On the Management of Distributed Learning Agents

Ph.D. Thesis Proposal

CUCS-032-97

Andreas L. Prodromidis
andreas@cs.columbia.edu

Department of Computer Science
Columbia University
New York, NY 10027

January 26, 1998

Contents

List of Figures

Abstract

This thesis research concentrates on the problem of managing a distributed collection of intelligent learning agents across large and distributed databases. The main challenge is to identify and address the issues related to the efficiency, scalability, adaptivity and compatibility of these agents and the design and implemention of a complete and coherent distributed meta-learning system for large scale data mining applications. The resulting system should be able to scale with many large databases and make effective use of the available system resources. Furthemore, it should be capable to adapt to changes in its computational environment and be flexible enough to circumvent variances in database schema definitions. In this thesis proposal we present the architecture of *JAM* (Java Agents for Meta-learning), a distributed data mining system, and we describe in detail several methods to cope with the issues of scalability, efficiency, adaptivity and compatibility. Through experiments, performed on actual credit card and other public domain data sets, we evaluate the effectiveness and performance of our approaches and we demonstrate their potential.

1 Introduction

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* [30], a computer program is said to **learn** with respect to a class of tasks if its performance on these tasks, as measured by some performance measure, improves with its experience and interactions with its environment.

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 labelled 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.

Machine learning constitutes a significant part in the overall *Knowledge Discovery in Databases* (*KDD*) process, the process of extracting useful knowledge from large databases. Plain data is neither knowledge nor information; information refers to the results of processing data with respect to a particular problem, question or objective, whereas knowledge reflects the answers to these problems or questions. The benefits in maintaining volumes of data depend on the degree the stored data can be analyzed and exploited. One means of analyzing and mining useful information from large databases is to apply various machine learning algorithms to discover patterns exhibited in the data and compute descriptive representations (also called concepts or models). The field of machine learning has made substantial progress over the last few years 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 miscellaneous problems in diverse fields. One of the main challenges in knowledge discovery and data mining communities is the development of inductive learning techniques that **scale up** to large and may be physically distributed data sets. The number and size of 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 physically dispersed mobile platforms like an armada of ships, security and fault tolerant distribution of data and services, competitive (business) reasons, as well as statutory constraints imposed by law. In such situations, it may not be possible, nor feasible, to inspect all of the data at one processing site to compute one primary "global" concept or model. We call the problem of learning useful new information from large and inherently distributed databases, the *scaling problem for machine learning*.

Meta-learning is a technique developed recently that deals with the scaling problem. Meta-learning aims to compute a number of independent classifiers (concepts) by applying learning programs to a collection of independent and inherently distributed databases in parallel. The "base classifiers" computed are then integrated 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 question addressed in this thesis is: "Can we build a **scalable**, **efficient** and **adaptive distributed meta-learning** system that is generic and flexible enough to accommodate even **incompatible** yet **comparable** databases (i.e. similar databases but of different schemas)?

The proposed meta-learning system is the JAM system (Java Agents for Meta-learning), a powerful and portable agent based system for large scale data mining applications. Efficiency, scalability, adaptivity and compatibility are the main focus of this research. Efficiency and scalability are addressed first by introducing distributed management protocols and second by evaluating and deploying only the most essential classifiers and thus deterring complex and sizeable meta-learning hierarchies. Adaptivity is achieved by re-applying the meta-learning principles to update the derived concepts as patterns evolve over time, while compatibility is attained by several techniques that allow the system to cope with *incompatible* classifiers, i.e. classifiers that are trained from databases with similar but not identical schemas. By alleviating the differences the system can combine classifiers with somewhat different view of the classification problem.

The remainder of this thesis proposal is organized as follows: Section 2 gives an overview of data mining, inductive learning and meta-learning. Section 3 outlines our proposed research and examines the related work. In section 4 we discuss the problems and issues of distributed data mining systems, we describe the architecture of the JAM system and we detail our methods for evaluating and managing distributed learning agents. Section 5 presents the implementation aspects of JAM and the results of our first experiments. The current status of our research along with our future plans are reported in section 6. Finally, Section 7 concludes our proposal with a summary of the proposed research.

2 Data Mining and Meta-Learning

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

Over the past decade, machine learning has evolved from a field of laboratory demonstrations to a field of significant commercial value [31]. Machine-learning algorithms have been deployed in heart disease diagnosis [39], in predicting glucose levels for diabetic patients [17], in detecting credit card fraud [41], in steering vehicles driving autonomously on public highways at 70 miles an hour [33], in predicting stock option pricing [32], in computing customizing electronic newspapers[21] etc. Many large business institutions and market analysis firms attempt to distinguish the low-risk (high profit) potential customers by learn simple categorical classifications of their potential customer data base. Similarly, defense and intelligence operations utilize similar methodologies on vast information sources to predict a wide range of conditions in various contexts. Many organizations seeking similar added value from their data are already dealing with overwhelming amounts of global information that in time will likely grow in size faster than available improvements in machine resources.

Inductive learning (or *learning from examples* [28]) 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 $y = f(\mathbf{x})$, each interpreted as a set of attribute (feature) vectors $\mathbf{x}$ of the form $\{x_1, x_2, ..., x_m\}$ and a class label y associated with each vector, 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* or *class* y.

Some of the common representations used for the generated classifiers are decision trees, rules, version spaces, neural networks, distance functions, and probability distributions. In general, these representations are associated with different types of algorithms that extract different types of information from the database and provide alternative capabilities besides the common ability to classify unknown exemplars drawn from some domain. For example, decision trees are declarative and thus more comprehensible to humans than weights computed within a neural network architecture. However, both are able to compute concept y and classify unknown records (examples). Decision trees are used in ID3 [37], where each concept is represented as a conjunction of terms on a path from the root of a tree to a leaf. Rules in CN2 [12] are if-then expressions, where the antecedent is a pattern expression and the consequent is a class label. Each version space learned in VS [29] defines the most general and specific description boundaries of a concept using a restricted version of first order formulae. Neural networks compute separating hyperplanes in n-dimensional feature space to classify data [25]. The learned distance functions in exemplar-based learning algorithms (or nearest neighbor algorithms) define a similarity or "closeness" measure between two instances [40]. Conditional probability distributions used by Bayesian classifiers are derived from the frequency distributions of attribute values and reflect the likelihood of a certain instance belonging to a particular classification [11]. Implicit decision rules classify according to maximal probabilities.

Meta-learning [8] is itself a learning process aiming to improve accuracy and efficiency. Loosely defined, meta-learning is about learning from learned knowledge. The idea of this approach is to execute a number of concept learning processes on a number of data subsets in parallel, and combine their collective results through an extra level of learning. Initially, each concept learning task, also called *base learner*, computes a concept or *base classifier* that models its underlying data subset or *training set*. Next, a separate concept learning task, also called *meta learner* combines these independently built base classifiers into a higher level concept or classifier, called *meta classifier*, by learning from a *meta-level training set*. This meta-level training set is basically composed from the predictions of the individual base-classifiers when tested against a separate subset of the training data, also called *validation set*. From their predictions, the meta-learner will presumably detect the properties, the behavior and performance of the base-classifiers and compute a meta-classifier that represents a model of the "global" data set.

Meta-learning improves efficiency by executing *in parallel* the base-learning processes (each implemented as a distinct serial program) on (possibly disjoint) subsets of the training data set (*a data reduction technique*). This approach has the advantage, first, of using the same serial code without the time-consuming process of parallelizing it, and second, of learning from small subsets of data that fit in main memory.

Meta-learning improves accuracy by combining different learning systems each having different *inductive bias* (e.g representation, search heuristics, search space) [29]. Furthermore, by combining separately learned concepts, meta-learning is expected to derive a higher level learned model that explains a large database more accurately than any of the individual learners.

Meta-Learning 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.

There are several strategies and variations of meta-learning that depend on the way the meta-level training set is formed and the way the final prediction of the meta-classifier is synthesized (i.e. *class-combiner, class-attribute-combiner, different-arbiter, different-incorrect-arbiter, etc* to name

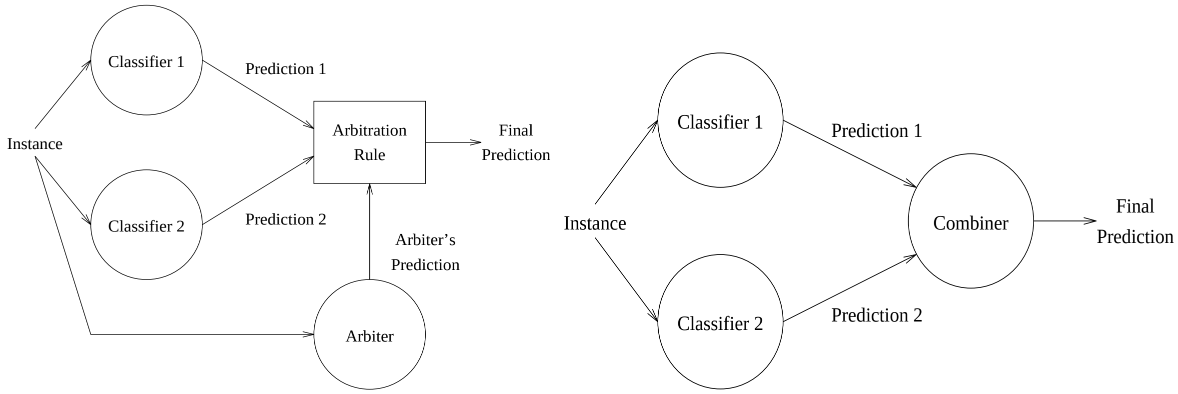

Figure 1: Left: An arbiter with two classifiers. Right: A combiner with two classifiers

Class	Attribute vector	Example	Base classifiers' predictions		
$class(x)$	$attrvec(x)$	x	$C_1(x)$	$C_2(x)$	$C_3(x)$
`mammal`	$attrvec_1$	x_1	`mammal`	`mammal`	`mammal`
`fish`	$attrvec_2$	x_2	`fish`	`mammal`	`fish`
`bird`	$attrvec_3$	x_3	`bird`	`mammal`	`mammal`

Training set for the *class-combiner* scheme	
Class	Attribute vector
`mammal`	(`mammal`, `mammal`, `mammal`)
`fish`	(`fish`, `mammal`, `fish`)
`bird`	(`bird`, `mammal`, `mammal`)

Figure 2: Sample training sets generated by the *class-combiner* strategy

a few). These strategies and variations are described in more detail in [7]. Briefly, an *arbiter* [9] is the result of a learning algorithm that learns to arbitrate among predictions generated by different base classifiers. This arbiter, together with an *arbitration rule*, decides a final classification outcome based upon the base predictions. The left diagram of Figure 1, depicts how the final prediction is made with input predictions from two base classifiers and a single arbiter.

In the *combiner* [7] strategy, the predictions of the learned base classifiers on the validation set form the basis of the meta-learner's training set. A *composition rule*, which varies in different schemes, determines the content of the meta-level training examples for the meta-learner. From these examples, the meta-learner generates a meta-classifier, that we call a *combiner*. In classifying an instance, the base classifiers first generate their predictions. Based on the same composition rule, a new meta-level instance is generated from the predictions, which is then classified by the combiner (see right diagram of Figure 1). The aim of this strategy is to "correlate" the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction. A combiner computes a prediction that may be entirely different from any proposed by a base classifier, whereas an arbiter chooses one of the predictions from the base classifiers and the arbiter itself.

In the *class-combiner* strategy, for example, the combiner seeks to learn the correlations of predictions among a number of base classifiers. For this purpose, a separate data set, called the validation set, is applied to every base classifier. The predictions by the base classifiers along with the true label of the input data item reveals the correlation of predictions among the base classifiers. The base classifier predictions as well as the correct class label are used to train the meta-level classifier. A particular example is displayed in Figure 2. In the *class-attribute-combiner* strategy, on the other hand, the combiner is trained over the base classifier predictions, the correct class label and the attribute vector, hence it seeks to learn the correlations of both predictions and attribute vectors.

