100% would signify a decision tree model that has perfectly learned to imitate its meta-classifier. According to this figure, learning the behavior of these meta-classifiers has been fairly successful, with decision tree models achieving a 98%-99% accuracy rate, even when pruning is as heavy as 80%.

The last phase of the post-training pruning algorithm aims to re-combine the

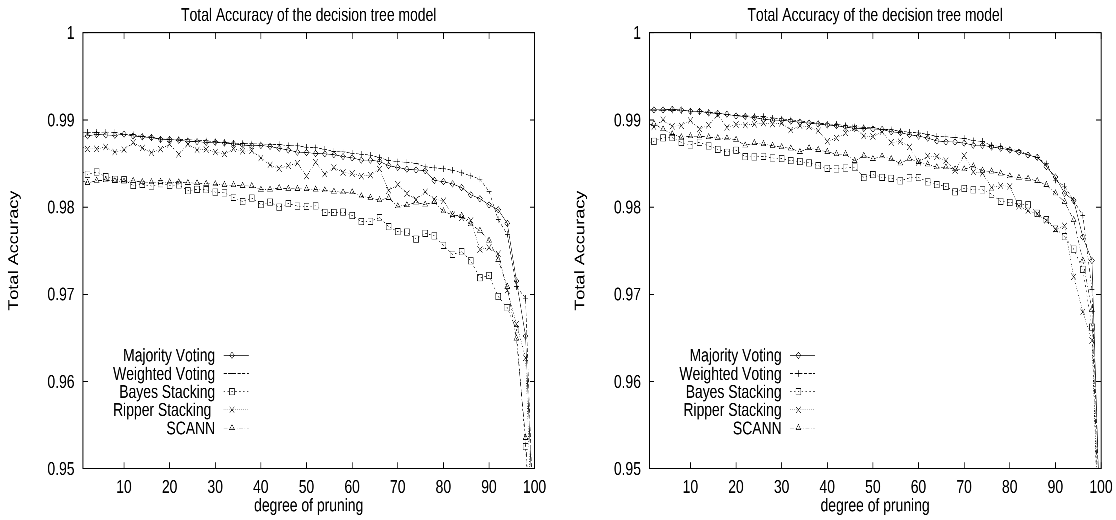

Figure 6.3: Accuracy of decision tree models when emulating the behavior of Chase (left) and First Union (right) meta-classifiers.

remaining base classifiers (those included in the decision tree model) using the original meta-learning algorithm. To evaluate the effectiveness of the pruning algorithm, we measured the accuracy of all the intermediate meta-classifiers obtained as we increased the degree of pruning from 0% (original unpruned meta-classifier) to 100% (all base classifiers are pruned and the default prediction corresponds to the most frequent class, i.e., the legitimate transaction).

In Figure 6.4 we present the accuracy results for the Chase (left column) and the First Union (right column) meta-classifiers. The plots at the top row correspond to the stacking meta-learning methods (Bayes, Ripper), the plots in the middle represent the voting (majority, weighted) meta-learning methods and the plots at the bottom row display the SCANN meta-learning method. These graphs demonstrate that post-training pruning was quite successful in all cases examined. The algorithm determined and pruned the redundant and/or the less “contributing” base classifiers, and generated substantially smaller meta-classifiers without significant performance penalties. In all cases, the pruned meta-classifiers are shown to exhibit predictive performance that is similar to that of the unpruned meta-classifiers, even when their degree of pruning is as high as 80%.

6.2.2 Decision Tree Models as Meta-Classifiers

The graphs also depict the overall accuracy of the decision tree models of the meta-classifiers. Recall that these are models trained to “imitate” the behavior of an ensemble, not to detect fraudulent transactions. As expected, their predictive performance is inferior to that of their corresponding meta-classifier. On the other hand, it is interesting to note

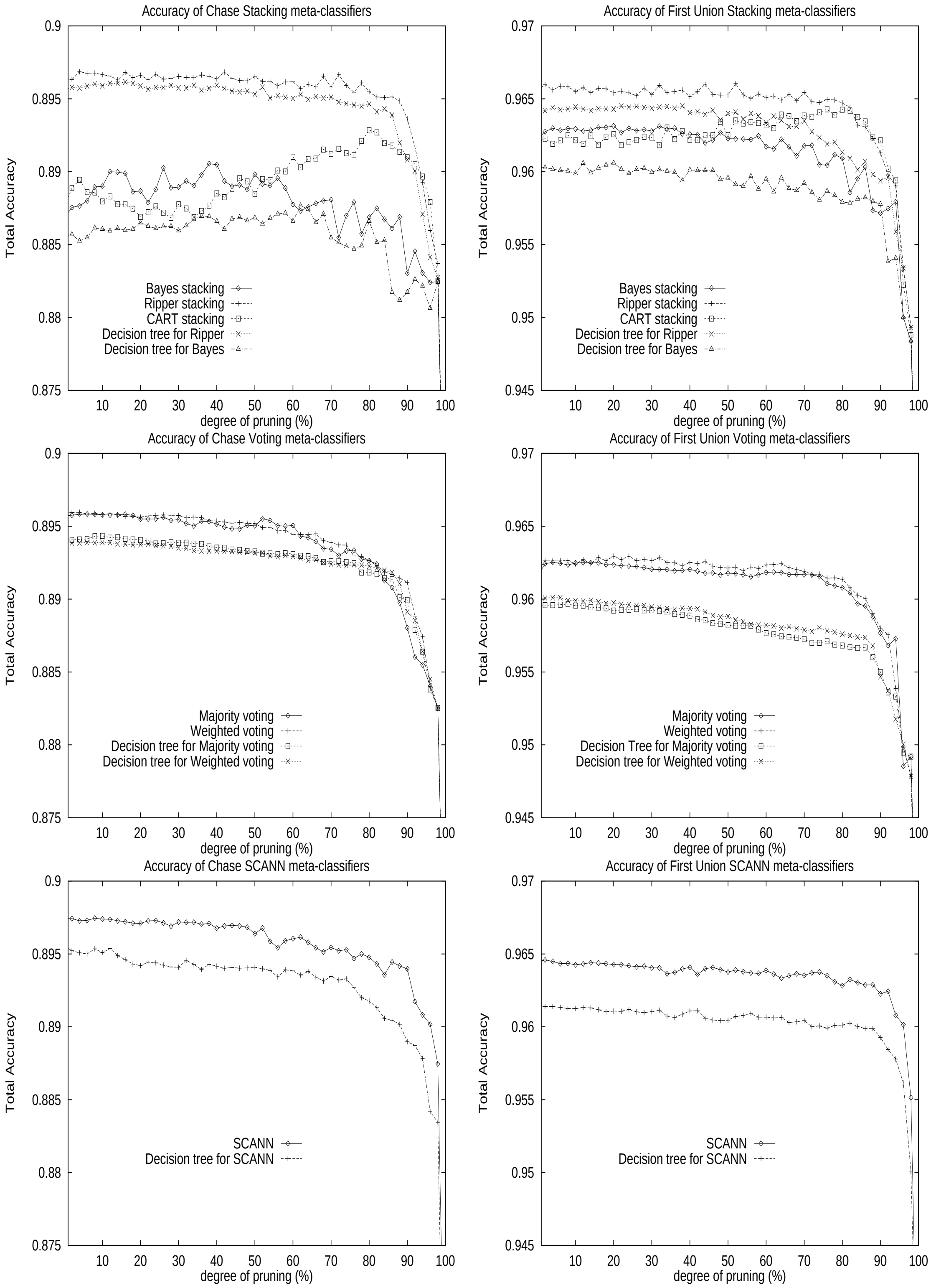

Figure 6.4: Accuracy of the Stacking (top), Voting (middle) and SCANN (bottom) meta-classifiers and their decision tree models for Chase (left) and First Union (right) data.

that it is possible to compute decision tree models that outperform other original meta-classifiers. In these experiments, for example, the decision tree model of Ripper (and those of the voting meta-classifiers and the SCANN meta-classifiers) outperforms the original Bayesian meta-classifier.

To further explore the last observation, we also measured the accuracy of the original stacking CART meta-classifier as a function of the degree of pruning. In other words, we studied the direct performance of a decision tree learning algorithm as a method of combining base classifiers. The results are depicted by the curve denoted as "CART stacking" in the top plots of Figure 6.4. Surprisingly, CART appears to be more effective when learning to combine base classifiers indirectly (e.g., by observing the Ripper meta-classifier) than through direct learning of this target concept. Although it is not entirely clear why this may be so, and is a subject of further study, this also suggests that searching the hypothesis space as modeled by a previously computed meta-classifier may, in some cases, be easier than searching the original hypothesis space. As a result, in cases where meta-classifiers are considered as black boxes, or meta-learning algorithms are not available, it may be beneficial to compute, prune and use their "decision tree equivalents" instead.

6.2.3 Cost Complexity vs. Correlation Pruning

The second post-training pruning algorithm operates by discarding the base classifiers that are least correlated to the meta-classifiers. In this section we compare the two post-training pruning methods by applying them to the meta-classifiers generated by the best meta-learning algorithms, the Ripper stacking method for accuracy and the Bayes stacking method for the TP-FP spread and the cost model.

The performance results of the pruned meta-classifiers for the Chase and First Union data sets are presented in Figure 6.5, with the Chase graphs at the left side, and the First Union graphs at the right side. As with the previous experiments, we display the accuracy (top plots), the TP-FP spreads (middle plots) and the savings (bottom plots) achieved.

According to these figures, in most of the cases, the cost-complexity post-training pruning algorithm computes superior meta-classifiers compared to those of the correlation-based pruning method. In general, the cost-complexity algorithm is successful in pruning the majority of the base-classifiers without any performance penalties. In contrast, the correlation-based pruned meta-classifiers perform better only with respect to the cost model and only for the Chase data set. This can be attributed, however, to the manner in which the specific meta-classifiers correlate to their base-classifiers and not to the search heuristics of the method. The pruning method favors (retains) the base classifiers that are more correlated to the Bayesian meta-classifier, which in this case happens to be the Bayesian base classifiers with the higher cost savings. As a result, the pruned meta-

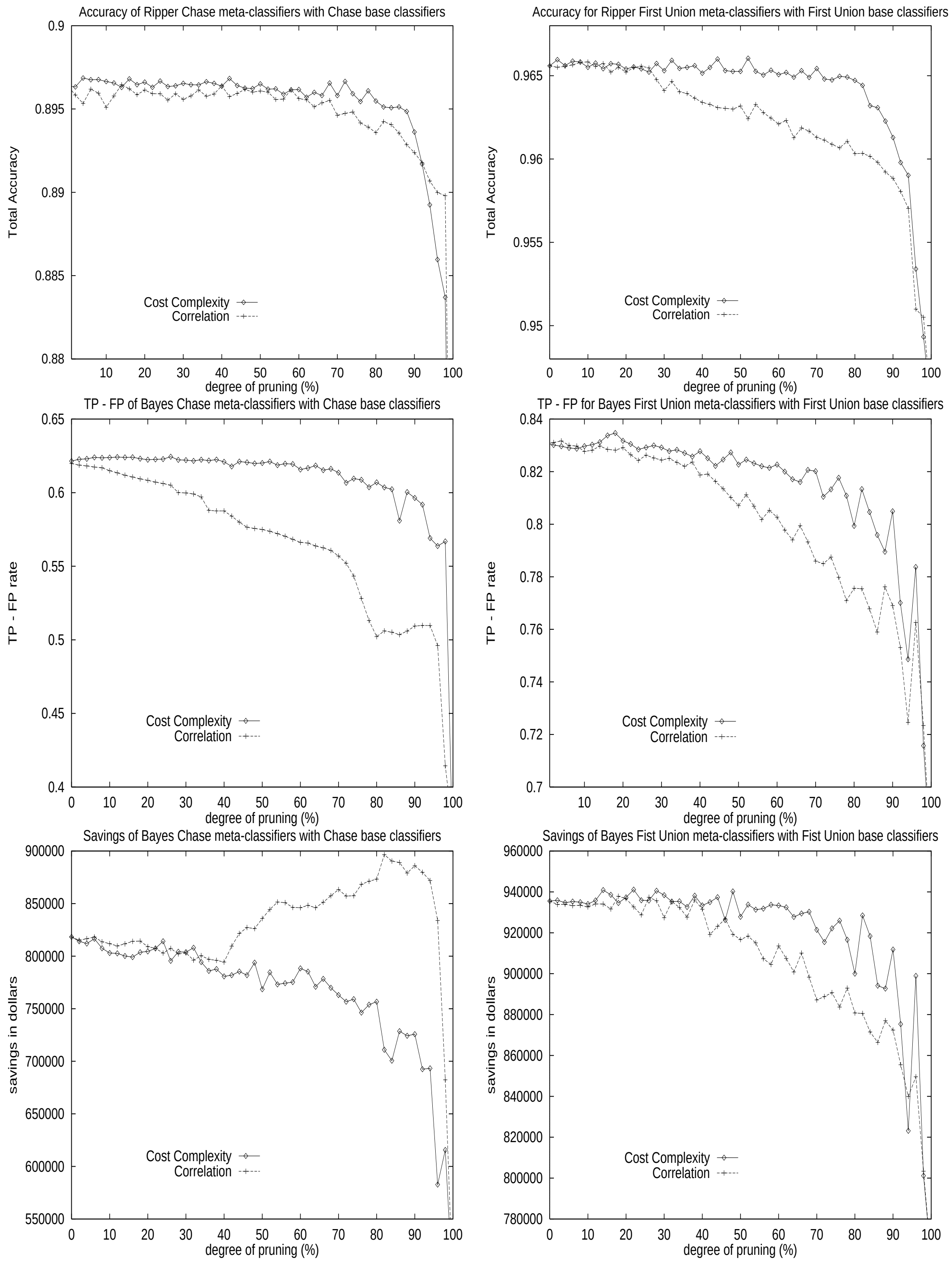

Figure 6.5: Post-training pruning algorithms: Accuracy (top), TP-FP spread (middle) and savings of meta-classifiers on Chase (left) and First Union (right) credit card data.

classifiers demonstrate increased savings even compared to the complete meta-classifier. Conversely, the cost-complexity pruning algorithm generates pruned meta-classifiers that best match the behavior and performance of the complete meta-classifiers. Overall, the correlation-based pruning cannot be considered a reliable pruning method unless only a few base classifiers need to be removed, while the cost-complexity algorithm is more robust and appropriate when many base classifiers need to be discarded.