3 Overview of Proposed Research

Knowledge discovery in databases (KDD) is an emerging field that spans across several areas such as databases, data mining, machine learning, statistics, data visualization, summarization, distributed systems and high performance computing and refers to the overall process of discovering useful knowledge from data [18]. Database theories and tools provide the necessary infrastructure to store, access and manipulate data; machine learning and statistics are concerned with inferring models from data and quantifying the results; summarization and data visualization examine methods to summarize, visualize and interpret the information gathered, while distributed and high performance computing deal with the scalability of the distributed systems, the protocols employed between the data sites and the efficiency and scalability of algorithms in the context of massive databases.

KDD is a very broad subject that cannot be exhaustively covered in this thesis research. Instead, we will 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. In this process, we will face many issues associated with the management and use of different models inferred by various machine learning agents when applied to different databases. Among them, we will address several problems related to the scalability of the protocols and the efficient and effective integration of these models.

A distributed data-mining system can be considered effective and useful if it is extensible, portable, scalable, efficient, adaptive, compatible with similar but not identical databases and last but not least accurate. The objective of this work is to investigate systematically the issues behind each of these characteristics and combine the results into JAM (Java Agents for Meta-learning), a complete and coherent distributed meta-learning system that builds upon existing agent technology available over the internet today. Specifically, our work addresses the following questions:

1. Assuming that JAM consists of several databases interconnected through an intranet or internet, can we design and implement a distributed data mining system where the data sites take advantage of their own local information and the information that may be available at the other data sites?

2. Being a distributed data mining system, JAM may be called to operate across different environments (e.g. over the internet). Moreover, since JAM relies on existing learning technology, it should be flexible to incorporate any past, present or future learning algorithm that is made available. Can JAM be made portable and extensible?

3. The benefits of storing volumes of data depend on our ability to extract and exploit useful information. In the context of machine learning and distributed data mining, this translates to our ability to compute, manage and deploy models derived from large databases. Given that the size and number of data bases is increasing, can we spot the bottlenecks and devise methods to elevate the efficiency and scalability of our system without sacrificing accuracy?

4. With more data collected and deployed, traditional learning and classification systems turn obsolete during their lifetime. It is desirable to introduce adaptive classification systems able to extract "fresh" information, discern new trends and enhance old models. We need a strategy in which the learning system will be able to extend and incorporate the new information without discarding or depreciating the knowledge it has accumulated. Can we equip our classification models with the capability to evolve and adapt to the changes of their environment?

5. Integrating classification models derived from distinct databases may not always be feasible. Minor differences in the schemas between databases instigate incompatible classifiers, yet these classifiers target the same concept. Can we devise methods that "bridge" their differences?

Before we discuss the details of our approach, we first summarize the existing techniques and current research in scaling-up machine learning tasks, in combining different models and in evaluating classifiers.

Scaling up The main challenge in data mining and machine learning is to deal with large problems in "reasonable" amount of time. So far, no research group has attempted to take advantage of the benefits of meta-learning to develop a distributed data mining system. On the other hand, the literature is quite rich of methods that facilitate the use of inductive learning algorithms for mining very large databases. In fact, Provost and Kolluri [35] have conducted a survey on the available methods and have categorized them into tree main groups; the methods that rely on the design of fast algorithms, the methods that reduce the problem size by partitioning the data and the 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. Part of this work will be, first, to design the architecture of a distributed system that would not hinder the scalability capabilities of meta learning and second, to maintain accuracy while repelling the meta classifier hierarchy from growing uncontrollably.

Evaluation and comparison metrics Several researchers from the Machine Learning and KDD communities have studied and defined metrics for the evaluation of ensembles (groups) of classifiers. Kwok and Carter [23] showed that ensembles with decision trees that were more syntactically diverse achieved lower error rates than ensembles consisting of less diverse decision trees. On the same subject Ali and Pazzani [2] suggested that when the average number of gain ties[1] is large, the syntactic diversity of the ensemble is greater which may lead to less correlated errors among the classifiers and hence lower error rates. However, they also cautioned that syntactical 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. Brodley and

[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 clause body of a rule) would result in a tree (rule) that would separate into the different classes as many as examples possible.

Lane [4] defined as *coverage* the fraction of instances for which at least one of the base classifiers produces the correct prediction. As they illustrate, 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. Last but not least, Chan introduced an additional metric, the *specialty* metric, to be equal to one minus the average normalized entropy over C classifiers:

$$specialty = 1 - \frac{1}{C}\sum_{j}^{C}\frac{1}{\log c}\sum_{k}^{c} -p_{jk}\log(p_{jk}) \tag{1}$$

where c represents the number of classes and p_{jk} denotes the normalized accuracy of the j^{th} base-classifier on the k^{th} class. In essence, the larger the value of specialty the more specialized the base-classifiers are to certain classes. He also defined *diversity* to be:

$$diversity = \frac{1}{n}\sum_{i}^{n}\frac{1}{\log c}\sum_{k}^{c} -p_{ik}\log(p_{ik}) \tag{2}$$

where n represents the number of instances examined and p_{ik} denotes the fraction of the base-classifiers predicting the k^{th} class for the i^{th} instance. According to this definition, when the value of diversity grows, the predictions from the base-classifiers are more evenly distributed (higher entropy) and, therefore, more diverse. In this study [6], Chan examined several characteristics of the base classifiers (i.e. diversity, coverage, correlated error and specialty) and explored the effects of these characteristics on the accuracy of the various integrating meta learning schemes. The results strengthened the belief that larger accuracy improvement can actually be achieved by employing more diverse base classifiers with higher coverage and fewer correlated errors.

Selecting models Margineantu and Dietterich [26] study the problem of selecting a subset of the hypothesis (classifiers) obtained by the boosting algorithm ADABOOST [19]. In essense, boosting learns a set of classifiers where each classifier concentrates on the examples of the training set misclassified by its predecessors. The algorithm draws examples for each classifier of the list 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 output of each classifier of the list according to its accuracy on the training set.

In their paper, Margineantu and Dietterich 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*, and they show that it is possible to obtain nearly the same level of performance with a subset of the classifiers as with the entire set. Very briefly, early stopping referred to the blind approach of keeping the first M classifiers obtained, whereas KL-divergence pruning focused on detecting the most diverse classifier by examining the probability distributions of their training sets. Both methods, however, fail to produce results of practical use. The Kappa-Error Convex Hull Pruning method, on the other hand, was more promising, but it was restricted to select a fixed number of classifiers. It chose its classifiers from the accuracy-diversity convex hall of the available classifiers. Overall the best pruning methods found were Kappa Pruning and Reduce-Error Pruning with Backfitting. The former relied on discovering the most diverse classifiers by inspecting their predictions on the training set, while the later took a more direct approach and selected the subset of the 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 selecting classifiers derived by the same learning program when trained on different subsets of the same training set. In this thesis proposal, we are considering these two dimensions as well. We study the more general setting where classifiers can be obtained by training (possibly) different learning algorithms over (possibly) distinct databases. Furthermore, instead of voting over the predictions of classifiers for the final classification, we adopt meta-learning to discover the importance of the individual classifiers.

Provost and Fawcett in [34] introduce 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 flexible in the sense that it allows classifier comparison under different metrics (e.g. accuracy, true positive/false negative rates, error cost, etc). Furthermore, by identifying the classifiers that are potentially optimal, the ROC convex hull method minimizes the management of classifier performance data. On the other hand, this method cannot deal with multiple class problems (it is geared towards two-class problems) and provides limited information about the interdependencies among the base classifiers when combined into higher level meta-classifiers.

4 Management of Distributed Learning Agents

In this chapter we describe the details of our strategies addressing the questions and obstacles associated with the design and implementation of distributed data mining systems.

JAM is a distributed agent based data mining system that provides a set of learning programs, implemented either as JAVA applets or applications, that compute models (concepts) 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. Furthermore, it employs a special distribution mechanism which allows the migration of the derived models or *classifier agents* to other remote sites.

4.1 Efficiency and Scalability

Distributed systems have an additional level of complexity comparing to independent stand-alone systems. With respect to data mining, this translates to the need to deal with possibly heterogenous platforms, with several databases with (possibly) different schemas, with the design and implementation of scalable and effective protocols and with the selective and efficient use of the information gathered from the peer data sites. The scalability of a data mining system depends on the protocols that allow the collaboration and transfer of information among the data sites while efficiency relies on the appropriate evaluation and filtration of the total available information to minimize redundant use of system resources.

4.1.1 Problem Description

One of the most intricate problems for the design of a distributed data mining system is to combine scalability and efficiency without sacrificing accuracy performance and vice versa. To understand the issues and tackle the complexity of the problem, we examine scalability and efficiency at two levels, the system architecture level and the meta-learning level.

In the **first** level, we focus on the building components of the system and the overall architecture. Assuming that the data mining system comprises of several data sites, each with its own resources, databases, machine learning agents and meta-learning capabilities, the problem is to design the protocol that would allow the data sites to collaborate efficiently without hindering their progress.

Asynchronous protocols avoid the overheads of synchronization points introduced by synchronous protocols, on the other hand, synchronous protocols are simpler and easier to design and implement.

Furthermore, it would be desirable to allow the data sites to be able to operate both independently or in collaboration with other peer data sites as necessary and also autonomously, i.e. without depending or being controlled by other "manager" sites. The protocol in a data mining system should be dynamic, in the sense that it should support dynamic reconfiguration of the system architecture (in case more data sites become available) and scalable, in the sense that it should be efficient even when data sites participate in large numbers. Distributed protocols avoid the bottlenecks and limitations of centralized approaches at the expense of being more complicated to implement and potentially more cumbersome.

In the **second** level, we delve inside the data sites to explore the types, the characteristics and properties of the classifiers. Employing efficient distributed protocols addresses the scalability problem only partially. The analysis of the dependencies and parameters of the classifiers and the management of the agents within the data sites, constitutes the other half of the scalability problem. If not controlled properly, the size and arrangement of the classifiers inside each data site may incur unnecessary and prohibitive overheads.

Meta classifiers are defined recursively as collections of classifiers structured in multi-level trees. Being hierarchically structured, meta classifiers are known, in general, to promote scalability and efficiency in a simple and straight forward manner. It is possible, however, that certain conditions yield the opposite effects. Brute force meta-learning techniques can result in bulk, expensive, inefficient and some times inaccurate meta-classifiers. On the other hand, the construction of modest, well managed and effective structures is not trivial. Lower level classifiers (base or meta-classifiers) may be combined in many different ways even when they are in small numbers originating from only a few different data sites.

Assume, for example, the common case, in which 10 data sites with disjoined data sets participate in a meta-learning system where all data sites can collaborate with each other. If we also assume that each data site is equipped with 2 learning agents (local or remote) and 2 meta-learning agents (not necessarily the same) then each data site can contribute up to 2 base classifiers to the common pool of base classifiers. If each data site imports all 20 base classifiers, then it will have to choose to build one level-1 local meta classifier out of 2,097,112 possible $(2 \times (2^{20} - 20))$ combinations! The different permutations and combinations of the lower level classifiers exhibit very different behaviors and accuracy results. Part of this research will be to study and identify the properties, characteristics and relations among the classifiers in order to select the most appropriate meta-classifier. From the example, it is evident that if we change the parameters and increase the number of participating data sites (very reasonable with large databases), the number of learning agents (a plethora of machine learning algorithms is already available) and construct higher (> 1) level meta-classifiers, the selection and combination of the right components to build meta-classifier trees is a hard problem by itself.

To make things worse, meta-classifier trees can be re-built, and grow at any point in their lifetime. As we will see in section 4.2, evolving classifiers can incorporate new components even after they are put in use. As new information comes in, trees can grow both in breadth and depth. It is therefore apparent that we need strategies that would reasonably bound the size of the meta-classifier trees and at the same time improve, if possible, their accuracy. The techniques for selecting the building components for a meta-classifier tree implicitly provide a way to contain the size of the tree. These methods however, provide only a partial solution. We also need to monitor the performance of the meta-classifier after it is built and while in use in order to verify that the chosen meta-classifier performs according to expectations. Corrective steps (e.g. discarding a lower level classifier that has negative influence on the meta-classifier or choose a different meta-classifier altogether) should be taken otherwise. The first level of classifier selection (choosing the appropriate combination of classifiers) is called *pre-training pruning*, while the second level (dropping classifier components or re-training poor classifiers) is called *post-training pruning*. Both levels are essential and complementary with respect to the improvement of accuracy and the containment of complexity (size) and hence scalability.

4.1.2 Approach

In this section we describe in detail the proposed system architecture of JAM and then we present the evaluation metrics and methods used for the *pre-training* and *post-training* pruning steps of the meta-learning phase.

4.1.2.1 System Architecture

The JAM system is designed around the idea of meta-learning hence it can take full advantage of its inherent parallelism and distributed nature. Recall that meta-learning improves efficiency by executing *in parallel* the same or different serial learning processes over different subsets of the training data set. Assuming that all database sites are capable of meta-learning, the JAM system can be viewed as a coarse grain parallel application where most of the times every database site functions autonomously and occasionally exchanges classifiers with the other database sites. Our plan is to equip JAM with fully distributed and asynchronous protocols that would enable the participating database sites to operate and progress independently or collaborate with other peer database sites as necessary thus eliminating the need for centralized control and synchronization points.

JAM is architected as an agent based system, a distributed computing construct that is designed as an extension of OS environments. It is a distributed meta-learning system that supports the launching of learning and meta-learning agents to distributed database sites. JAM is implemented as a collection of distributed learning and classification programs linked together through a network of *Datasites*. Each JAM Datasite consists of:

- A local database(s),
- One or more learning agents, or in other words machine learning programs that may migrate to other sites as JAVA applets, or be locally stored as native applications callable by JAVA applets,
- One or more meta-learning agents,
- A local user configuration file,
- Graphical User Interface and Animation facilities.

The JAM Datasites have been designed to collaborate[2] with each other to exchange classifier agents that are computed by the learning agents.

First, *local learning agents* operate on the *local database* and compute the Datasite's local classifiers. Each Datasite may then import (remote) classifiers from its peer Datasites and combine these with its own local classifier using the *local meta-learning agent*. Finally, once the base and meta-classifiers are computed, the JAM system manages the execution of these modules to classify and label data sets of interest. These actions may take place at all Datasites simultaneously and independently.

[2]A Datasite may also operate independently without any changes.

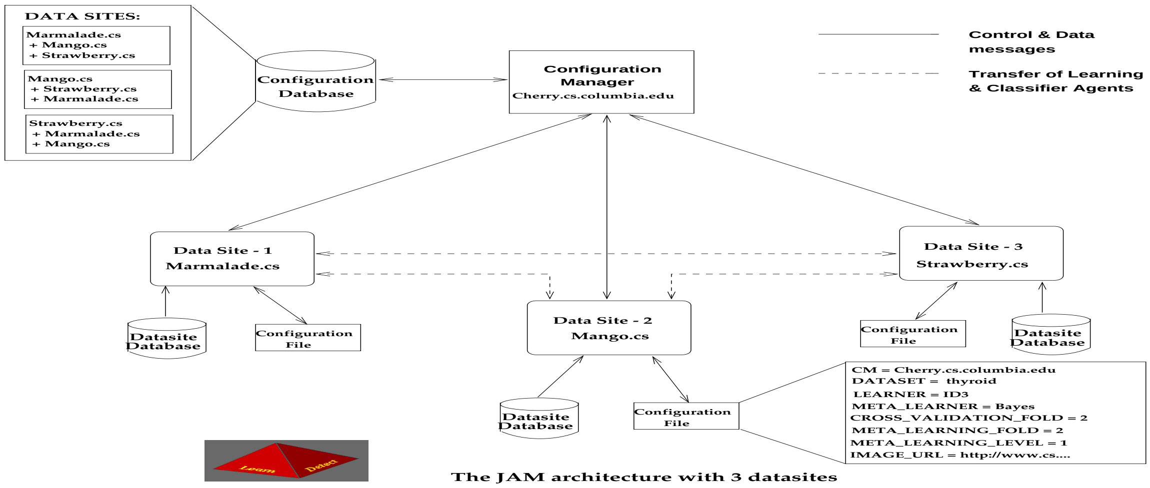

Figure 3: The architecture of the meta-learning system.

The owner of a Datasite 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 [3] specification of the local parameters, the owner of the Datasite can also employ *JAM*'s *graphical user interface* and *animation facilities* to supervise agent exchanges and administer dynamically the meta-learning process. With this graphical interface, the owner may access more information such as accuracy, trends, statistics and logs and compare and analyze results in order to improve performance.

The configuration of the distributed system is maintained by the Configuration Manager (CM), a central and independent module responsible for keeping the state of the system up-to-date. The CM is a server that provides information about the participating Datasites and logs events for future reference and evaluation. The logical architecture of the *JAM* meta-learning system is presented in Figure 3. In this example, three *JAM* Datasites Marmalade, Mango and Strawberry exchange their base classifiers to share their local view of the learning task. The owner of the Datasite 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 folding parameters, etc. In this example, the CM runs on Cherry and each Datasite ends up with three base classifiers (one local plus the two imported classifiers).

4.1.2.2 Pre-training pruning of the meta-classifier

Pre-training pruning refers to the filtering of the classifiers before they are used in the training of a meta-classifier. Recall that a meta-classifier is built on top of other existing classifiers. Instead of treating classifiers as sealed entities combined in a brute force manner, with pre-training pruning we introduce a pre-meta-learning stage for analyzing the available classifiers and qualifying them inclusion in a meta-classifier. Only those classifiers that appear (according to one or more pre-defined metrics) most "promising" will participate. The goal of pre-training pruning is to build partially grown meta-classifiers (meta-classifiers with pruned subtrees) that are more efficient and scalable and at the same time achieve comparable or better performance (accuracy) results than

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

fully grown meta-classifiers.

In order to take advantage of the benefits of pre-training pruning we need to address the following questions:

- Given a classifier, which properties can provide valuable information for selecting that classifier?
- How do we compare classifiers?
- What qualifies a combination of classifiers as "good"?

To analyze and compare the classifiers, we will employ several different algorithms. Some of the algorithms are based on the evaluation metrics that were presented earlier in section 3, while the rest are based on new metrics introduced here.

Class specialty: The term *class specialty* defines a family of evaluation metrics that concentrate on the "bias" of a classifier towards certain classes. However, in this study, instead of calculating the combined specialty of the resulting meta-classifiers [6], the *class specialty* metrics focus on the specialty of each (base-) classifier for each class. 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 the wrong class. More formally, assuming that:

- c_k = k-th class,
- c = the number of classes,
- $C_j(y_i)$ = classification of the y_i instance by the C_j classifier,
- $L(y_i)$ = the correct classification (label) of instance y_i
- C = the number of available classifiers,

Given a classifier C_j and a data set containing n examples, we can construct a two dimensional contingency table where each cell T_{kl} contains the number of examples x for which $L(x) = c_k$ and $C_j(x) = c_l$. According to this definition, cell T_{kk} contains the number of examples classifier C_j classifies correctly as c_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. Naturally, the sum of all the cells T_{kl} adds up to n. Then, True Positive and False Positive rates are defined as:

$$TP(C_j, c_k) = \frac{T_{kk}}{\sum_{i=1}^{c} T_{ki}} \tag{3}$$

$$FP(C_j, c_k) = \frac{\sum_{i \neq k} T_{ik}}{\sum_{i \neq k} \sum_{j=1}^{c} T_{ij}} \tag{4}$$

The *class specialty* metric is an attempt to quantify the bias of a classifier towards a certain class. In particular, a classifier C_j is highly biased/specialized for class c_k when its $TP(C_j, c_k)$ is high and its $FP(C_j, c_k)$ is low.

One-sided class specialty metric: Given the definitions of the TP and FP rates, this is the simplest and most straight forward metric of the class specialty family. The one-sided class specialty metric evaluates a classifier by inspecting the TP rate or the FP rate (but not both) of the classifier on a given class over the validation data set. Then, based on the results, we select the classifiers to be used in the next level of the meta-classifier. The proposed algorithm is simple:

1. For each class c_k, $1 \leq k \leq c$, evaluate the *one-sided class specialty* of each (base-)classifier C_j, $1 \leq j \leq C$, by calculating the $TP(C_j, c_k)$ and $FP(C_j, c_k)$ rates over the validation set.

2. For each class c_k, $1 \leq k \leq c$, combine into the meta-classifier the (base-) classifiers C_j for which one of the two, or both, hold:

$$
\begin{aligned}
TP(C_j, c_k) &= max(TP(C_i, c_k)), 1 \leq i, j \leq C && (5) \\
FP(C_j, c_k) &= min(FP(C_i, c_k)), 1 \leq i, j \leq C. && (6)
\end{aligned}
$$

3. Proceed, if needed, in a similar fashion and recursively build higher level meta-classifier trees.

Two-sided class specialty metrics: The problem with the one-sided class specialty metric is that it may qualify poor classifiers. Lets take for example the extreme case of a classifier that predicts always the class c_k. This classifier is highly biased and the algorithm will select it. So, we define two new metrics, the *positive combined specialty* $PCS(C_j, c_k)$ and the *negative combined specialty* $NCS(C_j, c_k)$ 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 later is biased towards FP rates:

$$
\begin{aligned}
PCS(C_j, c_k) &= \frac{TP(C_j, c_k) - FP(C_j, c_k)}{1 - TP(C_j, c_k)} && (7) \\
NCS(C_j, c_k) &= \frac{TP(C_j, c_k) - FP(C_j, c_k)}{FP(C_j, c_k)} && (8)
\end{aligned}
$$

The classifier selection algorithm can be modified by replacing the TP and FP conditions in equations 5 and 6 with the PCS and NCS respectively.

Combined class specialty metric: A third alternative is to introduce 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. The *combined class specialty* metric, or $CCS(C_j, c_k)$, is defined as:

$$
CCS(C_j, c_k) = f_{TP}(c_k) \cdot TP(C_j, c_k) + f_{FP}(c_k) \cdot FP(C_j, c_k) \tag{9}
$$

where $-1 \leq f_{TP}(c_k), f_{FP}(c_k) \leq 1, \forall c_k$. The algorithm above (step-2) is modified to choose and integrate into a meta-classifier, the (base-) classifiers that exhibit the highest *combined class specialty* for each class. Coefficients functions f_{TP} and f_{FP} are single variable functions quantifying the importance each class according to the needs of the problem and the distribution of each class in the entire data set[4]. In many real world problems, e.g. medical data diagnosis, credit card fraud,

[4]A more general and elaborate specialty metric may take into account the individual instances as well:

$$
CCS(C_j, c_k, y_i) = f_{TP}(c_k, y_i) \cdot TP(C_j, c_k, y_i) + f_{FP}(c_k, y_i) \cdot FP(C_j, c_k, y_i) \tag{10}
$$

etc, the plain $TP(C_j, c_k)$ and $FP(C_j, c_k)$ rates fail to capture the entire story. The distribution of the classes in the data set may not be balanced and maximizing the TP rate of one class may be more important than maximizing total accuracy. In the credit card fraud detection problem, for instance, catching expensive fraudulent transactions is more vital than eliminating the possibility for false alarm. The *combined class specialty metric* provides the means to associate a cost model with the performance of each classifier and evaluate the classifiers from a different perspective.