6.2.4 Predictiveness/Throughput Tradeoff

The degree of pruning of a meta-classifier within a data site is dictated by the classification throughput requirements of the particular problem. In general, higher throughput requirements need heavier pruning. This last set of experiments investigates the trade-off between throughput and predictive performance.

To normalize the different evaluation metrics and better quantify the effects of pruning, we measured the ratio of the performance improvement of the pruned meta-classifier over the performance improvement of the complete (unpruned) meta-classifiers. In other words, we measured the performance *gain* ratio

$$G = \frac{P_{PRUNED} - P_{BASE}}{P_{COMPLETE} - P_{BASE}} \tag{6.3}$$

where P_{PRUNED}, $P_{COMPLETE}$ and P_{BASE} denote the performance (e.g., accuracy, TP-FP spread, cost savings) of the pruned meta-classifier, the complete meta-classifier and the best base classifier, respectively. Values of $G \simeq 1$ indicate pruned meta-classifiers that sustain the performance levels to that of the complete meta-classifier while values of $G < 1$ indicate performance losses. When only the best base classifier is used, there is no performance improvement and $G = 0$.

Figure 6.6 demonstrates the algorithm's effectiveness on the Chase (left plot) and First Union (right plot) classifiers by displaying the predictive performance and throughput of the pruned Ripper stacking meta-classifiers as a function of the degree of pruning. In this figure we have also included the performance results of the meta-classifier with respect to the other two evaluation metrics, the TP-FP spread, and the savings due to timely fraud detection. Almost identical results have been obtained for the other meta-classifiers as well. The black colored bars represent the accuracy gain ratios, the dark gray colored bars represent the TP-FP gain ratios and the light gray bars represent the savings gain ratios of the pruned meta-classifier. The very light gray bars correspond to the relative throughput of the pruned meta-classifier T_P to the throughput of the complete meta-classifier T_C.

To estimate the throughput of the meta-classifiers, we measured the time needed for a meta-classifier to generate a prediction. This time includes the time required to obtain the predictions of the constituent base classifiers sequentially on an unseen credit card transaction, the time required to assemble these predictions into a single meta-level

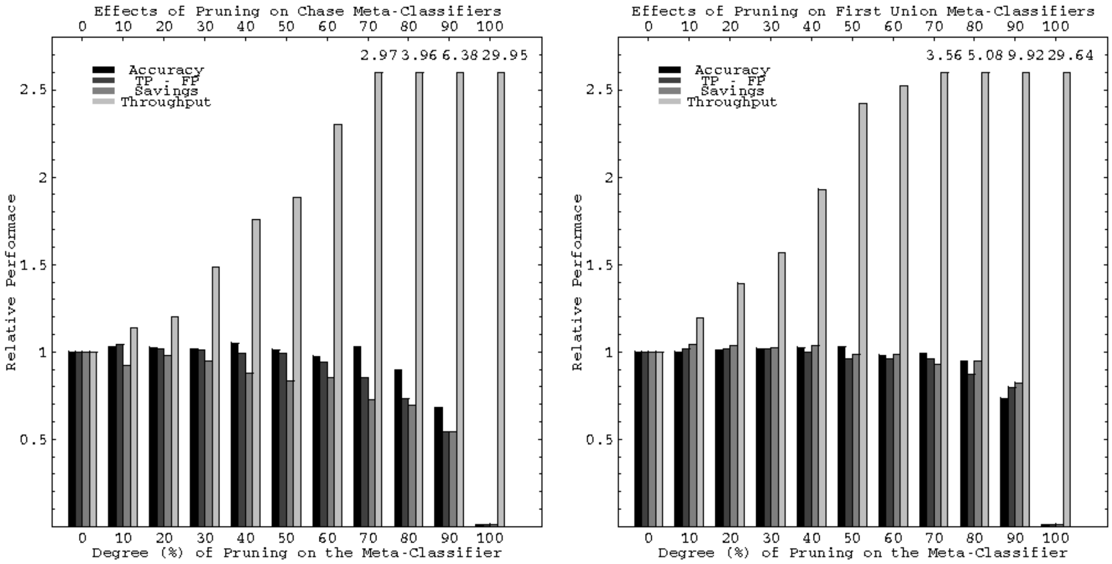

Figure 6.6: Bar charts of the accuracy (black), TP-FP (dark gray), savings (light gray) and throughput (very light gray) of the Chase (right) and First Union (left) meta-classifiers as a function of the degree of pruning.

"prediction" vector and the time required for the meta-classifier to input the vector and generate the final prediction. The measurements were performed on a PC with a 200MHz Pentium processor running Solaris 2.5.1. These measurements show that cost-complexity pruning is successful in finding Chase meta-classifiers that retain their performance levels to 100% of the original even with as much as 60% of the base classifiers pruned or within 60% of the original with 90% pruning. At the same time, the pruned classifiers exhibit 230% and 638% higher throughput. For the First Union base classifiers, the results are even better. With 80% pruning, the pruned meta-classifiers have gain ratios $G \simeq 1$ and with 90% pruning they are within 80% of the original performance. The throughput improvement in this case is 5.08 and 9.92 times better, respectively.

6.3 Combining Pre-Training and Post-Training Pruning

Pre-training pruning and post-training pruning are complementary methods that employ different metrics to discard base classifiers. It is possible to visualize and contrast the manner the different pruning methods select or discard the base classifiers by computing the meta-classifier correlation matrices (see Section 6.1.2) and by mapping them onto a modified *diversity* plot (see Section 4.4.1).

Figure 6.7, for instance, displays three plots for the Chase data set that represent the correlation between the meta-classifiers and the base classifiers as selected by the specialty/coverage (top left graph of Figure 6.7), the correlation-based (top right graph of Figure 6.7), and the cost-complexity (bottom center graph of Figure 6.7) pruning algorithms.

In these plots, each column corresponds to a single base classifier and each row corresponds to one meta-classifier, with the complete (unpruned) meta-classifier mapped on the bottom row, and the single base classifier mapped to the top. Dark gray cells represent base-classifiers that are highly correlated to their meta-classifier, while white cells identify the pruned (discarded) base classifiers. The 3-column density plots accompanying the correlation-density plot represent the performance of the individual meta-classifiers, namely their accuracy (left column), their TP-FP spread (middle column) and their savings (right column), with dark colors signifying "better" meta-classifiers.

Apart from exposing the relationships between meta classifiers and base classifiers, these plots provide information regarding the order the various base classifiers were selected or discarded. They can be read top-down for the pre-training pruning algorithms (e.g., the top left graph) and bottom-up for the post-training pruning algorithms (e.g., the top right and the bottom center graphs). The specialty/coverage method (top left graph), for example, begins by selecting a Ripper classifier (which corresponds to the January'96 data) and continues by adding mostly ID3 and CART classifiers. It also shows that the it selects meta-classifiers that are more correlated to the Ripper base classifiers than the rest.

The pruning algorithms examined in this thesis can be briefly described as iterative search methods that greedily (without back-tracking) add (in pre-training) or remove (in post-training) one classifier per iteration. Assuming that the data sets and the base classifiers are common to all pruning algorithms, the meta-classifiers computed by these methods (recall that the pruning methods discussed do not require intermediate meta-classifiers), depend on the starting point (e.g., the first base-classifier selected, the initial meta-classifier, etc.) on the search heuristic (metrics) evaluating the classifiers at each step, and on the portion of the validation data used for the evaluation. As a result, different pruning algorithms compute different combinations of classifiers and search distinct parts of the space of possible meta-classifiers. The correlation plots introduced above can serve as tools for studying the properties and limitations of the various pruning algorithms. Figure 6.7, for example, demonstrates that the three pruning algorithms consider very different combinations of base-classifiers. The first (specialty/coverage) tends to select diverse classifiers (i.e., ID3 and C4.5, c.f. to Figure 4.3), the second (correlation-based) clusters the classifiers by the algorithm type, while the third (cost-complexity) forms more complex relationships and thus appears more balanced and robust.

By allowing pre-training pruning to compute both the intermediate and the larger meta-classifiers (in a manner similar to that of Figure 5.3), and by applying post-training pruning to the best meta-classifier, it is possible to combine two different parts of the meta-classifier space and produce a final pruned meta-classifier that is even more effective and more efficient. The rationale parallels the strategy used in decision tree algorithms:

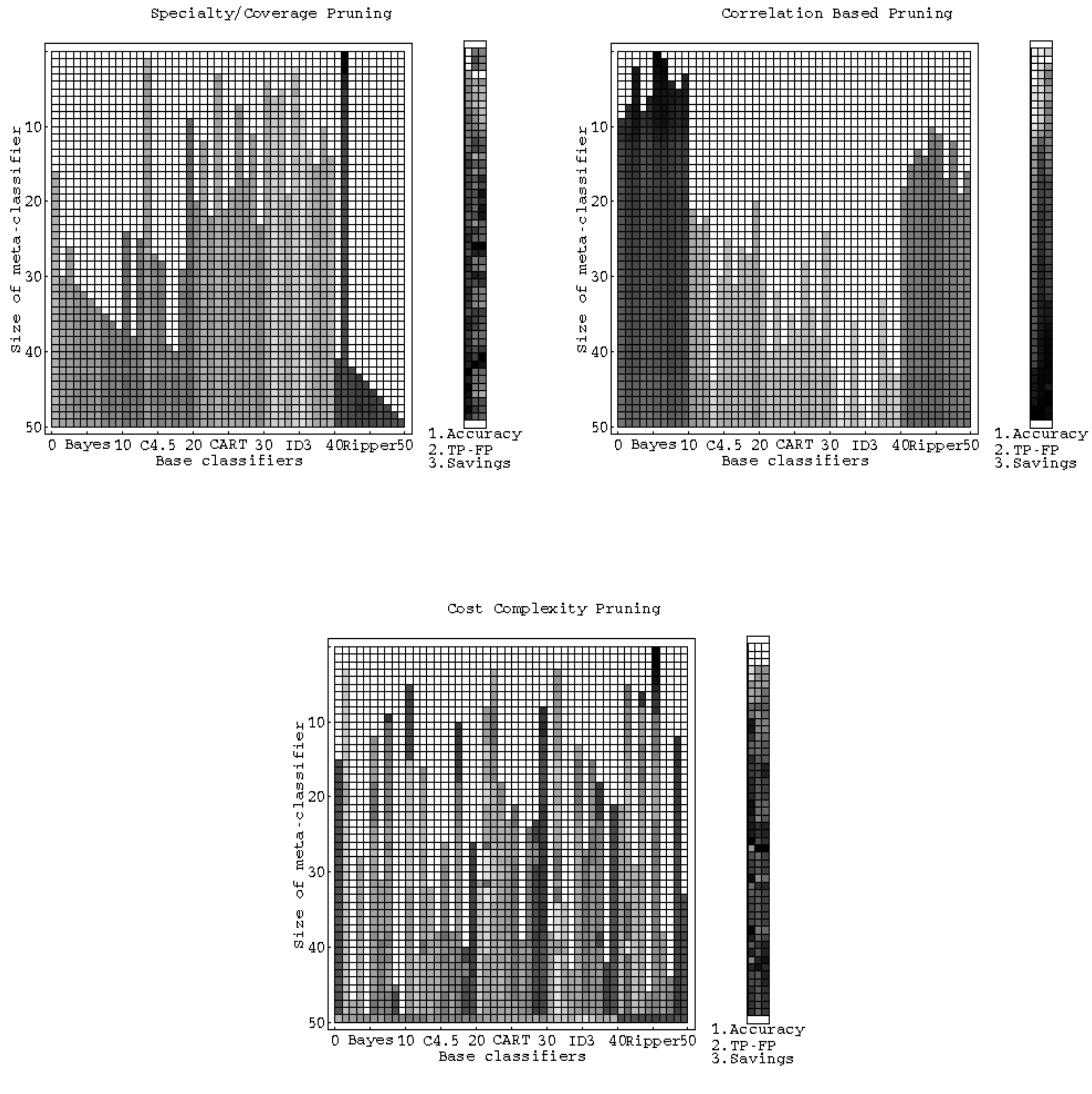

Figure 6.7: Selection of classifiers on Chase data: specialty/coverage-based pruning (top left) correlation-based pruning (top right) and cost-complexity-based pruning (bottom center).

Table 6.1: Performance and Relative Throughput of the Best Chase Meta-Classifiers.

	Accur.	Prune	T_P/T_C	TP - FP	Prune	T_P/T_C	Savings	Prune	T_P/T_C
Complete	89.74	0 %	1.0	0.621	0 %	1.0	818K	0 %	1.0
Pre Training	89.76	8 %	1.43	0.632	36 %	2.01	905K	90 %	6.5
+ Post (CC)	89.73	72 %	3.18	0.632	54 %	2.57	904K	92 %	8.25
+ Post (CB)	89.62	38 %	1.6	0.632	36 %	2.01	905K	94 %	10.11

first grow a large tree to avoid stopping too soon and then prune the tree back to compute subtrees with lower misclassification rate. In this case, post-training pruning is applied to the best intermediate meta-classifier that may not necessarily be the largest (complete) meta-classifier. This is particularly the case when the learning algorithm's target function is not aligned with the evaluation metric and only a small number of classifiers might be appropriate.

Table 6.2: Performance and Relative Throughput of the Best First Union Meta-Classifiers.