In all versions, the algorithm evaluates all available (base-)classifiers and then selects the ones with the highest specialty per class. Recall that high specialty for a class means high accuracy for that class, so, in essence, the algorithm chooses the (base-)classifiers with the most specialized and accurate view of each class. We expect that a meta-classifier trained on these (base-) classifiers will be able to uncover and learn their bias and take advantage of their properties.

Aggregate specialty: The *class specialty* metrics concentrate on the accuracy of a classifier on one particular class. Alternatively, we can characterize a classifier by measuring its "total" *specialty*, that is, its specialty when taking into account all classes together. For this we introduce a new metric the *aggregate specialty* $AS(C_j)$ metric of a classifier C_j as:

$$AS(C_j) = \sqrt[c]{\prod_{i=1}^{c} TP(C_j, c_i)} \tag{11}$$

which is basically the geometric mean of the accuracies measured on each class separately. The geometric mean of c quantities, reaches high values only if all values are high enough and in balance. In our case, $AS(C_j)$ has high values when classifier C_j performs relatively well on all c classes. A highly specialized classifier, on the other hand, exhibits lower $AS(C_j)$ values. This metric can prove very useful with skewed data sets [22] 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.

Combining metrics: This strategy can be extended in a straight forward fashion to employ other metrics as well, such as the metrics examined in section 3. Furthermore, instead of relying just on one criterion to choose the (base-) classifiers, we 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 example, one possible approach would be to combine the (base-) classifiers with high *coverage* and low *correlation error*. A different strategy that combines *CCS rates* and *coverage* would be to iteratively select classifiers based on their *CCS* score on examples which the preceding classifiers failed to cover. Other possible scenarios involve blending *general* classifiers (classifiers with high *aggregate specialties*) and highly *specialized* classifiers, or *general* classifiers that also exhibit high *diversity*. In another study [41] concerning credit card fraud detection the authors employ evaluation formulas for selecting classifiers that are based on characteristics such as *diversity*, *coverage* and *correlated error* or their combinations, i.e. *True Positive rate* and *diversity*. The investigation of the various pre-training pruning methods and their quality is a major component in this thesis research.

4.1.2.3 Post-training pruning of the meta-classifier

Post-training pruning refers to the partial or full dismissal of a meta-classifier after it is built. For a number of reasons, e.g. when pre-training pruning selects poor (base-) classifiers, or when the meta-learning algorithm fails to detect the bias of each underlying (base-) classifier and use it accordingly, a meta-classifier may not perform as well as expected. Our objective is to detect these inferior meta-classifiers as soon as possible and revise and/or discard them. For this, we introduce two additional stages where we evaluate the meta-classifiers; the first is immediately after the meta-learning phase and the second is during the prediction phase. During the former phase, we employ heuristic methods to assess the quality of the new meta-classifier, while in the later we can rely on the actual performance results that are available.

The difficulties in evaluating the meta-classifiers during the first screening stage originate from the fact that they are brand new and have not been tested yet, hence there can be no certain way to determine their quality. The second screening stage is free of this problem, but requires the presence of an expert to supervise periodically the predictions of the base- and meta-classifiers and determine their performance. Next, we describe some heuristic methods and rules and some evaluation metrics that we will introduce to overcome these obstacles:

Correlation metric: Given $C+1$ classifiers C_1, C_2, ... C_C and C' and a data set of n examples mapped onto c classes, we can construct a two dimensional $c \times C$ contingency matrix where each cell $M_{ij}^{C'}$ 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 $M_{ij}^{C'}$ 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 a given column j^{th} would add up to n only if C' and C_j produce identical predictions.

We call matrix $M_{ij}^{C'}$ the *correlation matrix* of C' since it captures the correlation information of the C' classifier with all the other classifiers $C_1 ... C_C$. In particular, given this matrix, we can define:

$$Corr(C', C_j) = \frac{\sum_{i=1}^{c} M_{ij}^{C'}}{n} \tag{12}$$

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 *post training pruning* context, we employ the above equation to compute the correlation $Corr(MC, C_j)$ between the meta-classifier MC and the (base-) classifiers C_j, $1 \leq j \leq C$, to get a measure of how much the meta-classifier relies on that (base-)classifier for its predictions. If evaluated and interpreted properly, the *correlation* metric can reveal valuable information regarding the quality of the meta-classifier. 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. Post-training pruning should be able to discard the meta-classifier in both cases and trigger the training of a different one.

Meta-classifier class specialty: The *correlation* metric $Corr(MC, C_j)$ concentrates on the dependencies of the meta-classifier with its constituent (base-) classifiers. An alternative would be to inspect closely the information registered in the *correlation matrix* M_{ij}^{MC} of the meta-classifier to determine the details of the relations between the meta-classifier and the (base-) classifiers with respect to the specialties of the (base-) classifiers. Recall that a specific row k of the M_{ij}^{MC} matrix provides information as to which (base-) classifier the meta-classifier trusts more when predicting class k. By comparing the (base-)classifier with the highest TP on the validation set for a specific class k to the (base-) classifier obtained by:

$$\underset{\mathrm{i=k}}{argmax}(M_{ij}^{MC}) \tag{13}$$

it is possible to assess whether the meta-learning agent had the acuteness to discern the specialties of the (base-) classifiers.

In all cases, detecting in early stages the classifiers that may potentially exhibit inferior performance is, undeniably, a desirable characteristic that can save from unwelcomed results and a lot of trouble. In general, post-training pruning may recur periodically to re-evaluate the meta-classifiers.

Post-training pruning can take advantage of information such as the true labels of the instances. By comparing the predictions of the meta-classifiers to the real labels, post-training pruning acquires first hand information regarding the actual performance of the meta-classifiers including the TP and FP rates, the CCS and AS scores, the correlated errors and the accuracy results. Furthermore, it can monitor their responses and detect the meta-classifiers with decreasing accuracy over time. Such phenomena usually signal changes in the patterns of unclassified data and should usually trigger corresponding changes in the meta-classifiers to compensate.

4.2 Adaptivity

Most classification systems operate in environments that are almost certainly bound to change. 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. Traditional learning systems, however, are static, with no means to adapt to their changing environment and no chance to maintain their performance. It is desirable to allow classifiers to evolve and adapt to the changes of their environment. As more and more information becomes available, adaptive classification systems should be able to extract "fresh" information, discern new trends and improve their old models.

4.2.1 Problem description

The classifiers C_k deployed in the traditional classification systems are usually obtained by applying machine learning programs over historical databases DB_i. In most of the cases, these machine learning programs are main memory based, meaning that they require the entire training data set loaded in main memory before they can begin the learning process. The resulting classifiers are tuned to fit the initial database and cannot be modified once created. The problem is to design a classification system that can evolve in case a new database DB_j becomes available.

One way to address this problem is to merge the old and new databases into a larger database DB and re-apply the machine learning programs to generate new classifiers. This, however, cannot 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 [45], an incremental version of ID3) i.e. machine learning programs that are not constrained to retain all training examples in main memory. The classifiers C_k initially trained over DB_i can be updated later by resuming their training on the new database

DB_j once it becomes available. The problem with this approach is that it is not general enough; instead it relies on specific algorithms and implementations and to our knowledge there are very few incremental machine learning algorithms implemented. The main memory requirement is inherently imposed by the nature of the current generation of machine learning algorithms.

We need a strategy that 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 should allow JAM to extend and incorporate new information without discarding or depreciating the knowledge it has accumulated over time.

4.2.2 Approach

We propose a new mechanism for integrating new information. New information is treated in a fashion similar to the information imported from remote data sites. Instead of combining classifiers from remote data sites (integration over space), adaptive learning systems combine classifiers acquired over different time periods (integration over time). We will employ meta-learning techniques to design learning systems capable of incorporating into their accumulated knowledge (existing classifiers) the new classifiers that capture emerging patterns.

In addition to solving the problem of how to make a learning system evolve and adjust according to its changing environment, the 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 simple and straight-forward manner.

2. It integrates uniformly with the existing approach 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 modifications.

4. It is module-oriented and efficient. The meta-learning based system does not have to repeat the entire training process from the beginning in order to create models that integrate new information (i.e capture characteristics of the entire (old and new) data set). Instead it can build independent models that would be able to plug-in into 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. 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. Normally, incorporating new classifiers in a meta-classifier hierarchy continuously would eventually result in large and inefficient tree-type hierarchies. But since the new classifiers are not different in nature from the "traditional" classifiers, it is possible that the meta-learning based system can analyze and compare them (pre- and post-training pruning) and keep only those that contribute to the overall accuracy and not over-burden the meta-classification process. For example, it can decide to collapse or substitute a sub-tree of the meta-classifier hierarchy with newly obtained classifier(s) that capture the same or more patterns.

4.3 Compatibility

Occasionally, classifiers for the same classification problem are induced from data sets with different schemas. Depending on the problem and the distance between the two schemas, it may be possible to combine the information carried by the different classifiers into a higher level meta-classifier. Assume, for instance, two data sets of credit card transactions from two different financial institutions (i.e. banks) and assume that the problem is to learn the patterns that can distinguish legitimate from fraudulent use of a credit card. It is desirable for both institutions to be able to exchange their classifiers and hence incorporate in their system useful information that would otherwise be inaccessible. Indeed, for each credit card transaction, both institutions record similar information, however, they also include specific fields containing important information that each has determined separately and which provides predictive value in determining fraudulent transaction patterns. The integration of this information across separately learned classifiers at each bank site is a non-trivial problem, and we call it *"the different schema integration"* problem.

4.3.1 Problem description

Lets consider two data sites A and B with databases DB_A and DB_B respectively with similar but not identical schemas. Without loss of generality, we assume that:

$$Schema(DB_A) = \{A_1, A_2, ..., A_n, A_{n+1}, C\} \tag{14}$$
$$Schema(DB_B) = \{B_1, B_2, ..., B_n, B_{n+1}, C\} \tag{15}$$

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) attribute 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. $A_{n+1} \approx B_{n+1}$: The two attributes are of similar type but slightly different semantics, e.g. A_{n+1} and B_{n+1} are fields with time dependent information but of different duration (i.e. 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).

2. $A_{n+1} \neq B_{n+1}$: The two attributes are of different type. Without lose of generality the problem can then be reduced to the following:

$$Schema(DB_A) = \{A_1, A_2, ..., A_n, A_{n+1}, C\} \tag{16}$$
$$Schema(DB_B) = \{A_1, A_2, ..., A_n, C\} \tag{17}$$

where we assume that attribute B_{n+1} is not present in DB_B.

In both cases (attribute B_{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. But the purpose of using a distributed data mining system and deploying learning agents and meta-learning their classifier agents is to be able to combine information from different sources. A major component of this thesis research will be to investigate ways, called *bridging* methods to overcome this incompatibility problem and integrate classifiers originating from databases with different schemas.

4.3.2 Approach

There are several possible approaches to address the *different schema integration* problem depending upon the type of the learning problem and the characteristics of the different or missing attribute B_{n+1} of DB_B:

- **Attribute B_{n+1} is missing, but can be predicted:** It may be possible to create an auxiliary classifier, which we call a *bridging 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 [3], locally weighted regression [5], MARS [20]) for continuous attributes and machine learning 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 $A_1, ..., An$. Then it can send all its local (primary and auxiliary) 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 A_{n+1} attribute and create a new database DB_B' with schema $\{A_1, ..., A_n, \hat{A}_{n+1}\}$ which can then be used in conjunction with classifier C_{Aj}.

- **Attribute B_{n+1} is missing and cannot be predicted:** To compute a model for the missing attribute implies that there is a correlation between that attribute and the rest. Nevertheless, such an assumption may be unwarranted in which case we adopt one of the following strategies:

  - **Classifier agents 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 (also known as "don't know" values) in a bogus A_{n+1} attribute added to DB_B. The resulting DB_B' database is a database compatible to 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.
  - **Learning agents at data site B can not 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:

    $$DB_A' = PROJECT \ \ (A_1, A_2, ..., A_N) \ \ FROM \ \ DB_A \tag{18}$$

    The first classifier can be stored locally for later use by the local meta-learning agents, while the later, can be sent over 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 only of 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 B_{n+1} is present, but semantically different:** It may be possible to integrate human expert knowledge and introduce auxiliary *bridging* agents either from data site A, or data site B that can preprocess the B_{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 project 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).