	Accur.	Prune	T_P/T_C	TP - FP	Prune	T_P/T_C	Savings	Prune	T_P/T_C
Complete	96.53	0 %	1.0	0.831	0 %	1.0	935K	0 %	1.0
Pre Training	96.59	54 %	2.48	0.848	72 %	4.33	944K	8%	1.16
+ Post (CC)	95.59	72 %	3.87	0.842	78 %	4.78	945K	48%	2.32
+ Post (CB)	96.59	54 %	2.48	0.849	78 %	5.25	944K	12%	1.25

Table 6.1 for the Chase data and Table 6.2 for the First Union data compare the effectiveness and efficiency of the complete meta-classifiers (first rows) to the most effective meta-classifiers computed by the pre-training pruning algorithms (second rows), the most effective meta-classifiers computed by the combination of the pre-training and the cost-complexity (CC) post-training pruning algorithms (third rows) and the most effective meta-classifiers computer by the pre-training and the correlation-based (CB) post training pruning algorithms (last rows). Effectiveness is measured by the meta-classifier's accuracy (first column, denoted by "Accur.") the meta-classifier's TP-FP spread (fourth column) and the meta-classifier's savings (seventh column) each accompanied by its degree of pruning (compared to the complete meta-classifier, denoted as "prune") and efficiency (relative throughput, denoted as T_P/T_C). As expected, there is no single best meta-classifier; depending on the evaluation criteria and the target learning task, different meta-classifiers should be constructed and deployed.

According to these experiments, pre-training pruning is able to compute more effective and more efficient meta-classifiers than those obtained by combining all available base classifiers, and post-training pruning and in particular the cost-complexity algorithm is successful in further reducing the size of these meta-classifiers. In fact, the larger the pre-trained pruned meta-classifier is, the larger the degree of additional pruning the cost-complexity algorithm exhibits. The post-training pruning algorithm that is based on the correlation metric, however, provides either limited or, is some cases, no further improvement, e.g., see the TP-FP column of Table 6.1. Another observation to note, is that the throughput of a meta-classifier is also a function of its constituent base classifiers. Some classifiers are faster in producing predictions than others (due to size, different representations, implementation details, etc.) and this may have a significant effect on the speed of the entire meta-classifier. As an example, observe the two post-trained pruned First Union meta-classifiers with the best TP-FP spreads of Table 6.2. They both have the same size, yet they are composed of different base classifiers and for that reason, they exhibit different throughputs.

6.4 Conclusions

In this chapter we continued the study of efficient and effective methods that combine multiple models, computed over distributed databases, into scalable and efficient meta-classifiers. Specifically, we addressed the shortcoming of meta-learning that has been largely ignored, the increased demand for run-time system resources on behalf of the meta-classifiers.

The final ensemble meta-classifier may consist of a large collection of base classifiers that require significantly more memory resources, while substantially slowing down classification throughput. We described a post-training pruning algorithm, called cost complexity-based pruning, and we compared it against a correlation-based pruning algorithm. The cost complexity-based method seeks to determine the base classifiers that are redundant or "less-important" to the classification process of the ensemble meta-classifier. Thus, given a set of system constraints and requirements, the algorithm discards these base classifiers and computes a smaller and faster ensemble meta-classifier.

On the same credit card transaction data sets from the Chase and First Union banks we demonstrated that post-training pruning base classifiers in meta-learning can achieve similar or better performance results than the brute-force assembled meta-classifiers at a substantially reduced computational cost and higher throughput. With the help of correlation-based visualization tools we analyzed base classifiers and meta-classifiers and we established that the suitability of a pre-training pruning method depends on the particular set of candidate base classifiers and the target learning task. Furthermore, we determined cost complexity-based post-training pruning to be an efficient, intuitive and robust method for discarding the base classifiers and we illustrated that combining cost-complexity pruning and pre-training pruning can yield meta-classifiers with further improved effectiveness and efficiency.

Chapter 7

Mining Databases with Different Schemata

To discover and combine useful information across multiple databases, the JAM system applies machine learning algorithms to compute models over distributed data sets and employs meta-learning techniques to combine the multiple models. Occasionally, however, these models (or classifiers) are induced from databases that have (moderately) different schemata and hence are incompatible. Alleviating the differences and bridging the JAM sites is the topic of this chapter.

As discussed in Chapter 4, one of the target applications of JAM in credit card fraud detection, is to allow financial institutions to share their models of fraudulent transactions without disclosing their proprietary data, thus meeting their competitive and legal restrictions. In all cases considered so far, however, the classification models were assumed to originate from databases with an identical schema. In fact, the experiments performed, described the meta-learning of base models derived from different disjoint subsets of the same database.

In the context where financial institutions cooperate and exchange their models, the identical schema assumption may not necessarily hold. Even though all institutions tend to record similar information, they may also include specific fields containing important information that each has acquired separately and that provides predictive value in determining fraudulent transaction patterns. Since classifiers depend directly on the format of the underlying data, minor differences in the schema between databases derive incompatible classifiers. Yet these classifiers are trained over the same concept and it is desirable to overcome the obstacles and combine them into a more effective higher level classification model.

Integrating the information captured by such classifiers is a non-trivial problem of great practical value. The reader is advised not to confuse this with schema integration over federated/mediated databases where the effort is towards the definition of a common schema across multiple data sets. We call it, *"the incompatible schema"* problem and is very common across the Internet (where multiple independently developed data sources exist) and the business world. Besides detecting credit card fraud across multiple financial institutions and the scenario where databases and schemata evolve over time, the incompatible schema problem can become a serious obstacle hampering rapid mergers between companies.

In this chapter we formulate the *incompatible schema* problem and detail several techniques that bridge the schema differences to allow JAM and other data mining systems to share *incompatible* and otherwise useless classifiers. Through experiments performed on the credit card data we evaluate the effectiveness of the proposed approaches and establish that combining models from different sources can substantially improve overall predictive performance.

7.1 Database Compatibility

Let's consider two data sites A and B with databases DB_A and DB_B respectively with similar but not identical schemata. For simplicity, and without loss of generality, we can assume that both databases define the same number of attributes (databases with different number of attributes are covered later as a specific case, e.g., if $B_{n+1} = \emptyset$):

$$Schema(DB_A) = \{A_1, A_2, ..., A_n, A_{n+1}, C\}$$
$$Schema(DB_B) = \{B_1, B_2, ..., B_n, B_{n+1}, C\}$$

where, A_i, B_i denote the i^{th} attribute of DB_A and DB_B, respectively, and C the class label (e.g., the fraud/legitimate label in the credit card fraud example) of each instance. Without loss of generality, we can further assume that $A_i = B_i$, $1 \leq i \leq n$. As for the A_{n+1} and B_{n+1} attributes, there are two possibilities:

1. $A_{n+1} \neq B_{n+1}$: The two attributes are of entirely different types drawn from distinct domains. In such a case, the problem can be reduced and examined as the two dual problems where one database has one more attribute than the other, i.e.,:

   $$\begin{aligned} Schema(DB_A) &= \{A_1, A_2, ..., A_n, A_{n+1}, C\} \\ Schema(DB_B) &= \{B_1, B_2, ..., B_n, C\} \end{aligned}$$

   and

   $$\begin{aligned} Schema(DB_A) &= \{A_1, A_2, ..., A_n, C\} \\ Schema(DB_B) &= \{B_1, B_2, ..., B_n, B_{n+1}, C\} \end{aligned}$$

   where we assume that attribute B_{n+1} is not present in DB_B in the first case and attribute B_{n+1} is not available in DB_A in its dual case.

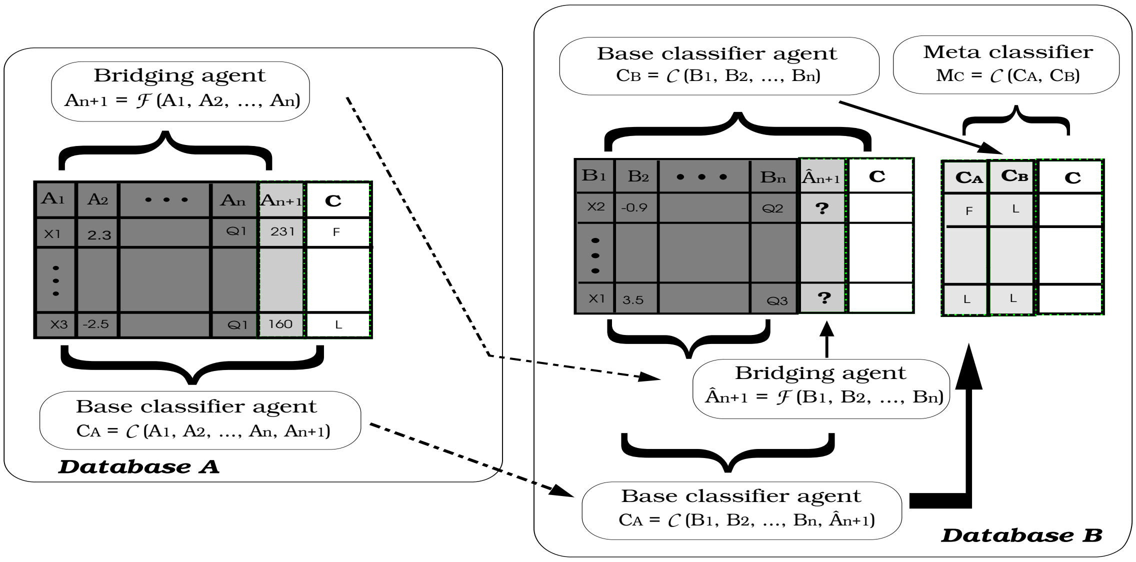

Figure 7.1: Bridging agents and Classifier agents transport from database A to database B to predict the missing attribute A_{n+1} and target class respectively.

2. $A_{n+1} \approx B_{n+1}$: The two attributes are of similar type but with slightly different semantics. In other words, there may be a mapping from the domain of one type to the domain of the other. For example, A_{n+1} and B_{n+1} can represent fields with time dependent information but of different duration (e.g., A_{n+1} may denote the number of times an event occurred within a window of half an hour and B_{n+1} may denote the number of times the same event occurred but within ten minutes).

In both cases (attribute A_{n+1} is either not present in DB_B or semantically different from the corresponding B_{n+1}) the classifiers C_{Aj} derived from DB_A are not compatible with DB_B's data and hence cannot be directly used in DB_B's site, and vice versa. Next section investigates ways, called *bridging* methods, that aim to overcome this incompatibility problem and integrate classifiers originating from databases with different schemata.

7.2 Bridging Methods

There are several possible approaches to address the problem depending upon the learning task and the characteristics of the different or missing attribute A_{n+1} of DB_B (or equivalently attribute B_{n+1} of DB_A):

- **Attribute A_{n+1} is missing, but can be predicted:** The method is presented in Figure 7.1. It may be possible to create an auxiliary classifier, which we call a *bridg-*

ing agent, from DB_A that can predict the value of the A_{n+1} attribute. To be more specific, by deploying regression methods (e.g., CART [Breiman *et al.*, 1984], locally weighted regression [Atkeson, Schaal, & Moore, 1999], linear regression fit [Myers, 1986], MARS [Friedman, 1991]) for continuous attributes and machine learning classification algorithms for categorical attributes, data site A can compute one or more auxiliary classifier agents $C_{Aj}{}'$ that predict the value of attribute A_{n+1} based on the common attributes $A_1, ..., A_n$. Then it can send all its local (base and bridging) classifiers to data site B. At the other side, data site B can deploy the auxiliary classifiers $C_{Aj}{}'$ to estimate the values of the missing attribute A_{n+1} and present to classifiers C_{Aj} a new database $DB_B{}'$ with schema $\{A_1, ..., A_n, \hat{A}_{n+1}\}$. From this point on, meta-learning and meta-classifying proceeds normally. That is, meta-classifiers receive the predictions of the individual (local and/or remote) classifiers and produce the final prediction.

- **Attribute A_{n+1} is missing and cannot be predicted:** Computing a model for the missing attribute assumes that there is a correlation between that attribute and the rest. Nevertheless, such a hypothesis may be unwarranted. In other cases, e.g., when an attribute is proprietary, modeling an attribute may be undesirable. In such situations, we adopt one of the following strategies:

  – **Classifier agent C_{Aj} supports missing values:** If the classifier agent C_{Aj} originating from DB_A can handle attributes with missing values, data site B can simply include null values in a fictitious A_{n+1} attribute added to DB_B. The resulting $DB_B{}'$ database is a database compatible with the C_{Aj} classifiers. Different classifier agents treat missing values in different ways. Some machine learning algorithms, for instance, treat them as a separate category, others replace them with the average or most frequent value, while most sophisticated algorithms treat them as "wild cards" and predict the most likely class of all possible, based on the other attribute-value pairs that are known.

  – **Classifier agents C_{Aj} cannot handle missing values:** If, on the other hand, the classifier agent C_{Aj} cannot deal with missing values, data site A can learn two separate classifiers, one over the original database DB_A and one over $DB_A{}'$, where $DB_A{}'$ is the DB_A database but without the A_{n+1} attribute:

  $$DB_A{}' = PROJECT\ (A_1, ..., A_n)\ FROM\ DB_A \tag{7.1}$$

  The first classifier can be stored locally for later use by the local meta-learning agents, while the later can be sent to data site B. Learning a second classifier without the A_{n+1} attribute, or in general with attributes that belong to the intersection of the attributes of the databases of the two data sites, implies that the second classifier makes use of only the attributes that are common among the participating data sites. Even though the rest of the attributes (i.e., not in the intersection) may have high predictive value for the data site that uses them, (e.g., data site A), they are of no value for the other data site (e.g., data site B). After all, the other data site (data site B) did not include them in its database, and presumably other attributes, including the common ones, do have predictive value.

- **Attribute A_{n+1} is present, but semantically different:** It may be possible to integrate human expert knowledge and introduce *bridging* agents either from data site A, or data site B that can preprocess the A_{n+1} values and translate them according to the A_{n+1} semantics. In the context of the example described earlier where the A_{n+1} and B_{n+1} fields capture time dependent information, the bridging agent may be able to map the B_{n+1} values into A_{n+1} semantics and present these new values to the C_{Aj} classifier. For example, the agent may estimate the number of times the event would occur in thirty minutes by tripling the B_{n+1} values or by employing other more sophisticated approximation formulas that rely on non uniformly distributed probabilities (e.g., Poisson).

These strategies systematically address the *incompatible schema* problem and meta-learning over these models should subsequently proceed in a straightforward manner. The idea of requesting missing definitions from remote sites (i.e., missing attributes) first appeared in [Maitan, Ras, & Zemankova, 1989]. In that paper, Maitan, Ras and Zemankova define a query language and describe a scheme for handling and processing global queries (queries that need to access multiple databases at more than one site) within distributed information systems. According to this scheme, each site compiles into rules some facts describing the data that belong to other neighboring sites that can subsequently be used to interpret and correctly resolve any non-standard queries posed (i.e., queries with unknown attributes).

More recently Zbigniew Ras in [Ras, 1998] further elaborated on this scheme and developed an algebraic theory to formally describe a query answering system for solving non-standard DNF queries in a distributed knowledge based system (DKBS). Given a non-standard query on a relational database with categorical or partially ordered set of attributes, his aim was to compute rules consistent with the distributed data to resolve unknown attributes and retrieve all the records of the database that satisfy it. Our approach, however, is more general, in that, it supports both categorical and continuous attributes, and it is not limited to a specific syntactic case or by the consistency of the generated rules. Instead, it employs machine learning techniques to compute models for the missing values.

7.3 Incorporating Bridging Agents in JAM

The JAM Classifier object (Section 3.4) is the core of a bridging agent object. After all, a bridging agent is, itself, a predictive model that is trained to estimate the value of a target attribute. In this case, however, the target attribute is not the class attribute of the database, but one of the missing (uncommon) attributes. To approximate the values of that attribute, the predictive model relies either on the values of the input (common) attributes (if it is a classification or regression model), or on a user-defined rule (when resolving semantic differences).

In addition to the JAM Classifier object, a bridging agent includes other components as well. The parent Bridge class defines a method for pre-processing the data sets to adhere to the specific format expected by its predictive model (the JAM Classifier object). For instance, its predictive model (classification, regression or rule) may expect to read the data sets as flat files with the last column allocated for the target attribute, while the underlying data set has positioned the target class in the first column. Furthermore, the Bridge class defines a method for populating the target (missing) attribute with the predicted values and a method for post-processing the resulting data sets to fit the format expected by the classifier agent.

To integrate the notion of bridging agents within JAM in a manner that is consistent with the design of the system, we altered the definition of the Classifier class to also include a vector of Bridge objects. The vector allocates one Bridge object for each attribute of the originating JAM site that is not present at the destination JAM site. When a JAM Client requests a Classifier object from another JAM site, the JAM Server serializes and sends each entry of the vector of Bridge objects as part of serializing and sending the requested Classifier agent. By de-serializing the receiving data stream, the JAM Client populates the vector of the Bridge objects and re-composes the Classifier agent. The Bridge agents are created upon request of Classifier objects. In more detail, the current version of the JAM system implements the following protocol:

- The JAM Client of a JAM site A issues a *JAMGetClassifiers* call to the JAM Server of another JAM site B to request a classifier C.
- The JAM Server of B requests the database schema description of JAM site A via its JAM Client and a *JAMGetDBProperties* call.
- A's JAM Server responds with the schema description.
- B's JAM Server sorts alphabetically the attribute names of A's database and compares them to the attribute names of its local database. For each attribute that is not present in A, the JAM Server computes a bridging agent and inserts it in the vector of Bridge Objects of classifier C. The particular method used for generating

a bridging agent (learning algorithm, regression technique, interpolation, etc.) is decided by the owner of JAM site B.

- B's JAM server returns classifier C and its bridging agents to A's JAM client.

The protocol is designed to comply to the interface published by JAM servers (see Table 3.2). It is possible to suppress or eliminate the second and third steps of the protocol in a future release of the JAM system, by allowing JAM Servers to cache the schema description, and/or by overloading the JAM Server interface (see Table 3.2) to support a *JAMGetAgent* and a *JAMGetClassifiers* methods that accept schema descriptions as input parameters (provided by the requesting JAM Client). Identifying attributes with syntactic or semantic differences when attribute names are identical, or distinguishing situations where names are different when in fact the attributes are the same, has not been addressed in this thesis. It is a matter of future research that entails the study and development of methods and languages for declaring and formally defining the schema of each database.

7.4 Empirical Evaluation

This stage of our experiments involves the exchange of base classifiers between the two banks. In addition to their 10 local and 50 "internal" classifier agents (those imported from their peer data sites), the data sites also import the 60 external classifier agents (those computed at the other bank). Each Chase data site is populated with 60 (10+50) Chase fraud detectors and 60 First Union fraud detectors and vice versa for First Union. Of the 120 available base classifiers, each site combines only the 110 remote classifiers (50 internal but imported and 60 external). To ensure fairness, the 10 local classifier agents of each site are not used in the local meta-learning. (Recall that these 10 base classifiers were trained on the local data that are subsequently also used in the pruning and the meta-learning and testing stages.) Note that the SCANN meta-learning technique is not used in this phase of the empirical evaluation, as being very expensive. The computer resources of our 300MHz PCs with 128MB of main memory and 465MB swap space were not sufficient to meta-learn more than 53 base classifiers over a typical meta-learning set of 42,000 examples.

7.4.1 Bridging Agents

To meta-learn over this set of classifier agents, however, we had to overcome additional obstacles. The two databases had differences in their schema definition, hence they produced *incompatible* classifiers:

1. Chase and First Union defined an attribute with different semantics (i.e., one bank recorded the number of times an event occurs within a specific time period while the

Table 7.1: Performance of the meta-classifiers.

	Chase			First Union		
Algorithm	Accuracy	TP-FP	Savings	Accuracy	TP-FP	Savings
Majority	81.44%	0.337	$ 281K	89.21%	0.474	$ 532K
Weighted	81.85%	0.343	$ 301K	89.61%	0.484	$ 541K
Bayes	67.05%	0.178	$ 176K	78.11%	0.424	$545K
C4.5	**83.78%**	0.320	$ 252K	89.96%	0.484	$ 541K
CART	82.15%	0.336	$ 303K	89.41%	0.493	$ **559K**
ID3	81.44%	**0.346**	$ **331K**	88.09%	**0.495**	$ 542K
Ripper	83.51%	0.278	$ 191K	**90.52%**	0.475	$ 521K

second bank recorded the number of times the same event occurs within a different time period).

2. Chase includes two (continuous) attributes not present in the First Union data.

For the first incompatibility, we had the values of the First Union data set mapped via interpolation to the semantics of the Chase data, while for the second incompatibility, we extended the First Union data set with two additional fields padded with null values to fit it to the Chase schema. Then we deployed classifier agents that support missing values. When predicting, the First Union classifiers disregarded the real values provided by the Chase data, while the Chase classifiers relied on the remaining attributes to reach a decision.

7.4.2 Meta-Learning External Classifiers Agents

Once transported to the other site, the base-classifiers are tested, evaluated, and combined into meta-classifiers using that site's local validation set. As a first step, we form meta-classifiers using only the external (from the other bank) base classifiers. The performance results of the unpruned meta-classifiers averaged over the 6 sites of each bank are reported in Table 7.1. The Chase meta-classifiers are composed solely of First Union base classifiers while the First Union meta-classifiers depend on Chase base-classifiers alone. The best result in every category is depicted in bold; the C4.5 and Ripper stacking methods are shown to form the most accurate meta-classifiers, ID3 composes the best performer with respect to the TP-FP spread and the ID3 and CART stacking techniques compute the most effective meta-classifiers measured by the cost model.

Although these meta-classifiers are inferior to those exhibited by the local meta-classifiers (Table 4.1), they are also successful in discerning a substantial portion of the fraudulent transactions. Compared to the default results (80% and 85% accuracy for Chase and First Union respectively, and 0 TP-FP spread and $0 savings for both banks) of not using any fraud detection method, we demonstrate that we can get real benefits just by importing and combining pre-computed classifiers.

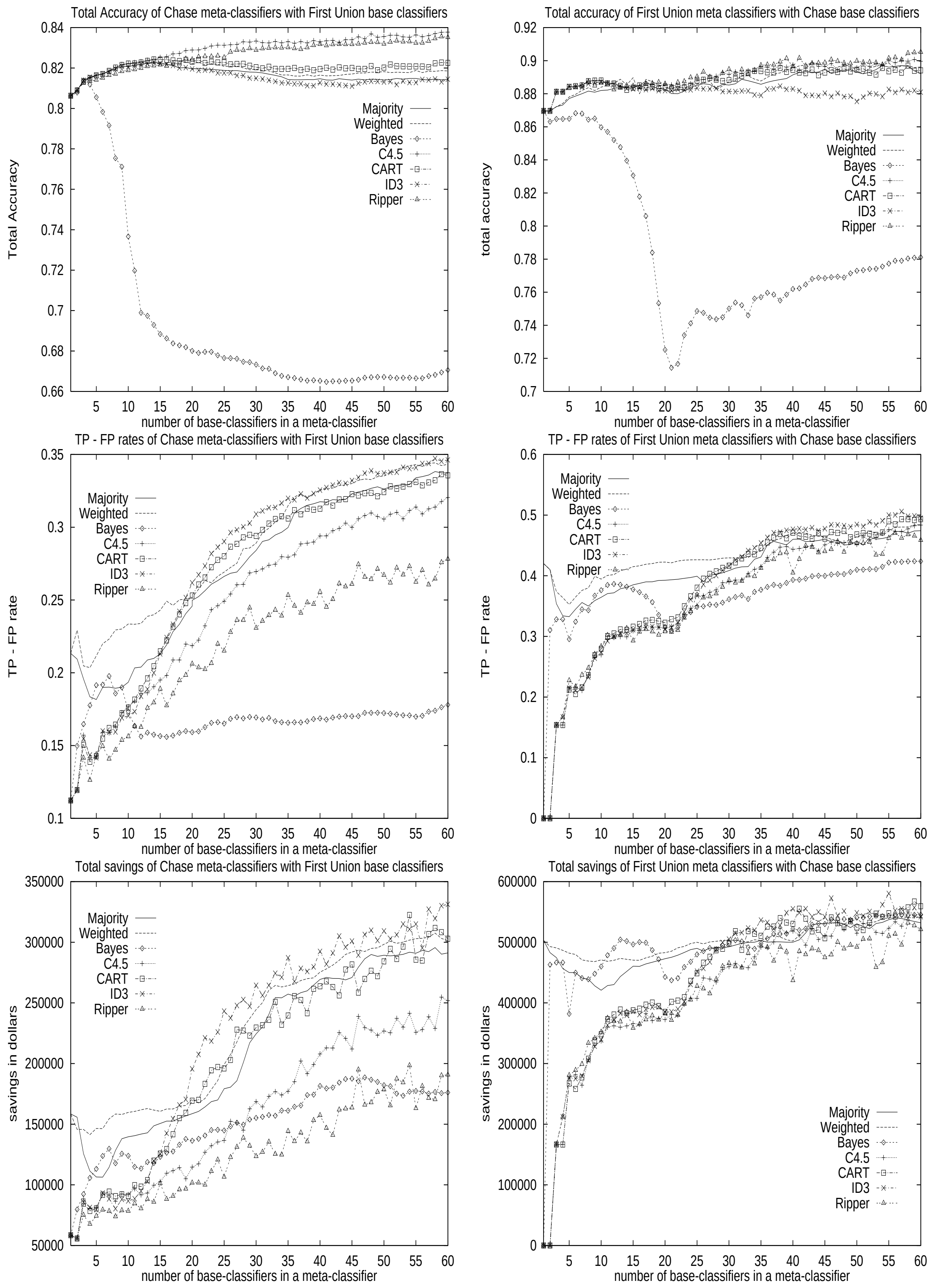

Figure 7.2: Accuracy (top), TP-FP spread (middle), and savings (bottom) of Chase-meta classifiers with First Union base classifiers (left) and First Union meta-classifiers with Chase base classifiers (right).

The results of applying the specialty/coverage pruning algorithm (Section 5.3) on the set of imported base classifiers are plotted in Figure 7.2. The graphs present the averaged (over the parallel 6-fold cross validation scheme) accuracy, the TP-FP spread and the savings of the Chase (three left-side graphs) and First Union meta-classifiers (three right-side graphs) on Chase and First Union data, respectively. The curves of each figure represent the performance of the 2 voting and 5 stacking meta-learning algorithms as they combine more base classifiers.

This experiment establishes that both Chase and First Union classifiers can be exchanged and applied to each of their respective data sets and catch significant amount of fraud despite the fact that the base classifiers are trained over data sets with different characteristics, patterns and fraud distribution.

7.4.3 Performance of Bridging Agents

With two features missing from the First Union data, Chase base classifiers are unable to take full advantage of their knowledge regarding credit card fraud detection. In an attempt to alleviate the problem, we deployed regression techniques as a means to approximate the missing values of the missing features of the First Union data. In the next experiment, we tested two different regression methods, namely, linear regression fit and regression trees, both available by Splus [Sta, 1996]. We applied these two methods on the 12 subsets of the Chase data to generate multiple bridging models and then we matched each (original) Chase classifier with one bridging classifier according to their performance on a First Union validation set. In Figure 7.3 we display the performance (accuracy, in the top left graph, TP-FP spread in the top right graph and total savings in the bottom center graph) of the Chase base classifiers with (black bar) and without (grey bar) the assistance of these bridging models. The first 12 bars represent the classifiers derived by the Bayesian learning algorithm when trained against the 12 different months of data, followed by the 12 C4.5, the 12 CART, the 12 ID3 and the 12 Ripper classifier agents.

The degree to which the base classifiers can benefit from bridging classifiers depends on the "importance" of the missing feature (e.g., its information gain or predictive value), the "dependence" of this feature on the remaining known attributes, the bias of the bridging learning algorithm and the quality of the training and testing data. In this case, as the figures demonstrate, the bridging classifiers are quite beneficial for the majority of the Chase base classifiers. On a couple of cases, the TP-FP spread remains negative despite the additional information provided by the bridging classifiers. Situations with inferior classifiers, such as this, are addressed in accordance with the methods described in Chapters 5 and 6. The meta-learning phase and the pruning algorithms are responsible for evaluating and pruning away these poor performers.