All these strategies address the *data schema integration* problem and meta-learning over these models should proceed in a straightforward manner [10].

5 Evaluation and Discussion

In this chapter we describe in detail the current status of the JAM system and we present our experiments in the management of classification models. The experiments focus on methods for evaluating the classification models, on algorithms searching for effective and efficient meta-level classification models and on strategies for combining seemingly incompatible classification models.

We assess our methods by evaluating the performance of their resulting classifiers. To evaluate the performance of base-classifiers or meta-classifiers we divide our original data set into two disjoint subsets, one called *training set* and one called *testing set*. A learning program is then presented with the training set and computes a classifier (concept) based on examples in the set; this is the *training* phase. To evaluate the classifier produced by the learning program, instances from the test set are presented for classification and *accuracy* is measured; this is the *testing* phase. To ensure randomness and acquire results with confidence, we use a *k-fold cross validation* [3] approach, that is, we perform the experiment k different times and we average the results. Each time we use a different partitioning of the data into training and testing sets. The entire data set is divided into k randomly chosen subsets, and each of which is used in turn as the test set while the rest form the training set.

Learning algorithms Five inductive learning algorithms are used in our experiments. ID3, its successor C4.5 [38], and Cart are decision tree based algorithms, Bayes, described in [16], is a naive bayesian classifier that is based on computing conditional probabilities, and Ripper [13] is a rule induction algorithm based on IREP [1].

Learning tasks Two data sets of real credit card transactions and two molecular biology sequence analysis data sets, were used in our experiments. The credit card data sets were provided by the Chase and First Union Banks, members of FSTC (Financial Services Technology Consortium) and the molecular biology sequences were obtained from the UCI Machine Learning repository [27].

The first two data sets contained credit card transactions labelled as fraudulent or legitimate. Each bank supplied .5 million records spanning one year. Chase bank data consisted, on average, of 42,000 sampled credit card transactions records per month with 20% fraud versus 80% non-fraud distribution, whereas First Union data were asymmetrically sampled (many records from some months, very few from others) with 15% fraud versus 85% non-fraud. The schemas of the databases was developed over years of experience and continuous analysis by bank personnel to capture important information for fraud detection. We cannot reveal the details of the schema beyond what is described in [42]. The records have a fixed length of 137 bytes each and about

30 numeric attributes including the binary class label (fraud/legitimate transaction). Some of the fields are arithmetic and the rest categorical, i.e. numbers were used to represent a few discrete categories.

The secondary protein structure data set (SS) [36], courtesy of Qian and Sejnowski, contains 21,625 sequences of amino acids and secondary structures at the corresponding positions. There are three structures (classes) and 20 amino acids (21 attributes because of a spacer [36]) in the data. The amino acid sequences were split into shorter sequences of length 13 according to a windowing technique used in [36]. There is one such sequence per example.

The DNA splice junction data set (SJ) [44], courtesy of Towell, Shavlik and Noordewier, contains 3,190 sequences of nucleotides and the type of splice junction, if any, at the center of each sequence (three classes). Each sequence (example) has 60 nucleotides with eight different values each (four base ones plus four combinations).

5.1 The JAM architecture

The architecture of the system has already been described in section 4. We have used JAVA technology to build the infrastructure of the system and developed the specific *agent operators* that compose and spawn new agents from existing classifier agents. JAVA technology provides the means to dispatch agents to remote sites and execute them under remote or local control. The graphical user interface, the animation facilities and most of the machine learning algorithms were also implemented in JAVA. The only parts that were imported in their native (C++) form were some of the machine learning programs; and this was done for faster prototype development and proof of concept. The *JAM* system builds upon the existing agent infrastructure available over the internet today. The platform-independence[5] of JAVA technology makes it easy to port *JAM* and delegate its agents to any participating site. (The modules that are implemented in native C++ are not yet platform independent.) For more details on the *JAM* system refer to [43].

Next we describe the various components of our distributed data mining system.

5.1.1 Configuration Manager

The CM assumes a role equivalent to that of a name server of a network system. It is responsible for maintaining the "global" configuration of the system and making it available to the participating Datasites.

The CM provides registration services to all Datasites that wish to become members and participate in the distributed meta-learning activity. When the CM receives a JOIN request from a new Datasite, it verifies both the validity of the request and the identity of the Datasite. Upon success, it acknowledges the request and registers the Datasite as active. Similarly, the CM can receive and verify the DEPARTURE request; it notes the requestor Datasite as inactive and removes it from its list of members. The CM, maintains the list of active member Datasites to establish contact and cooperation between peer Datasites. Apart from that, the CM keeps information regarding the groups that are formed (which Datasites collaborate with which Datasites), logs the events and displays the status of the system. Through the CM, the *JAM* system administrator may screen the Datasites that participate.

[5]In fact, *JAM* was been demonstrated on various occasions under several platforms including Solaris x86, Sparc Solaris, Windows NT and Windows 95

5.1.2 Datasites

Unlike CM which provides a passive configuration maintenance function, the Datasites are the active components of the meta-learning system. The Datasites are responsible for running the show. They avoid the overheads of centralized control and synchronization points by realizing a distributed and asynchronous protocol. A Datasite manages its local database, builds local classifiers, obtains remote classifiers, builds local meta classifiers and interacts with a JAM user. A Datasite is implemented as a multithreaded Java program with a special GUI.

Upon initialization, a Datasite starts up the GUI through which it can accept input and display status and results. During its initialization, among its other tasks, the Datasite registers with the CM, instantiates the local learning engine/agent[6] and creates a server socket for listening for connections[7] from the peer Datasites.

The Datasite is a module driven by input messages or commands. After initialization is complete the Datasite waits for the next event to occur. This can be either

1. A command issued by the owner via the GUI, or
2. A message from a peer Datasite via the open socket.

In both cases, the Datasite verifies that the input is valid and can be serviced. Once this is established, the Datasite allocates a separate thread and performs the required task. This task can be any of JAM's functions: computing a local classifier, starting the meta-learning process, sending local classifiers to peer Datasites or requesting remote classifiers from them, reporting the current status, or presenting computed results.

Figure 4 presents a snapshot of the *JAM* system during the meta-learning phase. In this example three Datasites, Marmalade, Strawberry and Mango (see the group panel of the figure) collaborate in order to share and improve their knowledge in diagnosing hypothyroidism. The snapshot taken is from "Marmalade's point of view". Initially, Marmalade consults the Datasite configuration file where the owner of the Datasite sets the parameters. In this case, the data set is a medical database with records, noted by thyroid in the Data Set panel. Other parameters include the host of the CM, the Cross-Validation Fold, the Meta-Learning Fold, the Meta-Learning Level, the names of the local learning agent and the local meta-learning agent, etc. Refer to [6] for more information on the meaning and use of these parameters. (Notice that Marmalade has established that Strawberry and Mango are its peer Datasites, having acquired this information from the CM.)

Then, Marmalade partitions the thyroid database (noted as thyroid.1.bld and thyroid.2.bld in the Data Set panel) for the 2-Cross-Validation Fold, and computes the local classifier, noted by Marmalade.1 (here by calling the ID3 learning agent) viewed at the main panel. Next, Marmalade Marmalade imports the remote classifiers, noted by Strawberry.1 and Mango.1 and begins the meta-learning process. The snapshot of Figure 4 displays the system at this stage. In the animated meta-learning process *JAM*'s GUI moves icons within the panel displaying the construction of a new meta-classifier. Marmalade will use this meta-classifier in the future to predict the classes of input data items (in this case unlabelled medical records).

The owner of a *JAM* Datasite has direct control over the stages and the progress of the learning and meta learning process. He/She can observe the internals of the generated classifiers and meta classifiers and get reports on the results and statistics.

[6] The Datasite consults the local Datasite configuration file (maintained by the owner of the Datasite) to obtain information regarding the central CM and the types of the available machine learning agents.

[7] For each connection, the Datasite spawns a separate thread.

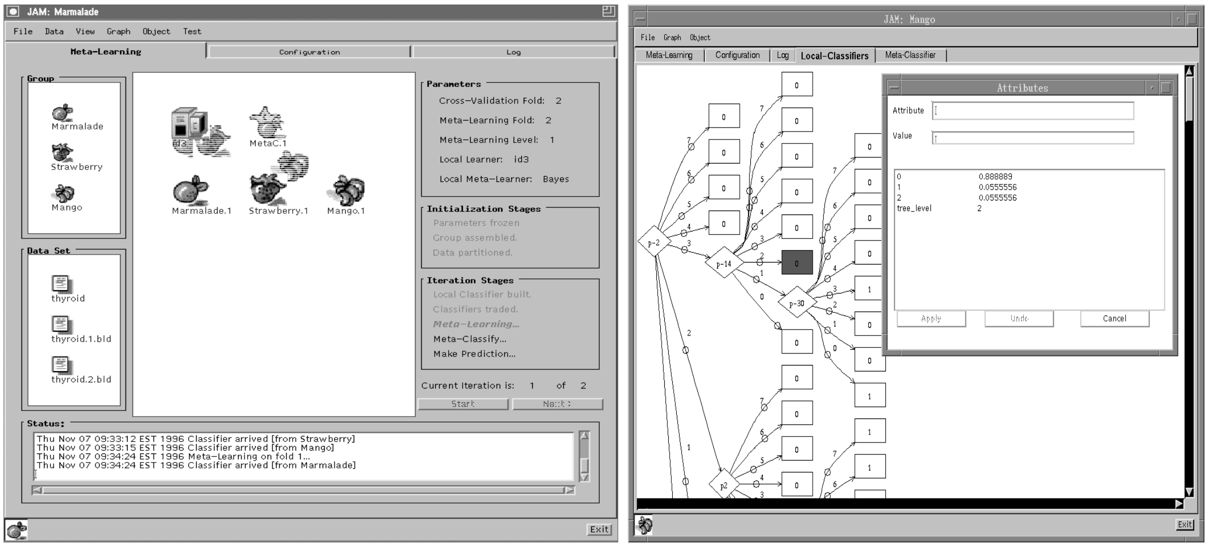

Figure 4: Two different snapshots of the JAM system in action. Left: Marmalade is building the meta classifier (meta learning stage). Right: A ID3 tree-structured classifier is being displayed in the Classifier Visualization Panel

Specifically, JAM provides graph drawing tools to help users understand the learned knowledge [18]. There are many kinds of classifiers, e.g., a decision tree by ID3, that can be represented as graphs. In JAM we have employed major components of *Grappa* [24], an extensible visualization system, that displays the classifier and allows the user to analyze the graph. Since each machine learning algorithm has its own format to represent the data classifier, JAM uses an algorithm-specific translator to read the classifier and generate a comprehensible representation.

Figure 4 shows the JAM classifier visualization panel with a decision tree, where the leaf nodes represent classes (decisions), the non-leaf nodes represent the attributes under test, and the edges represent the attribute values.

5.1.3 Agents

JAM's extensible plug-and-play architecture allows snapping-in learning agents. The learning and meta-learning agents are designed as objects. JAM provides the definition of the parent agent class and every instance agent (i.e. a program that implements any of your favorite learning algorithms ID3, Ripper, Cart, Bayes [15], Wpebls [14], CN2, etc.) is then defined as a subclass of this parent class. Among other definitions which are inherited by all agent subclasses, the parent agent class provides a very 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 in the JAM system. To be more specific, a JAM agent needs to have the following methods implemented for JAM to use it effectively:

1. A *constructor method* with no arguments. JAM can then instantiate the agent, provided it knows its name (which can be supplied by the owner of the Datasite through either the local user configuration file or the GUI).