The improvement in the performance of the individual classifiers is attributed to the extra information provided by the bridging agents. This can be explained by con-

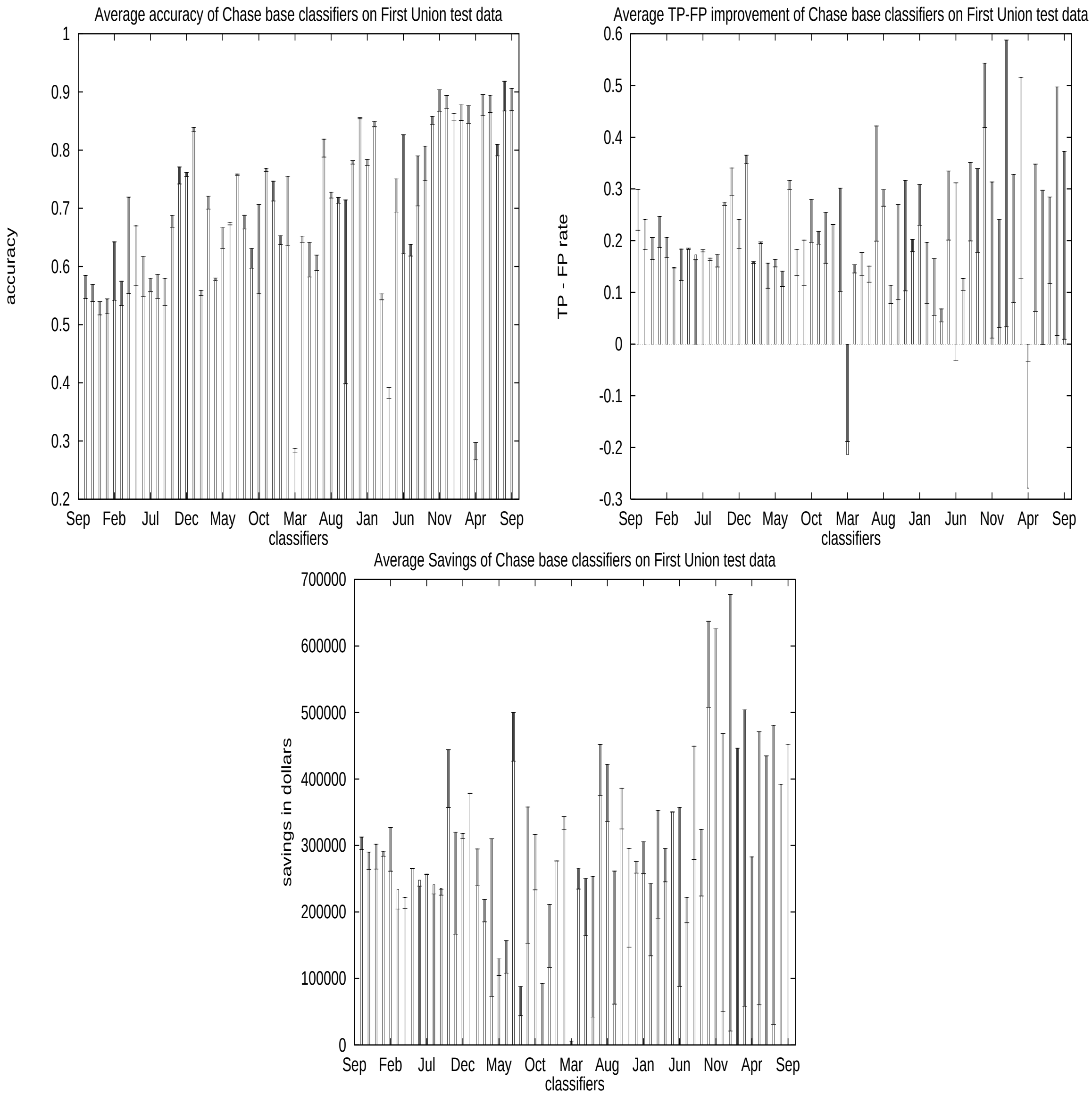

Figure 7.3: Accuracy (top left), TP-FP spread (top right) and total savings (bottom center) of plain Chase base-classifiers (grey bars) and of Chase base-classifier/bridging-agent pairs (black bars) on First Union data. The first 12 bars, from the left of each graph, correspond to the Bayes base classifiers each trained on a separate month, the next 12 bars represent the C4.5 classifiers, followed by 12 CART, 12 ID3 and 12 Ripper base classifiers.

Table 7.2: First Union meta-classifiers.

	Accuracy	TP-FP	Savings
Majority	95.81%	0.789	$ 705K
Weighted	96.11%	0.803	$ 715K
Bayes	88.25%	0.640	$660K
C4.5	**96.61%**	0.820	**$ 852K**
CART	96.39%	0.816	$ 831K
ID3	95.87%	**0.830**	$ 830K
Ripper	96.31%	0.803	$ 766K

sidering the assumptions made by the learning algorithms: to simplify the hard problem of searching for a viable hypothesis, learning algorithms examine the "predictiveness" of each attribute independently of the rest. As a result, base classifiers are incognizant of any beneficial association that may exist between input attributes. The bridging agents aim to uncover and exploit this overseen information, if it exists. In this case, they tapped on the redundancy that was existent in the credit card databases and successfully re-composed the missing values.

7.4.4 Meta-Learning External Base-Classifiers with Bridging Agents

After being matched and evaluated, a base-classifier/bridging classifier pair is considered as a single classifier agent that can be integrated into new meta-classifier hierarchies. In Table 7.2 we present the performance results of these First Union meta-classifiers. This experiment is analogous to the one reported in Table 7.1, except that each base classifier is augmented with a bridging agent. Again, the best result in every category is depicted in bold; the C4.5 stacking method is shown to form the most accurate and the most cost-model effective meta-classifier, while ID3 exhibits the best performance with respect to the TP-FP spread.

In Figure 7.4 we present the performance results of the pruned First Union meta-classifiers as computed by the specialty/coverage algorithm (Section 5.3). By contrasting the accuracy (top left), the TP-FP spread (top right) and the total savings (bottom center) plots of this figure against the three corresponding right-side plots of Figure 7.2 we demonstrate that meta-learning over the new set of the improved base classifiers (due to the bridging agents) results in significantly superior meta-classifiers. This translates into 6.1% higher accuracy, 0.325 larger TP-FP spread and $293K in additional savings than without the bridging agents, which is comparable the results achieved by meta-classifiers composed of internal (First Union) base classifiers (c.f. Table 4.1).

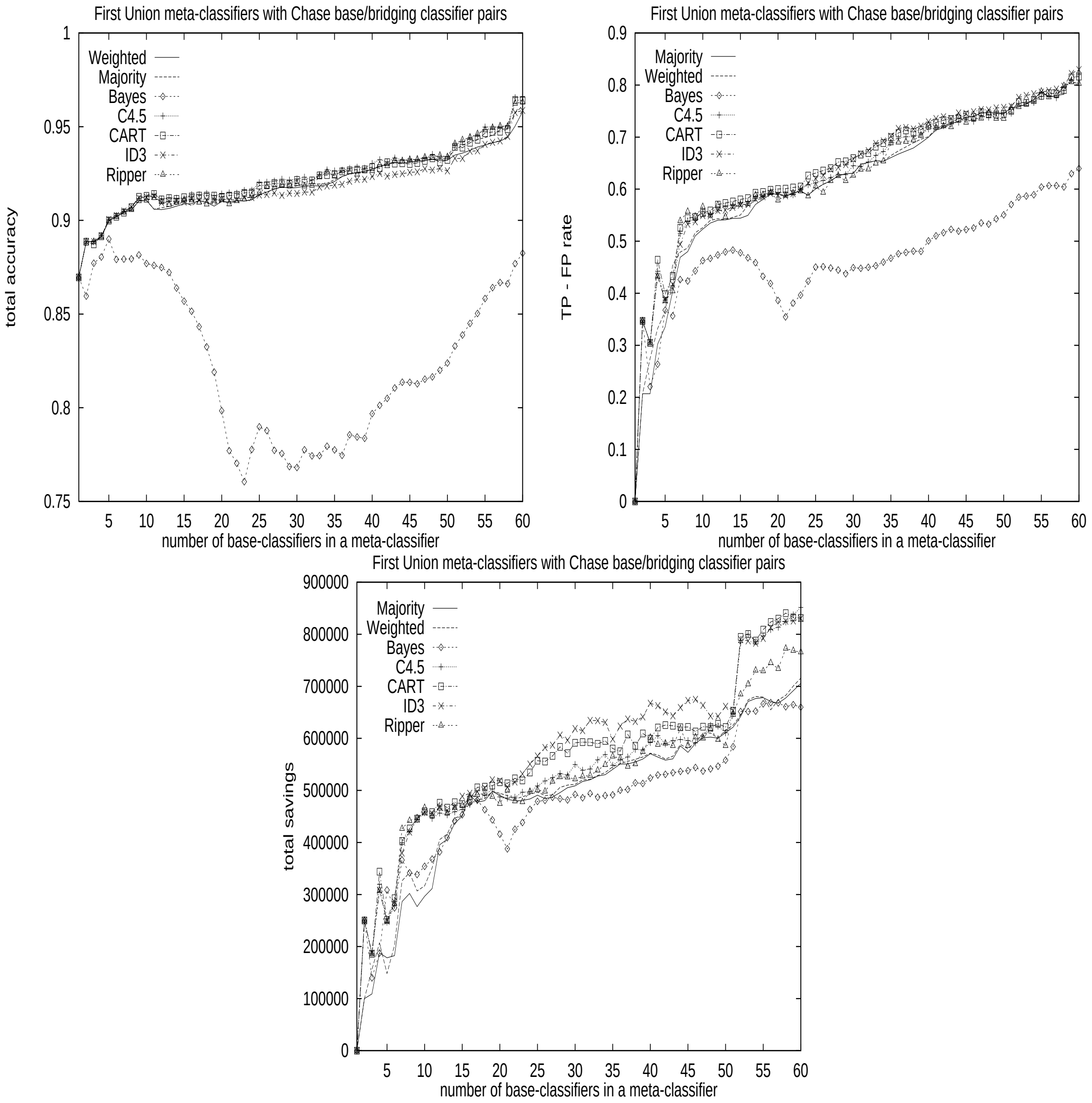

Figure 7.4: Accuracy (top left), TP-FP spread (top right) and savings (bottom center) of First Union meta-classifiers composed by Chase base-classifier/bridging-agent pairs.

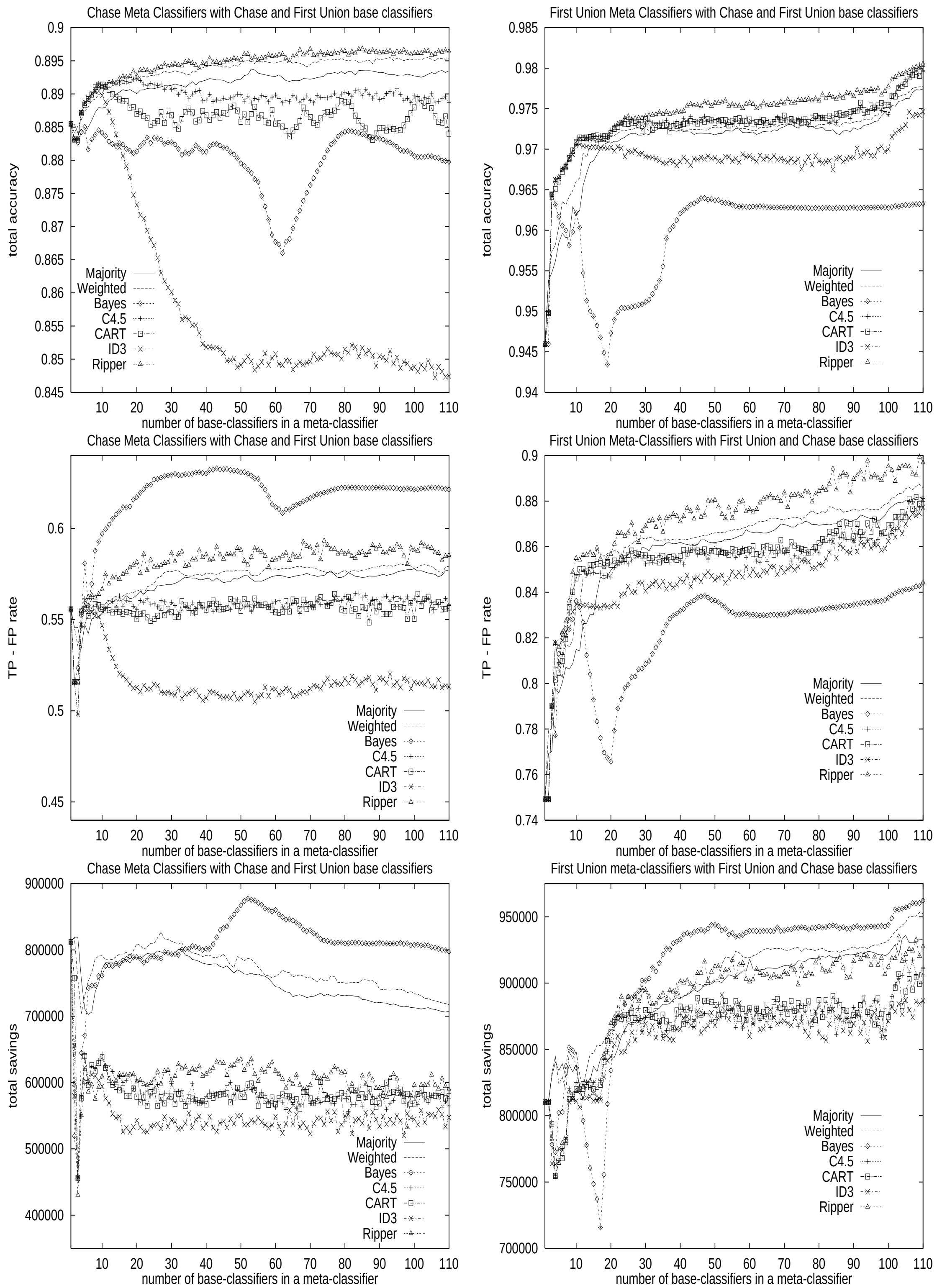

Figure 7.5: Accuracy (top), TP-FP spread (middle) and savings (bottom) of Chase (left) and First Union (right) meta-classifiers composed by Chase and First Union base classifiers.

Table 7.3: Performance of the meta-classifiers.

	Chase			First Union		
Algorithm	Accuracy	TP-FP	Savings	Accuracy	TP-FP	Savings
Majority	89.41%	0.577	$ 707K	**97.74%**	**0.882**	$ **932K**
Weighted	89.57%	0.579	$ 717K	**97.77%**	**0.886**	$ **952K**
Bayes	88.03%	0.621	$ 797K	**96.32%**	**0.844**	**$ 963K**
C4.5	88.93%	0.558	$ 564K	**97.97%**	**0.881**	**$ 912K**
CART	88.40%	0.556	$ 579K	**97.99%**	**0.881**	**$ 909K**
ID3	84.80%	0.513	$ 548K	**97.46%**	**0.877**	**$ 932K**
Ripper	89.70%	0.585	$ 589K	**98.05%**	**0.897**	**$ 927K**

7.4.5 Meta-Learning Internal and External Base-Classifiers

The final experiment combining classifiers, involves the meta-learning of base classifiers derived from both banks. Table 7.3 displays the accuracy, TP-FP spread and savings of each Chase and First Union meta-classifier. These results establish that both Chase and First Union fraud detectors can be exchanged, combined and applied to their respective data sets. The most apparent outcome of these experiments is the superior performance of the First Union meta-classifiers and the lack of improvement on the performance of the Chase meta-classifiers. In this table, entries in bold indicate a further improvement in performance of the meta-classifier with 99% statistical significance as compared by the 2-tailed paired t-test. This phenomenon can be easily explained in conjunction with the previous analysis that demonstrated that the missing attributes from the First Union data set are significant in modeling the Chase data set. (The First Union classifiers were not as effective as the Chase classifiers on the Chase data, and the Chase classifiers could not perform at full strength at the First Union sites without the bridging agents.)

Finally, Figure 7.5 displays the performance of the specialty/coverage pruning algorithm as it combines all available base classifiers, the internal Chase base classifiers and the external First Union base classifiers for Chase and the internal First Union base classifiers with the external Chase base/bridging classifier pairs for First Union. While no significant improvement is exhibited by the Chase meta-classifiers, the new First Union meta-classifier are superior even to the best “pure” First Union meta-classifiers (i.e., the meta-classifier composed by local base-classifiers alone) as reported in Table 6.2, improving total accuracy by 1.5%, TP-FP spread by 5.3% and savings by another $20K.

7.5 Conclusions

This chapter identifies the *incompatible schema* problem as a potential obstacle in distributed data mining. This problem refers to the case where classifier agents can be computed over similar databases but with some differences in their schema definition.

Distributed data-mining systems need to be flexible to accommodate such incompatible yet comparable databases. In this chapter we showed that, depending on the task and the particular incompatibility, it may be possible to combine the information carried by the different classifiers into a higher level meta-classifier. Through the use of the credit card transaction data sets, we demonstrated that special *bridging* agents can be trained at one database to predict the values of the missing attributes of the other database. These bridging agents compose an intermediate layer that provides the means to alleviate the differences among database with different schemata, and allows the exchange of useful classifiers, a facility that can result in significant performance improvements.

Chapter 8

Conclusions

The main focus of this thesis is on the **Management of Intelligent Learning Agents in Distributed Data Mining Systems**, or in other words, on the management of machine learning programs with the capacity to travel between computer sites to mine the local data. The term "management" denotes the ability to dispatch and exchange such programs across data sites, but also the potential to control, evaluate, filter, resolve compatibility problems and combine their products (that can too be intelligent agents).

Data mining refers to the process of extracting automatically, or semi-automatically novel, useful, and understandable pieces of information (e.g., patterns, rules, regularities, constraints) from data in large databases. One way of acquiring knowledge from databases is to apply various machine learning algorithms that search for patterns that may be exhibited in the data and compute descriptive representations.

Machine learning and classification techniques have been successfully applied in many problems in diverse areas with very good success. Although the field of machine learning has made substantial progress over the past few years, both empirically and theoretically, one of the continuing challenges is the development of inductive learning techniques that effectively scale up to large and possibly physically distributed data sets. Most of the current generation of learning algorithms are computationally complex and require all data to be resident in main memory, which is clearly implausible for many realistic problems and databases. Furthermore, in certain situations, data may be inherently distributed and cannot be merged into a single database for a variety of reasons including security, fault tolerance, legal constraints, competitive reasons, etc. In such cases, it may not be possible to examine all of the data at a central processing site to compute a single global model.

This dissertation investigates data mining techniques that scale up to large and physically distributed databases. In this respect, we designed and implemented JAM (Java agents for Meta-learning), an agent-based distributed data mining system that supports the remote dispatch and exchange of learning agents across multiple data sources and employs meta-learning techniques to combine the separately learned models into a higher level representation. Next, we summarize the contribution of our thesis work and discuss possible future research directions.

8.1 Results and Contributions

Meta-learning is a recent technique that seeks to compute higher level models, called meta-classifiers, that integrate in some principled fashion the information gleaned by the separately learned classifiers to improve predictive performance. By supporting meta-learning, the JAM system allows each data site to utilize its own local data and, at the same time, benefit from the data that is made available by the other data sites.

In the course of the design and implementation of JAM, we exposed several obstacles and addressed many issues related to distributed data mining systems, including dealing with heterogenous platforms, dealing with multiple databases and (possibly) different schemata, with the design and implementation of scalable protocols among the data sites, and with the efficient use of the models that are collected from remote data sites. Other important problems, intrinsic to data mining systems that were also tackled include, first, the ability of the system to adapt to environment changes (e.g., when data or targets change over time), and second, the capacity to extend it with new machine learning methods and data mining technologies.

To be more specific, JAM is:

- **scalable** to operate efficiently and without substantial or discernable reduction in performance as the number of data sites increases. The system was designed with asynchronous and distributed communication protocols that enable the participating database sites to function independently and collaborate with other peer sites as necessary, thus eliminating centralized control, congestion and synchronization points;

- **portable** to execute on multiple platforms including Solaris, Windows and Linux. It was built upon existing agent infrastructure available over the Internet using Java technology and algorithm-independent meta-learning techniques;

- **adaptive** to accommodate new input patterns as those change over time. This was achieved by facilitating the generation of new models based on the newly collected data and by extending meta-learning techniques to incorporate and combine them with existing classifiers;

- **extensible** with plug-and-play capabilities. Snapping-in new data mining technologies was attained via a well-defined object-oriented design that enabled the decoupling of the JAM system from the learning algorithms and the meta-learning techniques;

- **efficient** to make effective use of the system resources. By introducing special pruning methods before and after the meta-learning of classifiers, we provided a facility for the analysis of the tradeoff between predictive accuracy and efficiency and a means for addressing the shortcoming of meta-learning, namely its increase demand for run-time system resources;

- **compatible** to facilitate the use and the meta-learning of classifiers trained over databases with different schemata. Through the definition and deployment of specific bridging agents, we successfully resolved the differences between databases that would otherwise render meta-learning worthless;

- **effective** to compute highly predictive and practical classification models. This property is attained by allowing the evaluation of the generated models under various metrics depending on the particular task and the imposed constraints, including accuracy, true positive and false positive rates, performance under realistic cost models, classification throughput etc.

JAM release 3.0 is available in the public domain and is being used by many researchers world-wide. To our knowledge, JAM is the first distributed system that uses meta-learning as a means to mine knowledge from multiple databases with the ability to scale as the number of databases and data sites increase. JAM is novel in that it integrates all the above properties in a single coherent distributed data mining system while providing a general framework for evaluating (even over different time periods when data sets change) a large variety of learning and meta-learning techniques and data mining approaches.

Through an extensive empirical analysis of JAM on a real-world data mining task, we evaluated the effectiveness of our approaches and demonstrated their potential utility. Furthermore, we identified the shortcomings of the existing evaluation metrics in the context of many realistic problems and defined a new cost-sensitive metric to achieve more objective and meaningful evaluations among the various learning and meta-learning methods. Based on this metric, we introduced and compared three pre-training pruning algorithms that help reduce the size of meta-classifiers. Although we found no specific strategy to work best in all cases, our empirical study suggests that pruning base classifiers in meta-learning and using the cost-sensitive metric achieves similar or better performance results than the brute-force assembled meta-classifiers in a much more cost effective way, especially when the learning algorithm's target function is different from the evaluation metric and only a small number of classifiers is accurate. Moreover, the good news is, we demonstrated that a properly engineered meta-learning system does scale and does consistently outperform a single learning algorithm.

Besides the three pre-training pruning algorithms, the thesis describes a new post-training pruning algorithm. The novelty of the algorithm is in the manner it transforms arbitrary meta-classifiers to take advantage of effective existing decision-tree pruning techniques. Our empirical evaluation demonstrated (with sweeping success) post-training pruning to be a robust approach for discarding classification models while preserving the overall predictive performance of the ensemble meta-classifier.

Meta-learning and pruning of meta-classifiers computed over distributed data sites assumes databases of identical schema. As we showed, however, this may not be always true. We examined and formulated the incompatible schema problem and detailed several techniques that help bridge the schema differences among databases and allow JAM and other data mining systems to share incompatible and otherwise useless classifiers. Our experimental study established the effectiveness of our methods and demonstrated that combining models from incompatible sources can substantially improve overall predictive performance.

We applied the JAM system on actual credit card transaction data sets provided by two separate financial institutions, where the task is to detect fraudulent transactions. The strategic advantage of JAM in this situation are: its flexibility to allow financial institutions to share their models of fraudulent transactions without disclosing their proprietary data and its ability to scale as the number and size of data bases grow. The results, in a nutshell, is that JAM, as a pattern-directed inference system coupled with meta-learning methods, constitutes a protective shield against fraud and is capable of computing meta-detectors that exhibite far superior fraud detection capabilities comparing to single model approaches and traditional authorization/detection systems.

The empirical evaluation of the system involved:

1. the training of multiple fraud detectors (classifier agents) and meta-detectors within each bank using various learning algorithms and meta-learning schemes,

2. the exchange of classifiers and the application of the bridging methods to facilitate the meta-learning of meta-detectors across the two institutions, and

3. the use of the various pruning techniques as part of meta-learning, and the experimental analysis of their impact on the performance of the final meta-classifiers.

The empirical evaluation of JAM and the search for the best fraud prediction model generated a significant number of base-level and meta-level models. Overall we experimented with 5 learning algorithms, 12 subsets of data for each of the 2 databases (provided by Chase and First Union), 8 meta-learning schemes, 3 bridging methods and 11 pruning techniques each applied multiple times with different pruning specifications (i.e., number of classifiers to discard). The experiments were repeated 6 times each and the results were averaged. In all cases, the classification models were compared against 3 different evaluation metrics, namely the accuracy, the TP-FP spread and a cost model.

The 5 learning algorithms included the Naive Bayesian learning algorithm, the C4.5, CART and ID3 decision tree methods, and Ripper, a rule-based learning tech-

Table 8.1: Performance results for the Chase credit card data set.

Type of Classification Model	Size	Accuracy	TP - FP	Savings
COTS scoring system from Chase	-	85.7%	0.523	$ 682K
Best base classifier over single subset	1	88.7%	0.557	$ 843K
Best base classifier over largest subset	1	88.5%	0.553	$ 812K
Meta-classifier over Chase base classifiers	50	89.74%	0.621	$ 818K
Meta-classifier over Chase base classifiers	46	**89.76%**	0.574	$ 604K
Meta-classifier over Chase base classifiers	27	88.93%	0.632	$ 832K
Meta-classifier over Chase base classifiers	4	88.89%	0.551	**$ 905K**
Meta-classifier over Chase and First Union base classifiers (without bridging)	110	89.7%	0.621	$ 797K
Meta-classifier over Chase and First Union base classifiers (without bridging)	65	89.75%	0.571	$ 621K
Meta-classifier over Chase and First Union base classifiers (without bridging)	43	88.34%	**0.633**	$ 810K
Meta-classifier over Chase and First Union base classifiers (without bridging)	52	87.71%	0.625	$ 877K

nique. For meta-learning, we experimented with 2 voting schemes (majority voting and weighted voting), 5 stacking methods (corresponding to the 5 learning algorithms) and SCANN while the 3 bridging methods refer to the interpolation, the linear regression fit and the regression tree algorithms that were used to resolve the incompatible schema problem. The pruning techniques that were implemented and tested include an *arbitrary* method that randomly selects base classifiers; three instances of a *metric*-based method that evaluates each candidate classifier independently;[1] a *diversity*-based algorithm that decides by examining the classifiers in correlation with each other; four instances of a *specialty/coverage* method[2] that rely on the independent performance of the classifiers and the manner in which they predict with respect to each other and with respect to the underlying data set; a *cost complexity*-based technique (a technique used by the CART decision tree learning algorithm that seeks to minimize the cost (size) of its tree while reducing the misclassification rate) and a *correlation*-based method that discard classifiers based on the correlation between the classifiers and the meta-classifier.

The end result of this extensive empirical evaluation is summarized in Tables 8.1 and 8.2. Table 8.1 reports the performance results of the best classification models on Chase data, while Table 8.2 presents the performance results of the best performers on the First Union data. Both tables display the accuracy, the TP-FP spread and savings for each of the fraud predictors examined and the best result in every category is depicted

[1]The three instances correspond to the evaluation metric adopted, namely the accuracy, the TP-FP spread and the cost model.

[2]The three instances correspond to the evaluation metric adopted, namely, the accuracy, the TP-FP spread and the cost model and the fourth corresponds to the PCS specialty metric.

Table 8.2: Performance results for the First Union credit card data set.