2. An *initialize() method*. In most of the cases, if not all, the agent subclasses inherit this method from the parent agent class. Through this method, JAM can supply the necessary

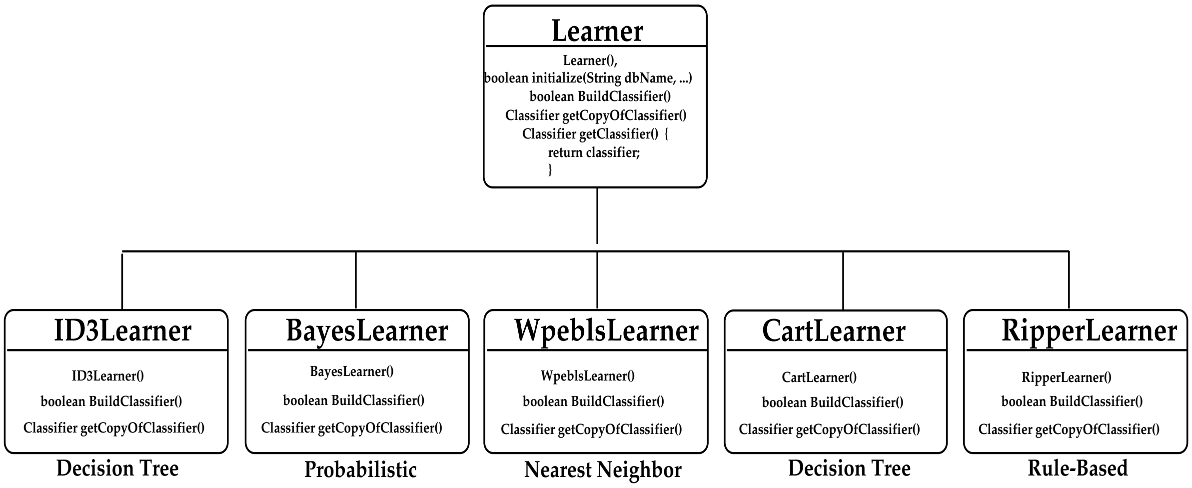

Figure 5: The class hierarchy of learning agents.

arguments to the agent. Arguments include the names of the training and test data sets, the name of the dictionary file, and the filename of the output classifier.

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

4. A *getClassifier()* and *getCopyOfClassifier() methods*. These methods are used by JAM to obtain the newly built classifiers. These are then encapsulated and can be "snapped-in" at any other participating Datasite! Hence, remote agent dispatch is easily accomplished.

The class hierarchy (only methods are shown) for five different learning agents is presented in Figure 5. ID3, Bayes, Wpebls, Cart and Ripper inherit the methods *initialize()* and *getClassifier()* from their parent learning agent class. The Meta-Learning, Classifier and Meta-Classifier classes are defined in similar hierarchies.

JAM is designed and implemented independently of the machine learning programs of interest. As long as a machine learning program is defined and encapsulated as an object conforming to the minimal interface requirements (most existing algorithms have similar interfaces already) it can be imported and used directly. This plug-and-play characteristic makes JAM truly powerful and extensible data mining facility.

5.2 Pre-training pruning experiments

Pre-training pruning refers to the evaluation and selection of classifiers before they are used for the training of the meta-classifier. In these initial experiments we use the Chase and First Union data sets, the set of the five machine learning programs and two pre-training pruning methods:

1. a method that combines the coverage metric with the (TP-FP) metric and

2. a method that combines the coverage metric with a cost model

First, we prepared the set of candidate base classifiers, i.e. the original set of base classifiers the pre-training pruning algorithm is called to evaluate. We obtained these classifiers by dividing the original data sets into 12 non-overlapping subsets and by applying the 5 learning algorithms on

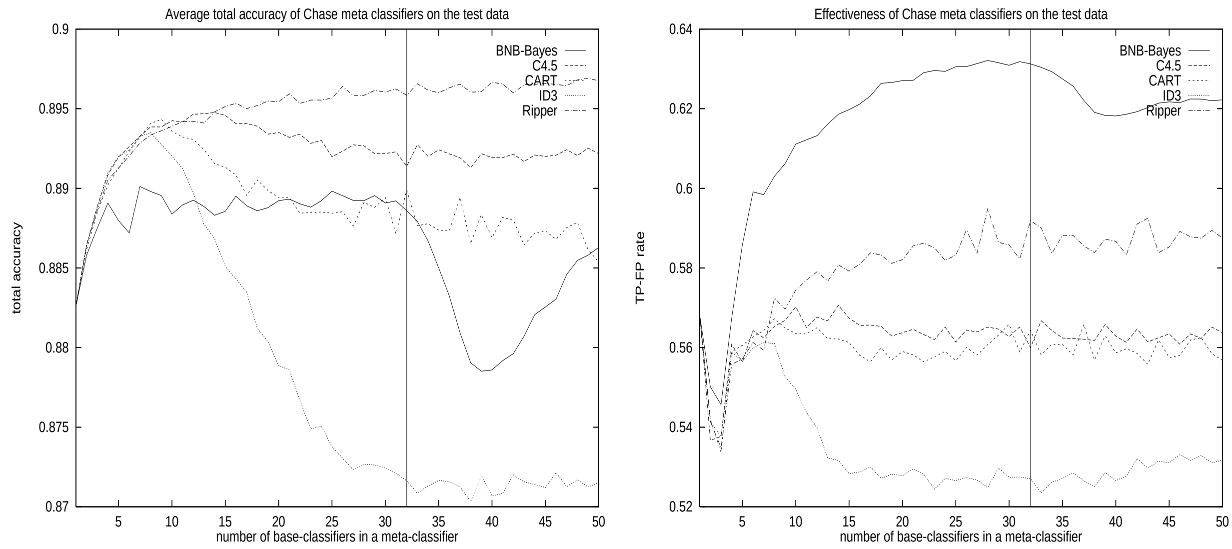

Figure 6: Total accuracy and $TP - FP$ graphs on CHASE credit card data.

each of these subsets. Overall we created 60 base classifiers for each data set. In our experiments we use a 6-fold cross validation approach, meaning that in each fold we use 10 subsets for training and we keep one subset for the meta-level training phase and one for the meta-level testing phase.

In each fold, there are 50 candidate base classifiers (10 *subsets* $\times$ 5 *learning algorithms*), hence there are $(2^{50} - 50)$ different possible meta-classifiers from which the pre-training pruning algorithm had to choose one. The evaluation and selection of the base classifiers is based on the subset that is used as the meta-level training set. The overall performance of the pre-training pruning method is judged against the subset used as the meta-level testing set. Both pre-training pruning algorithms that we examine here, follow a greedy approach:

Coverage/(TP-FP) combined metric: This algorithm combines the *coverage* metric and the $TP(C_j, fraud) - FP(C_j, fraud)$ rate[8]. Initially, the algorithm starts by choosing the base classifier with the best $TP(C_j, fraud) - FP(C_j, fraud)$ rate on the validation set. Then it continues by iteratively selecting classifiers based on their $TP(C_j, fraud) - FP(C_j, fraud)$ performance on the examples which the previously chosen classifiers failed to cover. The algorithm ends when there are no other examples to cover.

The results from this experiment are displayed in figures 6 and 7. The first figure presents the accuracy and $TP - FP$ rates of the Chase credit card data and the second figure of the First Union data. The vertical lines in the two figures denote the number of base classifiers integrated in the final meta-classifiers as determined by the algorithm. The final Chase meta-classifier combines 32 base classifiers, while the final First Union meta-classifier consists of 22 base classifiers. In these graphs we have also included the intermediate performance results (i.e. the accuracy and $TP - FP$ rates of the partially built meta-classifiers) as well as the performance results the redundant meta-classifiers would have had, had we used more base-classifier or not introduced the pre-training pruning phase.

The benefits from this method are clear. In both cases the performance of the pruned meta-classifiers is superior to that of the complete meta-classifier[9]also exhibit both accuracy-wise ($TP -$

[8]A simple case of the $CCS(C_j, fraud)$ rate.

[9]Pruned meta-classifiers are superior to the best base-classifiers as well.

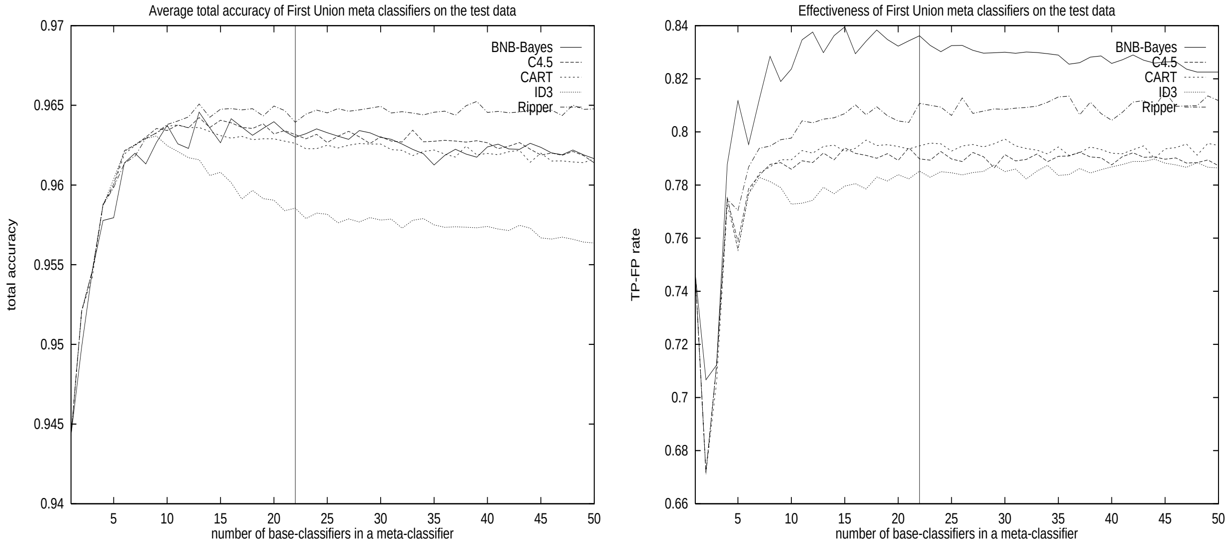

Figure 7: Total accuracy and $TP - FP$ graphs on First Union credit card data.

FP, overall accuracy) and efficiency-wise(less base-classifiers are retained); using more base-classifiers than selected (denoted by the vertical lines) has no positive impact on the performance of the meta-classifiers. Even though the algorithm performs a greedy search, it combines classifiers that are diverse (they classify correctly different subsets of data), accurate (they have the highest $TP - FP$ rate on the data set used for evaluation) and with high coverage.

Coverage/Cost Model combined metric In this set of experiments, we employ a cost model to assess the performance of the base- and meta-classifiers. The algorithm combines the coverage metric with the cost model $SAVINGS(C_j)$ of a classifier C_j.