Type of Classification Model	Size	Accuracy	TP - FP	Savings
Best base classifier over single subset	1	95.2%	0.749	$ 800K
Best base classifier over largest subset	1	95.5%	0.790	$ 803K
Meta-classifier over First Union base classifiers	50	96.53%	0.831	$ 935K
Meta-classifier over First Union base classifiers	14	96.59%	0.797	$ 891K
Meta-classifier over First Union base classifiers	12	96.53%	0.848	$ 944K
Meta-classifier over First Union base classifiers	26	96.50%	0.838	$ 945K
Meta-classifier over Chase and First Union base classifiers (without bridging)	110	96.6%	0.843	$ 942K
Meta-classifier over Chase and First Union base classifiers (with bridging)	110	**98.05%**	0.897	**$ 963K**
Meta-classifier over Chase and First Union base classifiers (with bridging)	56	98.02%	0.890	$ 953K
Meta-classifier over Chase and First Union base classifiers (with bridging)	61	98.01%	**0.899**	$ 950K
Meta-classifier over Chase and First Union base classifiers (with bridging)	53	98.00%	0.894	$ 962K

in bold. The maximum achievable savings for the "ideal" classifier, with respect to our cost model, is $1,470K for the Chase and $1,085K for the First Union data sets. The column denoted as "size" indicates the number of base-classifiers used in the classification system.

The first row of Table 8.1 shows the best possible performance of Chase's own COTS authorization/detection system on this data set. The next two rows present the performance of the best base classifiers over a single subset and over the largest possible [3] data subset, while the next four rows detail the performance of the unpruned (size of 50) and best pruned meta-classifiers for each of the evaluation metrics (size of 46 for accuracy, 27 for the TP-FP spread, and 4 for the cost model). Finally, the last four rows report on the performance of the unpruned (size of 110) and best pruned meta-classifiers (sizes of 65, 43, 52) according to accuracy, the TP-FP spread and the cost model respectively. The first four meta-classifiers combine only "internal" (from Chase) base classifiers, while the last four combine both internal and external (from Chase and First Union) base classifiers. Bridging agents were not used in these experiments, since all attributes needed by First Union agents, were already defined in the Chase data.

Similar data is recorded in Table 8.2 for the First Union set, with the exception of First Union's COTS authorization/detection performance (it was not made available to us), and the additional results obtained when employing special bridging agents from Chase to compute the values of First Union's missing attributes.

[3] Determined by the available system resources.

The most apparent outcome of these experiments is the superior performance of meta-learning over the single model approaches and over the traditional authorization/detection systems (at least for the given data sets). The meta-classifiers outperformed the single base classifiers (local or global) in every category. Moreover, by bridging the two databases, we managed to further improve the performance of the meta-learning system. Notice, however, that combining classifiers agents from the two banks directly (without bridging) was not very effective. This phenomenon was explained from the fact that the attribute missing from the First Union data set is significant in modeling the Chase data set. Hence, the First Union classifiers were not as effective as the Chase classifiers on the Chase data, and the Chase classifiers could not perform at full strength at the First Union sites without the bridging agents.

An additional result, evident from these tables, is the invaluable contribution of pruning. In all cases, pruning succeeded in computing meta-classifiers with similar or better fraud detection capabilities, while reducing their size and thus improving their efficiency.

8.2 Limitations and Future Research Directions

While JAM is a stable and flexible system, it can be further enhanced with *additional functionality*. As we have already discussed, future extensions of the Configuration Manager can support the formation of groups of sites and provide authentication capabilities, directory services, fault tolerance, etc. JAM sites can be extended with tools facilitating the *data selection problem*. The data selection problem refers to the preprocessing, transformation and projection of the available data to expressive and informative features, and is probably one of the hardest, but very important stages in the knowledge discovery process. The process depends on the particular data mining task and requires application domain knowledge. The current version of JAM, assumes well-defined schemata and data sets. The credit card data sets that were used in this thesis, for example, were first developed by experienced FSTC (Financial Services Technology Consortium) personnel and then cleaned and pre-processed by us in a separate off-line process before being used in JAM.

Introducing data selection tools and defining the JAM databases can be linked to the incompatible schema problem. Recall that comparing databases and identifying attributes with syntactic or semantic differences has not been addressed in this thesis. The study and development of *formal methods and languages* for declaring and defining schemata is a crucial and hard problem, suitable for extensive research (early work in this field can be found in [Garcia-Molina *et al.*, 1995; Haas *et al.*, 1999]). Resolving the incompatible schema problem can instigate the expansion of present data mining systems. The "visibility" of meta-learning systems will be extended to data sources that would otherwise remain unutilized, information will be shared more readily and meta-level classification models will improve their performance by automatically incorporating more diverse models.

On the other hand, the number of the distributed data sources and the degree their data schemata is different can pose a theoretical limitation on the scalability of the bridging methods. To collaborate with N other sites that define a total of K_R different attributes, the local JAM site with K_L attributes may be forced to request and import $K_R \times 2^{K_L}$ bridging agents. This worst case scenario, however, assumes a large number N ($> 2^{K_L}$) which, for all practical purposes, is unrealistic. In addition, the need for combining multiple incompatible classifiers originates when these classifiers are trained for the same classification problem, a fact that limits the likelihood of highly differentiated data sets (e.g., in the credit card data sets it is expected that all banks would try to capture the transaction amount, the merchant code, the date and time of the transaction, etc.).

The JAM system can be the basis for studying many related problems that remain open for research. One such problem stems out of the fact that meta-classifiers are also classifiers that too can be meta-learned and recursively combined into higher level meta-classifiers structured in multi-level tree hierarchies. While meta-learning constitutes a generic, scalable and efficient approach to computing broader and more effective models, it further complicates the already hard problem of deciding which classifiers or meta-classifiers to combine.

For example, classifiers can be joined in many different ways with other classifiers or meta-classifiers. Similarly meta-classifiers can be combined with other meta-classifiers of similar or different heights, possibly computed over different time periods. The adverse effect of this approach is large and relatively slow final meta-classifiers. As we have already described, to classify unlabeled instances, predictions need to be generated from the leaves of the multi-level tree (base-classifiers), and propagated through the internal nodes (intermediate meta-classifiers) to the root of the tree (final meta-classifier). In this thesis, we examined pruning techniques for discarding redundant base classifiers based on their predictive performance as a means to meet classification throughput requirements of real-time systems. Our future plans include the extension of these methods to *hierarchical meta-learning* and the investigation of additional and more *elaborate pruning techniques*. These new techniques would need to examine the *throughput/predictive performance tradeoff* as well, but at the same time they would also need to consider the speed of each classifier as an additional factor. As more databases become available, we expect that the pruning of larger meta-classifier hierarchies will be of even greater importance.

An orthogonal approach to improving classification throughput is to extend the JAM system with the capability to generate predictions in a distributed and parallel fashion. By supporting *distributed meta-classifiers* (i.e., classifier trees with nodes that are not necessarily localized on a single machine), and by augmenting JAM to remotely control the parallel execution of classifier agents to obtain their predictions simultaneously, we can achieve significant classification speed-up.

Another area of interest that emerged during the study of the credit card fraud detection problem, is *cost based learning*. With only a few recent exceptions, accuracy based learning has been the most commonly used technique when computing predictive models. As we showed in the credit card fraud domain, however, overall accuracy is inappropriate as the single measure of predictive performance. Generally, in cases, where the cost of fraud is not equal to the cost of a false alarm, we should seek for alternative "optimality criteria" that rely on ROC analysis techniques or specific cost models. While we have already examined various cost-based methods for computing meta-classifiers that are more sensitive to expensive fraudulent transactions, unless we study this problem at an earlier stage, e.g., when training the "base-classifiers", we will not be able to solve the problem as well as possible. There are two directions to consider here, to develop generic cost-sensitive learning algorithms that can be easily tailored to specific learning tasks, and to apply existing accuracy based learning algorithms but to appropriately transformed data sets and thus compute cost based sensitive predictive models.

The main objective in this research was the development of a system and techniques that facilitate the discovery and sharing of information that is distributed across multiple sites in an intranet of internet. The scope of this thesis was limited to structured data sets. An analogous problem, over less structured data can be explored in the context of Information Retrieval (IR), and more specifically, in *Collection Fusion*, the area of collecting and combining the results from multiple IR systems, also known as meta-searching. The task here, is to generate a list of more relevant documents to a query than any of the individual IR systems. The approach parallels that of meta-learning. First launch a query to multiple IR systems to search different document databases in parallel and then, integrate in some principled fashion the compiled results in a single list of documents. However, significant differences in the setting of IR systems necessitate additional research: local search engines (the equivallent of the base classifiers) are not easily "exportable" and separate validation sets are not likely to be available. But as with meta-learning, the possibilities for meta-searching are numerous; from investigating simple techniques for combining results from existing IR systems, to augmenting the existing fusion methods with other modalities (deduced perhaps from additional document lists generated from similar queries) to even developing an end-to-end real-time system for launching agents in parallel to dynamically provide additional information for the retrieved documents under user-imposed time constraints.

Bibliography

[Agresti, 1990] Agresti, A. 1990. *Categorical Data Analysis.* J. Wiley and Sons Inc.

[Aha, Kibler, & Albert, 1991] Aha, D.; Kibler, D.; and Albert, M. 1991. Instance-based learning algorithms. *Machine learning* 6:37–66.

[Ali & Pazzani, 1996] Ali, K., and Pazzani, M. 1996. Error reduction through learning multiple descriptions. *Machine Learning* 24:173–202.

[Arnold & Gosling, 1998] Arnold, K., and Gosling, J. 1998. *The Java Programming Language, second edition.* Reading, MA: Addison-Wesley.

[Atkeson, Schaal, & Moore, 1999] Atkeson, C. G.; Schaal, S. A.; and Moore, A. W. 1999. Locally weighted learning. AI Review, In press.

[Belford, 1998] Belford, M. 1998. Information overload. In *Computer Shopper.*

[Bishop, Fienberg, & Holland, 1975] Bishop, Y.; Fienberg, S.; and Holland, P. 1975. *Discrete Multivariate analusis: Theory and Practice.* Cambirdge, MA: MIT Press.

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

[Breiman, 1994] Breiman, L. 1994. Heuristics of instability in model selection. Technical report, Department of Statistics, University of California at Berkeley.

[Breiman, 1996] Breiman, L. 1996. Stacked regressions. *Machine Learning* 24:41–48.

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

[Carson & Fischer, 1990] Carson, E. R., and Fischer, U. 1990. Models and computers in diabetes research and diabetes care. *Computer methods and programs in Biomedicine, Special Issue* 32.

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

[Catlett, 1992] Catlett, J. 1992. *Megainduction: machine learning on very large databases.* Ph.D. Dissertation, Dept. of Computer Sci., Univ. of Sydney, Sydney, Australia.

[C.Brodley, 1993] C.Brodley. 1993. Addressing the selective superiority problem: Automatic algorithm/model class selection. In *Proc. 10th Intl. Conf. Machine Learning*, 17–24. Morgan Kaufmann.

[Chan & Stolfo, 1993a] Chan, P., and Stolfo, S. 1993a. Meta-learning for multistrategy and parallel learning. In *Proc. Second Intl. Work. Multistrategy Learning*, 150–165.

[Chan & Stolfo, 1993b] Chan, P., and Stolfo, S. 1993b. Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Knowledge Discovery in Databases*, 227–240.

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

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

[Chan & Stolfo, 1998] Chan, P., and Stolfo, S. 1998. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proc. Fourth Intl. Conf. Knowledge Discovery and Data Mining*, 164–168.

[Chan *et al.*, 1999] Chan, P.; Fan, W.; Prodromidis, A.; and Stolfo, S. 1999. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems magazine on Data Mining.* In press.

[Chan, Stolfo, & Wolpert, 1996] Chan, P.; Stolfo, S.; and Wolpert, D., eds. 1996. *Working Notes for the AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms.*

[Chan, 1996] Chan, P. 1996. *An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning.* Ph.D. Dissertation, Department of Computer Science, Columbia University, New York, NY.

[Cheeseman *et al.*, 1988] Cheeseman, P.; Kelly, J.; Self, M.; Stutz, J.; Taylor, W.; and Freeman, D. 1988. Autoclass: A bayesian classification system. In *Proc. Fifth Intl. Conf. Machine Learning*, 54–64.

[Clark & Niblett, 1989] Clark, P., and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 3:261–285.

[Clearwater *et al.*, 1989] Clearwater, S. H.; Cheng, T. P.; Hirsh, H.; and Buchanan, B. G. 1989. Incremental batch learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, 366–370. San Mateo, CA: Morgan Kaufmann.

[Cohen, 1960] Cohen, J. 1960. A coefficient of agreement for nominal scales. *Educational And Psycological Meas.* 20:37–46.

[Cohen, 1995] Cohen, W. 1995. Fast effective rule induction. In *Proc. 12th Intl. Conf. Machine Learning*, 115–123. Morgan Kaufmann.

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

[Cover & Thomas, 1991] Cover, T., and Thomas, J. 1991. *Elements of Information Theory*. J. Wiley and Sons Inc.

[DeJong, Spears, & Gordon, 1993] DeJong, K. A.; Spears, W. M.; and Gordon, D. F. 1993. Using genetic algorithms for concept learning. *Machine Learning* 13:161–188.

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

[Detrano *et al.*, 1989] Detrano, R.; Janosi, A.; Steinbrunn, W.; Pfisterer, M.; Schmid, J.; Sandhu, S.; Guppy, K.; Lee, S.; and Froelicher, V. 1989. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology* 64:304–310.

[Dietterich, 1997] Dietterich, T. 1997. Machine learning research: Four current directions. *AI Magazine* 18(4):97–136.

[Dietterich, 1998] Dietterich, T. 1998. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation* 10(7):1895–1924.

[Domingos, 1996] Domingos, P. 1996. Efficient specific-to-general rule induction. In *Proceedings Second International Conference on Knowledge Discovery & Data Mining*, 319–322. Portland, OR: AAAI Press.

[Domingos, 1997] Domingos, P. 1997. Knowledge acquisition from examples via multiple models. In *Proc. Fourteenth Intl. Conf. Machine Learning*, 98–106.

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

[Everitt, 1977] Everitt, B. 1977. *The analysis of contigency tables.* London, UK: Chapman and Hall.

[Fawcett & Provost, 1997] Fawcett, T., and Provost, F. 1997. Adaptive fraud detection. *Data Mining and Knowledge Discovery* 1(3):291–316.

[Fayyad *et al.*, 1996] Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R. 1996. *Advances in Knowledge Discovery and Data Mining.* Menlo Park, California/Cambridge, Massachusetts/London, England: AAAI Press/MIT Press.

[Fayyad, Piatetsky-Shapiro, & Smyth, 1996] Fayyad, U.; Piatetsky-Shapiro, G.; and Smyth, P. 1996. The KDD process for extracting useful knowledge from volumes of data. *Communication of the ACM* 39(11):27–34.

[Freund & Schapire, 1995] Freund, Y., and Schapire, R. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, 23–37. Springer-Verlag.

[Freund & Schapire, 1996] Freund, Y., and Schapire, R. 1996. Experiments with a new boosting algorithm. In *Proc. Thirteenth Conf. Machine Learning*, 148–156.

[Friedman, 1991] Friedman, J. H. 1991. Multivariate adaptive regression splines. *The Annals of Statistics* 19(1):1–141.

[Garcia-Molina *et al.*, 1995] Garcia-Molina, H.; Hammer, J.; Ireland, K.; Papakonstantinou, Y.; Ullman, J.; and Widom, J. 1995. Integrating and accessing heterogeneous information sources in tsimis. In *Proc of the AAAI Symposium on Information Gathering*, 61–64.

[Greenacre, 1984] Greenacre, M. J. 1984. *Theory and Application of Correspondence Analysis.* London: Academic Press.

[Guo & Sutiwaraphun, 1998] Guo, Y., and Sutiwaraphun, J. 1998. Knowledge probing in distributed data mining. In H. Kargupta, P. C., ed., *Work. Notes KDD-98 Workshop on Distributed Data Mining*, 61–69. AAAI Press.

[Haas *et al.*, 1999] Haas, L. M.; Miller, R. J.; Niswonger, B.; Roth, M. T.; Schwarz, P. M.; and Wimmers, E. L. 1999. Transforming heterogeneous data with database middleware: Beyond integration. *Data Engineering Bulletin.*

[Holland, 1975] Holland, J. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

[Holland, 1986] Holland, J. 1986. Escaping brittleness: The possiblilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R.;

Carbonell, J.; and Mitchell, T., eds., *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*. Los Altos, CA: Morgan Kaufmann. 593–623.

[Hopfield, 1982] Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. In *Proc. of the National Academy of Sciences*, volume 79, 2554–2558.

[Jacobs *et al.*, 1991] Jacobs, R.; Jordan, M.; Nowlan, S. J.; and Hinton, G. E. 1991. Adaptive mixture of local experts. *Neural Computation* 3(1):79–87.

[Jordan & Jacobs, 1994] Jordan, M. I., and Jacobs, R. A. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation* 6:181–214.

[Jordan & Xu, 1993] Jordan, M. I., and Xu, L. 1993. Convergence results for the em approach to mixtures of experts architectures. In *AI memo 1458.*

[K.Lang, 1995] K.Lang. 1995. NEWS WEEDER: Learning to filter net news. In Prieditis, A., and S.Russel., eds., *Proc. 12th Intl. Conf. Machine Learning*, 331–339. Morgan Kaufmann.

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

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

[Kubat & Matwin, 1997] Kubat, M., and Matwin, S. 1997. Addressing the curse of imbalanced training sets: One-sided selection. In *Proc. 14th Intl. Conf. Machine Learning*, 179–186.

[Kwok & Carter, 1990] Kwok, S., and Carter, C. 1990. Multiple decision trees. In *Uncertainty in Artificial Intelligence 4*, 327–335.

[Lawrence & Giles, 1999] Lawrence, S., and Giles, C. L. 1999. Accessibility of information on the WEB. In *Nature*, volume 8, 107–109.

[LeBlanc & Tibshirani, 1993] LeBlanc, M., and Tibshirani, R. 1993. Combining estimates in regression and classification. Technical Report 9318, Department of Statistics, University of Toronto, Toronto, ON.

[Lee, Barghouti, & Moccenigo, 1997] Lee, W.; Barghouti, N. S.; and Moccenigo, J. 1997. Grappa: Graph package in java. In *Graph Drawing, Rome, Italy.*

[Lindholm & Yellin, 1999] Lindholm, T., and Yellin, F. 1999. *The Java Virtual Machine Specification, second edition.* Reading, MA: Addison-Wesley.

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

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

[Maitan, Ras, & Zemankova, 1989] Maitan, J.; Ras, Z. W.; and Zemankova, M. 1989. Query handling and learning in a distributed intelligent system. In Ras, Z. W., ed., *Methodologies for Intelligent Systems, 4*, 118–127. Charlotte, North Carolina: North Holland.

[Malliaris & Salchenberger, 1993] Malliaris, M., and Salchenberger, L. 1993. A neural network model for estimating option prices. *Applied Intelligence* 3(3):193–206.

[Margineantu & Dietterich, 1997] Margineantu, D., and Dietterich, T. 1997. Pruning adaptive boosting. In *Proc. Fourteenth Intl. Conf. Machine Learning*, 211–218.

[Mehta, Agrawal, & Rissanen, 1996] Mehta, M.; Agrawal, R.; and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. In *Proc. of the fifth Int'l Conf. on Extending Database Technology.*

[Merz & Murphy, 1996] Merz, C., and 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.

[Merz & Pazzani, 1999] Merz, C., and Pazzani, M. 1999. A principal components approach to combining regression estimates. *Machine Learning.* In press.

[Merz, 1999] Merz, C. 1999. Using correspondence analysis to combine classifiers. *Machine Learning.* In press.

[Michalski *et al.*, 1986] Michalski, R. S.; Mozetic, I.; Hong, J.; and Lavrac, N. 1986. The multipurpose incremental leaning system AQ51 and its testing application to three medical domains. In *Proc. AAAI-86*, 1041–1045.

[Michalski, 1983] Michalski, R. 1983. A theory and methodology of inductive learning. In Michalski, R.; Carbonell, J.; and Mitchell, T., eds., *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann. 83–134.

[Minksy & Papert, 1969] Minksy, M., and Papert, S. 1969. *Perceptrons: An Introduction to Computation Geometry.* Cambridge, MA: MIT Press. (Expanded edition, 1988).

[Mitchell, 1982] Mitchell, T. 1982. Generalization as search. *Artificial Intelligence* 18:203–226.

[Mitchell, 1997a] Mitchell, T. 1997a. *Machine Learning*. McGraw-Hill.

[Mitchell, 1997b] Mitchell, T. M. 1997b. Does machine learning really work? *AI Magazine* 18(3):11–20.

[Myers, 1986] Myers, R. 1986. *Classical and Modern Regression with Applications*. Boston, MA: Duxbury.

[Opitz & Shavlik, 1996] Opitz, D. W., and Shavlik, J. J. W. 1996. Generating accurate and diverse members of a neural-network ensemble. *Advances in Neural Information Processing Systems* 8:535–541.

[Ortega, Koppel, & Argamon-Engelson, 1999] Ortega, J.; Koppel, M.; and Argamon-Engelson, S. 1999. Arbitrating among competing classifiers using learned referees. *Machine Learning*. in press.

[Perrone & Cooper, 1993] Perrone, M. P., and Cooper, L. N. 1993. When networks disagree: Ensemble methods for hybrid neural networks. *Artificial Neural Networks for Speech and Vision* 126–142.

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

[Prodromidis & Stolfo, 1998a] Prodromidis, A. L., and Stolfo, S. J. 1998a. Mining databases with different schemas: Integrating incompatible classifiers. In R Agrawal, P. Stolorz, G. P.-S., ed., *Proc. 4th Intl. Conf. Knowledge Discovery and Data Mining*, 314–318. AAAI Press.

[Prodromidis & Stolfo, 1998b] Prodromidis, A. L., and Stolfo, S. J. 1998b. Pruning meta-classifiers in a distributed data mining system. In *Proc of the KDD'98 workshop in Distributed Data Mining*, 22–30.

[Prodromidis & Stolfo, 1998c] Prodromidis, A. L., and Stolfo, S. J. 1998c. Pruning meta-classifiers in a distributed data mining system. In *Proc of the First National Conference on New Information Technologies*, 151–160. Extended version.

[Prodromidis & Stolfo, 1999a] Prodromidis, A. L., and Stolfo, S. J. 1999a. Agent-based distributed learning applied to fraud detection. CUCS-014-99.

[Prodromidis & Stolfo, 1999b] Prodromidis, A. L., and Stolfo, S. J. 1999b. Cost complexity-based pruning of ensemble classifiers. Technical Report, CUCS-028-99.

[Prodromidis & Stolfo, 1999c] Prodromidis, A., and Stolfo, S. 1999c. A comparative evaluation of meta-learning strategies over large and distributed data sets. In *Workshop on Meta-learning, Sixteenth Intl. Conf. Machine Learning*, 18–27.

[Prodromidis & Stolfo, 1999d] Prodromidis, A., and Stolfo, S. 1999d. Minimal cost complexity pruning of meta-classifiers. In *Proc. Sixteenth National Conference on Artificial Intelligence (AAAI-99).*

[Prodromidis, Chan, & Stolfo, 1999] Prodromidis, A.; Chan, P.; and Stolfo, S. 1999. *Advances of Distributed Data Mining.* Menlo Park, California: AAAI Press. In press.

[Prodromidis, Stolfo, & Chan, 1999] Prodromidis, A. L.; Stolfo, S. J.; and Chan, P. K. 1999. Effective and efficient pruning of meta-classifiers in a distributed data mining system. Technical report, Columbia Univ. CUCS-017-99.

[Prodromidis, 1997] Prodromidis, A. L. 1997. On the management of distributed learning agents. Technical Report CUCS-032-97 (Ph.D. Thesis proposal), Department of Computer Science, Columbia University, New York, NY.

[Provost & Fawcett, 1997] Provost, F., and Fawcett, T. 1997. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proc. Third Intl. Conf. Knowledge Discovery and Data Mining*, 43–48.

[Provost & Fawcett, 1998] Provost, F., and Fawcett, T. 1998. Robust classification systems for imprecise environments. In *Proc. AAAI-98.* AAAI Press.

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

[Provost & Kolluri, 1997] Provost, F., and Kolluri, V. 1997. Scaling up inductive algorithms: An overview. In *Proc. Third Intl. Conf. Knowledge Discovery and Data Mining*, 239–242.

[Provost, Fawcett, & Kohavi, 1998] Provost, F.; Fawcett, T.; and Kohavi, R. 1998. The case against accuracy estimation for comparing induction algorithms. In *Proc. Fifteenth Intl. Conf. Machine Learning*, 445–553.

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

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

[R. & J., 1994] R., W. S., and J., R. A. 1994. Classification using hierarchical mixtures of experts. In *IEEE Workshop on Neural Networks for Signal Processing IV*, 177–186.

[Ras, 1998] Ras, Z. W. 1998. Answering non-standard queries in distributed knowledge-based systems. In A. Skowron, L. P., ed., *Rough sets in Knowledge Discovery, Studies in Fuzziness and Soft Computing*, volume 2, 98–108. Physica Verlag.

[Rumelhart & McClelland, 1986] Rumelhart, D., and McClelland, J. L. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. Cambridge, MA: MIT Press.

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

[Shafer, Agrawal, & Metha, 1996] Shafer, J. C.; Agrawal, R.; and Metha, M. 1996. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the 22nd Int'l Conf. on Very Large Databases*.

[Sta, 1996] StatSci Division, MathSoft, Seattle. 1996. *Splus, Version 3.4*.

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

[Stolfo *et al.*, 1997a] Stolfo, S.; Fan, W.; Lee, W.; Prodromidis, A.; and Chan, P. 1997a. Credit card fraud detection using meta-learning: Issues and initial results. In *Working notes of AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*.

[Stolfo *et al.*, 1997b] Stolfo, S.; Prodromidis, A.; Tselepis, S.; Lee, W.; Fan, W.; and Chan, P. 1997b. JAM: Java agents for meta-learning over distributed databases. In *Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, 74–81.

[Stolfo *et al.*, 1998] Stolfo, S.; Fan, W.; W.Lee, A. P.; Tselepis, S.; and Chan, P. K. 1998. Agent-based fraud and intrusion detection in financial information systems. Available from [http://www.cs.columbia.edu/∼sal/JAM/PROJECT].

[T. Oates, 1998] T. Oates, D. J. 1998. Large datasets lead to overly complex models: As explanation and a solution. In R. Agrawal, P. Stolorz, G. P.-S., ed., *Proc. Fourth Intl. Conf. Knowledge Discovery and Data Mining*, 294–298. AAAI Press.

[Tresp & Taniguchi, 1995] Tresp, V., and Taniguchi, M. 1995. Combining estimators using non-constant weighting functions. *Advances in Neural Information Processing Systems* 7:419–426.

[Turney, 1995] Turney, P. D. 1995. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of AI Research* 2:369–409.

[Utgoff, 1988] Utgoff, P. 1988. ID5: An incremental ID3. In *Proc. 5th Intl. Conf. Mach. Learning*, 107–120. Morgan Kaufmann.

[Utgoff, 1989] Utgoff, P. 1989. Incremental induction of decision trees. *Machine Learning* 4:161–186.

[Utgoff, 1994] Utgoff, P. 1994. An improved algorithm for incremental induction of decision trees. In *Proc. of the Eleventh Intl. Conference on Machine Learning*, 318–325.

[W. Lee, 1998] W. Lee, S. Stolfo, K. M. 1998. Mining audit data to build intrusion models. In R Agrawal, P. Stolorz, G. P.-S., ed., *Proc. Fourth Intl. Conf. Knowledge Discovery and Data Mining*, 66–72. AAAI Press.

[Way & Smith, 1991] Way, J., and Smith, E. A. 1991. The evolution of synthetic aperture radar systems and their progression to the EOS SAR. *IEEE Transactions on Geoscience and Remote Sensing* 29(6):962–985.

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

[Wu & Lo, 1998] Wu, X., and Lo, H. W. 1998. Multi-layer incremental induction. In *Proceedings of the fifth Pacific Rim International Conference on Artificial Intelligence.*