We are experimenting with data sets of credit card transactions, hence our cost model associates the performance of a classifier with its ability to save more or less money. We seek to produce classifiers and meta-classifiers that generate the maximum savings (in dollars). For concreteness,

$$tranamt = the\ transaction\ amount\ associated\ with\ each\ transaction \tag{19}$$

$$tp?(C_j, y_i) = \begin{cases} 1 & \text{if } C_j(y_i) = L(y_i) = fraud \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

$$fp?(C_j, y_i) = \begin{cases} 1 & \text{if } C_j(y_i) = fraud \ \ and \ \ L(y_i) = legitimate \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

Credit card companies have introduced a threshold value for challenging the legitimacy of the use of a credit card. In other words, if *tranamt* 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 $\$X$. Therefore, even if we could accurately predict and identify all fraudulent transactions, those whose *tranamt* is less than $\$X$ would produce $\$X - tranamt$ in losses anyway. With this overhead cost taken into account, we define our cost model to be:

$$SAVINGS(C_j) = \sum_{i=1}^{n} [tp?(C_j, y_i) \times savings(C_j, y_i) - fp?(C_j, y_i) \times \$X] \tag{22}$$

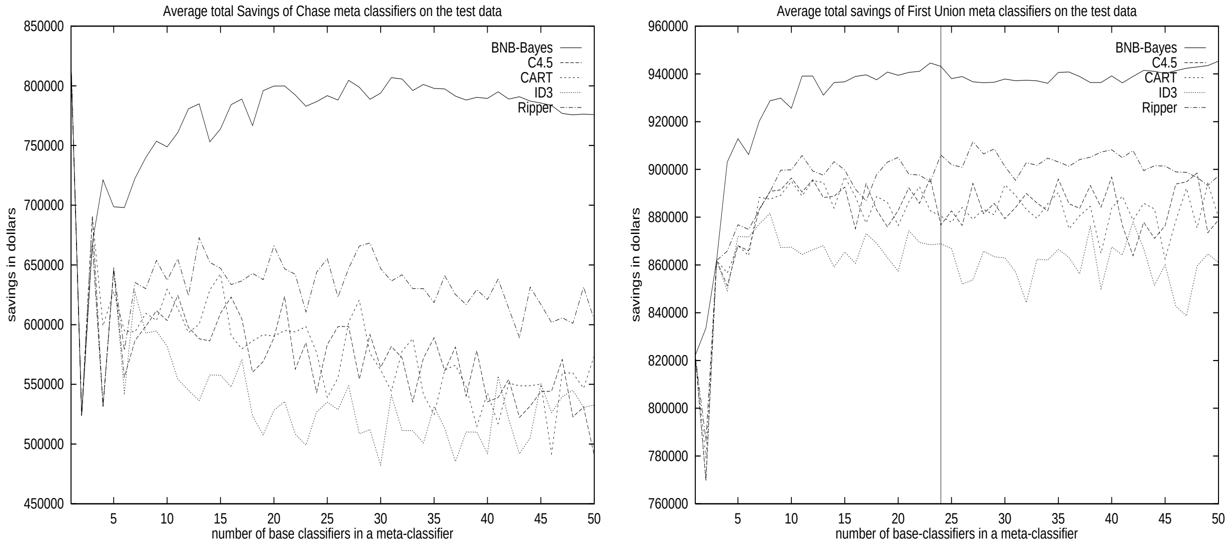

Figure 8: Chase and First Union savings (dollars).

where $savings(C_j, x_i)$ is defined as:

$$savings(C_j, y_i) = \begin{cases} tranamt(y_i) & \text{if } tranamt(y_i) \geq \$X \\ \$X - tranamt(y_i) & \text{otherwise} \end{cases} \tag{23}$$

In every other respect, the algorithm is similar to the *coverage/(TP-FP)* algorithm presented earlier. Initially, the algorithm starts by choosing the base classifier with the highest $SAVINGS(C_j)$ on the meta-level training set. Then it continues by iteratively selecting classifiers based on their $SAVINGS(C_j)$ amount on the examples which the previously chosen classifiers failed to cover. Again, the algorithm ends when there are no other examples to cover.

The results of this experiment are displayed in figure 8. The first plot presents the savings in dollars on the Chase credit card data while the second plot refers to First Union data. The final Chase meta-classifier combines 32 base classifiers, while the final First Union meta-classifier consists of 24 base classifiers. In these graphs we have also included the intermediate performance results as well as the performance results the meta-classifier would have had, had we not introduced the pre-training pruning phase.

In contrast to the results of the previous experiment and the performance of the First Union meta-classifiers of this experiment, the Chase meta-classifiers exhibit inferior performance to that of the "best" Chase base-classifier. Upon closer inspection, however, we determined that this is not due to a shortcoming of the pre-training pruning method, but due to the nearsightedness of the meta-learning algorithm and the ill-definition of the learning task of the base-learners. Specifically, with only two exceptions, the Chase base-classifiers were inclined towards catching "cheap" fraudulent transactions and for this they exhibited low savings scores. After all, the base-classifiers were unaware of the adopted cost model and the actual value (in dollars) of the fraud/legitimate label. Similarly, the meta-learners were trained to maximize the overall accuracy not by examining the savings in dollars but by relying on the predictions of the base-classifiers alone. Naturally, the resulting meta-classifiers trusted the wrong base-classifiers for the wrong reasons. In the First Union experiments, on the other hand, where most of the First Union base-classifiers could catch the "expensive" fraudulent transactions, the pre-training pruning methods proved to be beneficial. One way to rectify the Chase situation is to tune the learning problem according to the adopted cost model. For example, we can transform the binary classification problem into a c-class problem by multiplexing the binary class and the *tranamt* attribute into a new continuous class (target) and by dividing the values of this class into c "bins". In the credit card case, for instance, we can replace the binary class with a new class with {\$0-\$X}-F, {\$X-\$Y}-F, {\$0-\$X}-L, {\$X-\$Y}-L,..., as example values, where \$X and \$Y are *tranamt* values and F,L stand for the fraud/legitimate label. The classifiers derived from the modified problem would presumably fit better to the specifications of the cost model and achieve better results.

In any case, the *post-training pruning* stage should act as a net and detect the potentially inferior meta-classifiers.

5.3 Post-training pruning

Post-training pruning refers to the partial or full dismissal of a meta- classifier after it is built. In the experiments presented in this section we demonstrate that occasionally meta-learners fail to distinguish the patterns and biases of their (base-) classifiers and thus generate meta-classifiers that cannot take full advantage of their (base-) classifiers' potential. In this set of experiments we use the SS and SJ data sets and we employ the *correlation* metric that was introduced in section 4 to evaluate the relations between meta-classifiers and their constituents (base-) classifiers. As we will demonstrate, the *correlation* metric may be able to uncover the meta-classifiers with weaknesses.

First we divided the original data set into 3 subsets, the training data set, the validation data set and the testing data set. The candidate base classifiers were generated by applying the five machine learning programs on the same training data set. Similarly, the final meta-classifiers were generated by applying the five machine learning programs on the meta-level training data sets. The meta-level training data sets were formed from the predictions of the base classifiers over the validation data set. Naturally, different combinations of base-classifiers generated different meta-level training sets and hence different meta-classifiers.

Among the five base classifiers obtained from the *SS* data set, Bayes exhibited the best performance topping all categories (e.g overall performance, class specialties, etc). As a result, only the meta-classifiers that correlated 100% to this base classifier were able to sustain high performance levels. The experiments show that the *correlation* metric can be used to distinguish between the "good" but "redundant" meta-classifiers (they can be replaced by the single Bayes base classifier) and the rest "inferior" meta-classifiers.

Furthermore, the *correlation* metric can be used to assess whether a meta-learning program learned properly its base-classifiers. Figure 9 presents another set of experiments with two different sets of meta-classifiers. The first set involves the four meta-classifiers (Bayes, C4.5, CART and ID3) consisting of two base classifiers (Cart and ID3) each, while the second involves four meta-classifiers (Bayes, C4.5, Cart and ID3) consisting of three base classifiers (C4.5, ID3 and Ripper) each. The left graph shows the correlations of the four meta-classifiers with their constituent base classifiers, while the right graph displays the accuracies of all the related classifiers. The first pair of vertical lines of the left figure, for example, denote the correlation of the Bayes meta-classifier to the CART and ID3 base classifiers respectively. According to these graphs, the Cart meta-learner in the first case, i.e. the third pair, and the C4.5 meta-learner in the second, i.e. the second triplet, failed to discover the correct relations among their base-classifiers (observe the notable differences in the correlations of these meta-classifiers with the correlations exhibited in the rest) and both generated meta-classifiers of worse performance. In both situations, however, the *correlation metric*, succeeded in detecting the flaw without the need of pre-labelled examples or the presence of a human expert.

In the *SJ* data set there is no single base classifier out-ranking the rest. In figure 10 we present two sets of meta-classifiers. The first set describes a situation similar to those encountered in the *SS*

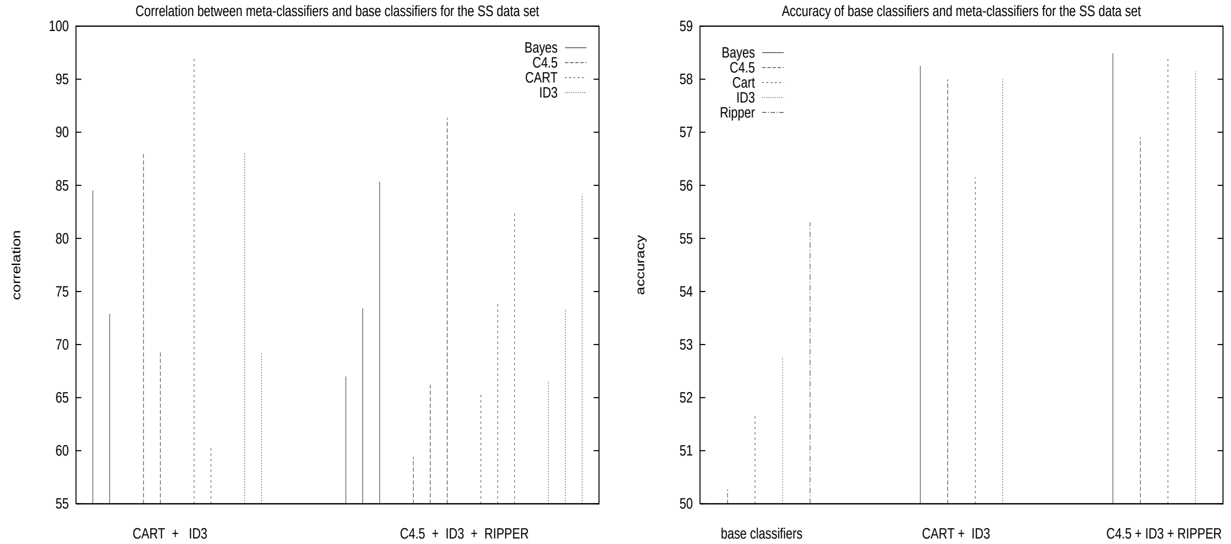

Figure 9: Correlation and accuracy of Meta classifiers for the SS data.

data set, whereas the second set displays a "healthy" situation with all meta-classifiers performing comparably. The first set of meta-classifiers involves the four meta-classifiers (Bayes, C4.5, Cart and ID3) consisting of four base classifiers (Bayes, C4.5, Cart and ID3), while the second involves four meta-classifiers (Bayes, C4.5, Cart and ID3) that consist of three base classifiers (Bayes, C4.5 and Ripper). According to the graphs, the Bayes meta-learner failed to discover the correct relations among its base-classifiers and thus generated a meta-classifiers of worse performance. Again, the *correlation metric*, managed to see through the two cases and identify the problematic meta-classifier.

5.4 Different schema integration

Combining multiple classification models has been receiving increased attention by the machine learning and data mining communities. The majority of the work, however, is concerned with combining models obtained from different subsets (not necessarily distinct) of a single data set[10]. In contrast, in this thesis research, we have also been dealing with classification models generated from distributed non-overlapping data sets. So far, however, all these models were originating from databases of identical schemas.

In this section we focus on integrating otherwise incompatible classifiers, i.e. classification models derived from databases of different schemas. In our initial experiments, we used the Chase and First Union credit card data sets. Although both sets consist of credit card transactions prelabelled as fraudulent or legitimate, each data set also includes special features containing information, determined independently, that provides predictive value in discerning fraudulent transaction patterns. Here, we demonstrate our techniques for circumventing the obstacles imposed by the different schemas by applying Chase classifiers onto the First Union data and vice versa. These techniques enable the exchange of fraud detectors between the two banks and the sharing of valuable information. The two databases had the following differences:

1. Chase included two features not present in the First Union data

[10] E.g. by imposing probability distribution over the instances of the training set, or by creating stratified subsets, or by sub-sampling, etc.

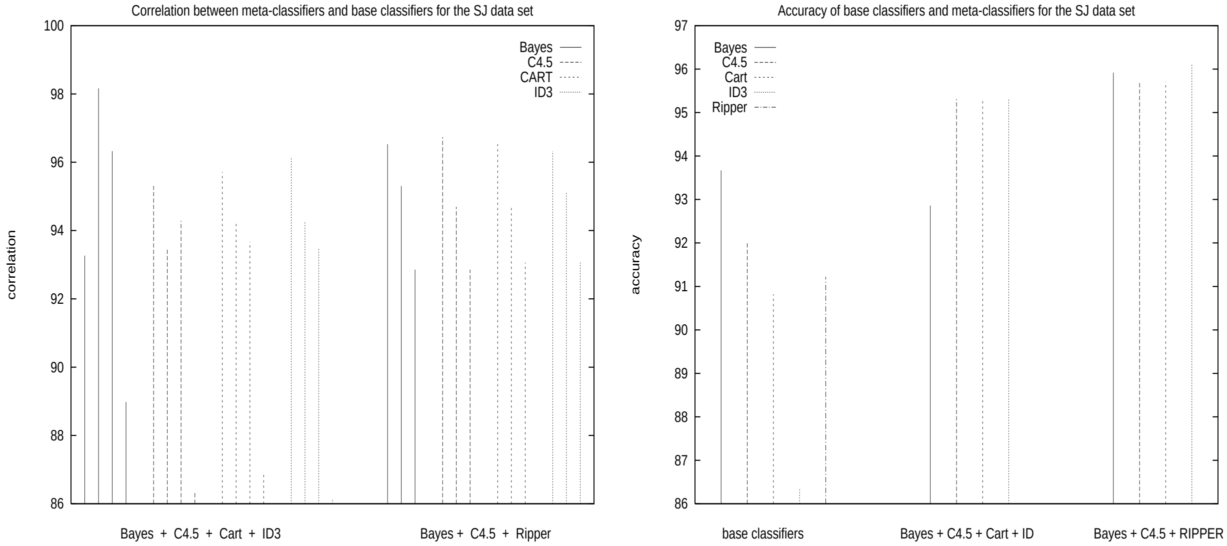

Figure 10: Correlation and accuracy of Meta classifiers for the SJ data.

2. Chase and First Union defined a feature with different semantics

For the **first** incompatibility, we introduced to the First Union data two fictitious fields padded with null values and we deployed classifier agents that support missing values. The resulting First Union classifiers were capable of ignoring the real values provided by the Chase data, and the Chase classifiers were able to deal with the situation and rely on the other attributes for the predictions. For the **second** incompatibility, we applied the technique about semantically different attributes described in section 4.3. In other words, we had the values of the First Union data translated to the semantics of the Chase data.

To test these approaches, we divided each database into 12 distinct subsets and we applied the five machine learning programs on each of these subsets. The result was 60 Chase base-classifiers and 60 First Union base-classifiers. Then, we run each Chase classifier against each First Union subset and each First Union classifier against each Chase subset. The averaged performance results of a sample of these classifiers (one classifier per learning algorithm) are presented in figure 11. The graph on the left compares the TP and FP rates of the Chase classifiers on the Chase data and the First Union data, whereas the graph on the right compares the TP and FP rates of the First Union classifiers on the First Union and the Chase data.

As expected, the performance of the Chase classifiers on First Union data was inferior to the performance of the same classifiers on the Chase data. After all, the Chase classifiers were trained over data sets with different characteristics, patterns and fraud distribution. In addition, they were not given the chance to use the predictiveness of the fields missing from the First Union data. Nevertheless, the Chase classifiers can still catch a decent portion of the fraudulent transactions of the first Union credit card data. Similarly, the performance of the First Union classifiers over the Chase data was not analogous to their performance over the First Union data. The purpose of these tests, was not to compare the Chase classifiers head to head with the First Union classifiers, but to establish the feasibility of overcoming the different schema integration problem. The benefits of importing classifiers from different sources would, presumably, manifest after the integration of local and remote classifiers. The remote classifiers (in this case from Chase/First Union), may be able detect patterns of fraud that that would otherwise leak through the local fraud detectors (in

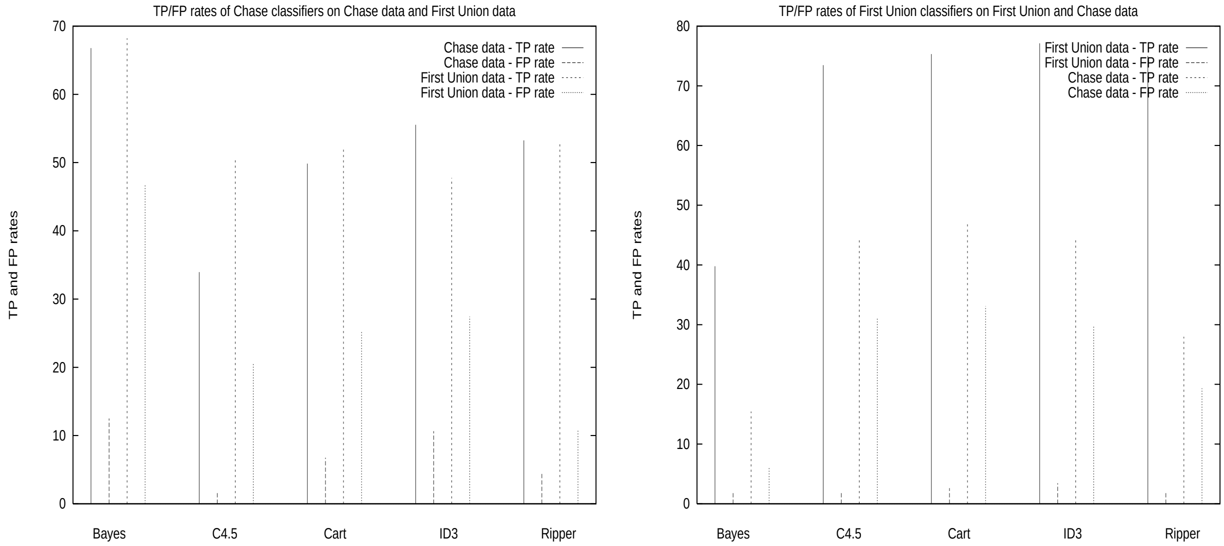

Figure 11: Left: TP and FP rates for the Chase classifiers on the Chase and First Union data. Right: TP and FP rates for the First Union classifiers on the First Union and Chase data.

this case First Union/Chase).

The settlement of the incompatible database schemas 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 incorporating more diverse models.

6 Current Status and Research Plans

A considerable amount of work has already been completed:

- A prototype (design and implementation) of the *JAM* system with the basic infrastructure.
- Published papers on *JAM* with experiments on credit card data. The paper "JAM: Java Agents for Meta-Learning over distributed databases" was voted runner up best application paper in KDD'97
- Implementation of pre-training pruning and post-training pruning methods.
- Experiments on the pre-training pruning methods over the Chase and First Union data sets.
- Experiments on the post-training pruning methods over the SS and SJ data sets.
- Initial experiments on the bridging methods over Chase and First Union data sets

We plan to graduate next year; until then much remains to be accomplished:

- Eliminate the bottleneck of the centralized Configuration Manager by reducing it to a simple name server and by extending the Datasites first with the functionality to control and maintain the configuration of the system in a distributed fashion and second with the capability to collaborate with the Datasites of their own choice.

- Experiments and evaluation of additional pre-training pruning methods.
- Experiments and evaluation of additional post-training pruning methods.
- Experiments and evaluation of the bridging methods. Use of classification and regression methods to compute the missing attributes.
- Experiments and evaluation of the adaptive method. The two credit card data sets are suitable for this task; they consist of sampled transactions from different, yet overlapping periods.
- Measurements of the overheads introduced by the pre- and post-training pruning stages and comparison of the performance of the data mining system with and without these stages. Despite the fact that the two methods add complexity in the meta-learning process, their contribution in the efficiency of the meta-learning system is expected to be positive.
- Measurements of the overheads incurred by the distributed protocol in each Datasite with respect to the number of the Datasites collaborating.
- Write the thesis and graduate.

7 Summary

Data mining systems aim to discover patterns and extract useful information from facts recorded in databases. A widely accepted approach to this objective is to apply to these databases various machine learning programs that discover patterns that may be exhibited in the data and compute descriptive representations of the data, called models or classifiers. In this thesis proposal, we concentrated on the problem of acquiring useful information, efficiently and accurately, from large and distributed databases. In this respect, we proposed the *JAM* system, a powerful, distributed agent-based meta-learning system for large scale data mining applications. *Meta-learning* is a general method, that facilitates the combining of results obtained independently by the various machine learning programs and supports the scaling of large data mining applications.

In the course of the design and implementation of *JAM* we encountered several issues related to scalability, efficiency, longevity and compatibility of distributed data mining systems. Efficiency and scalability were addressed first by employing distributed and asynchronous protocols at the architectural level for managing the learning agents across the data sites of the system, and second by introducing special selection algorithms at the data site level to evaluate, keep and use only the most essential (base/meta) classifiers. Longevity (or adaptivity) was achieved by extending the meta-learning principles to combine both older and newer classifiers while compatibility was tackled by deploying auxiliary agents that resolved the differences.

A prototype of the *JAM* system with the basic infrastructure is complete. In this proposal we presented the architecture of the system and the various components and protocols. Furthermore, we described several experiments that measured the usefulness and effectiveness of the *pre-* and *post- training pruning* methods and the soundness of the techniques bridging the differences between incompatible classifiers. The experiments suggest that pre-training pruning can achieve similar or better performance results than the brute-force assembled meta-classifier at a much more cost effective way, and that post-training pruning can detect the meta-classifiers with weaknesses without the need of human intervention and supervision. Last but not least, we demonstrated that certain *bridging* agents with learning or pre-processing capabilities can alleviate the differences among database with different schemas, and extend the “visibility” of data mining systems.

The design and implementation of useful and practical distributed data mining systems requires extensive research on all these issues. This area is open and active. These problems have not been fully explored yet and our first results suggest that we have the potential to contribute useful results with broad applicability.

References

[1] J. Fürnkranz abd G. Widmer. Incremental reduced error pruning. In *Proc. 11th Intl. Conf. Mach. Learning*. Morgan Kaufmann, 1994.

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

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

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

[5] A. W. Moore C. G. Atkeson, S. A. Schaal. Locally weighted learning. AI Review, To Appear, 1997.

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

[7] P. Chan and S. Stolfo. Experiments on multistrategy learning by meta-learning. In *Proc. Second Intl. Conf. Information and Knowledge Management*, pages 314–323, 1993.

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

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

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

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

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

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

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

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

[16] C. Elkan. Boosting and naive bayesian learning [http://www-cse.ucsd.edu/~elkan/papers/bnb.ps]. Department of Computer Science and Engineering, Univ. of California, San Diego, CA, 1997.

[17] E.R.Carson and U.Fischer. Models and computers in diabetes research and diabetes care. *Computer methods and programs in biomedicine, special issue*, 32, 1990.

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

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

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

[21] K.Lang. News weeder: Learning to filter net news. In A.Prieditis and S.Russel, editors, *Proc. 12th Intl. Conf. Machine Learning*, pages 331–339. Morgan Kaufmann, 1995.

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

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

[24] Wenke Lee, Naser S. Barghouti, and John Moccenigo. Grappa: Graph package in java. In *Graph Drawing, Rome, Italy*, Rome, Italy, September 1997.

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

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

[27] C. Merz and P. Murphy. 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, 1996.

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

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

[30] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

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

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

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

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

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

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

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

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

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

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

[42] S. Stolfo, W.D. Fan, A. Prodromidis W.Lee, S. Tselepis, and P. K. Chan. Agent-based fraud and intrusion detection in financial information systems. Submitted to IEEE Symposium on Security and Privacy, 1998.

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

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

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