Meta-learning in distributed data mining systems: Issues and approaches

Andreas L. Prodromidis[*]
Computer Science Department
Columbia University
New York, NY 10027
andreas@cs.columbia.edu

Philip K. Chan
Computer Science Department
Florida Institute of Technology
Melbourne, FL 32901
pkc@cs.fit.edu

Salvatore J. Stolfo
Computer Science Department
Columbia University
New York, NY 10027
sal@cs.columbia.edu

Abstract

Data mining systems aim to discover patterns and extract useful information from facts recorded in databases. A widely adopted approach to this objective is to apply various machine learning algorithms to compute descriptive models of the available data. Here, we explore one of the main challenges in this research area, the development of techniques that scale up to large and possibly physically distributed databases.

Meta-learning is a technique that seeks to compute higher-level classifiers (or classification models), called meta-classifiers, that integrate in some principled fashion multiple classifiers computed separately over different databases. This study, describes meta-learning and presents the JAM system (Java Agents for Meta-learning), an agent-based meta-learning system for large-scale data mining applications. Specifically, it identifies and addresses several important desiderata for distributed data mining systems that stem from their additional complexity compared to centralized or host-based systems. Distributed systems may need to deal with heterogenous platforms, with multiple databases and (possibly) different schemas, with the design and implementation of scalable and effective protocols for communicating among the data sites, and the selective and efficient use of the information that is gathered from other peer data sites. Other important problems, intrinsic within

[*]Supported in part by an IBM fellowship.

data mining systems that must not be ignored, include, first, the ability to take advantage of newly acquired information that was not previously available when models were computed and combine it with existing models, and second, the flexibility to incorporate new machine learning methods and data mining technologies. We explore these issues within the context of JAM and evaluate various proposed solutions through extensive empirical studies.

1 Introduction

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

- NASA's Earth Observing System (EOS) of orbiting satellites and other spaceborne instruments send one terabyte of data to receiving stations every day [73].
- The world wide web is estimated [2] to have at least 450,000 hosts, one million sites and as many as eighty million pages (as of March 1998).
- By the year 2000 a typical Fortune 500 company is projected to possess more than 400 trillion characters in their electronic databases requiring 400 terabytes of mass storage.

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

The relatively new field of *Knowledge Discovery and Data Mining* (KDD) has emerged to compensate for these deficiencies. *Knowledge discovery* in databases denotes the complex process of identifying valid, novel, potentially useful and ultimately understandable patterns in data [23]. *Data mining* refers to a particular step in the KDD process. According to the most recent and broad definition [23], "data mining consists of particular algorithms (methods) that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (models) over the data."

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

The field of machine learning has made substantial progress over the last few decades and numerous algorithms, ranging from those based on stochastic models to those based on purely symbolic representations like rules and decision trees, have already been developed and applied to many problems in diverse areas. Over the past decade, machine learning has evolved from a field of laboratory demonstrations to a field of significant commercial value [45]. Machine-learning algorithms have been deployed in heart disease diagnosis [61], in predicting glucose levels for diabetic patients [22], in detecting credit card fraud [65], in steering vehicles driving autonomously on public highways at 70 miles an hour [50], in predicting stock option pricing [46] and in computing customized electronic newspapers[33], to name a few applications. Many large business institutions and market analysis firms attempt to distinguish the low-risk (high profit) potential customers by learning simple categorical classifications of their potential customer base. Similarly, defense and intelligence operations utilize similar methodologies on vast information sources to predict a wide range of conditions in various contexts. Recently, for example, data mining techniques have been successfully applied to intrusion detection in network-based systems [72].

One of the main challenges in machine learning and data mining is the development of inductive learning techniques that scale up to large and possibly physically distributed data sets. Many organizations seeking added value from their data are already dealing with overwhelming amounts of information. The number and size of their databases and data warehouses grows at phenomenal rates, faster than the corresponding improvements in machine resources and inductive learning techniques. Most of the current generation of learning algorithms are computationally complex and require all data to be resident in main memory which is clearly untenable for many realistic problems and databases. Notable exceptions include IBM's SPRINT [63] and SLIQ [39] decision tree-based algorithms and Provost's and Hennessy's DRL rule-based algorithm for multi-processor learning.

Furthermore, in certain cases, data may be inherently distributed and cannot be localized on any one machine (even by a trusted third party) for a variety of practical reasons including security and fault tolerant distribution of data and services, competitive (business) reasons, statutory constraints imposed by law as well as physically dispersed databases or mobile platforms like an armada of ships. In such situations, it may not be possible, nor feasible, to inspect all of the data at one processing site to compute one primary "global" classifier.

Meta-learning is a technique recently developed that deals with the problem of computing a "global" classifier from large and inherently distributed databases. Meta-learning aims to compute a number of independent classifiers (concepts or models) by applying learning programs to a collection of independent and inherently distributed databases in parallel. The "base classifiers" computed are then collected and combined by another learning process. Here meta-learning seeks to compute a "meta-classifier" that integrates in some principled fashion the separately learned classifiers to boost overall predictive accuracy.

Our *main objective* is to take advantage of the inherent parallelism and distributed nature of meta-learning and design and implement a powerful and practical distributed data mining system. Assuming that a system consists of several databases interconnected through an intranet or internet, the goal is to provide the means for each data site to utilize its own local data and, at the same time, benefit from the data that is available at other data sites without transferring or directly accessing that data. In this context, this can be materialized by learning agents that execute at remote data sites and generate classifier agents that can subsequently be transfered among the sites. We have achieved this goal through the implementation and demonstration of a system we call JAM (Java Agents for Meta-Learning). To our knowledge, JAM is the first system to date that employs meta-learning as a means to mine distributed databases. (A commercial system based upon JAM has recently appeared [26].)

JAM, however, is more than an implementation of a distributed meta-learning system. It is a distributed data mining system addressing many practical problems for which centralized or host-based systems are not appropriate. On the other hand, distributed systems have increased complexity. Their practical value depends on the *scalability* of the distributed protocols as the number of the data sites and the size of the databases increases, and on the *efficiency* of their methods to use the system resources effectively. Furthermore, distributed systems may need to run across heterogenous platforms (*portability*) or operate over databases that may (possibly) have different schemas (*compatibility*). There are other important problems, intrinsic within data mining systems that should not be ignored. Data mining systems should be *adaptive* to environment changes (e.g. when data and objectives change over time), *extensible* to support new and more advanced data mining technologies and last but not least, highly *effective*. The intent of this study is to identify and describe each of these issues separately and to present an overview of existing approaches — detailed discussions of techniques and findings appear in publications cited throughout this study.

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

The rest of this paper describes the JAM system, views the scalability, efficiency, portability, compatibility, adaptivity, extensibility and effectiveness desiderata as an integral part of the design and implementation of JAM, and presents the efficacy of our approaches in predicting fraudulent credit card transactions as a case study. In Section 2 we introduce the meta-learning process. Section 3 presents the distributed architecture of JAM, together with our proposed approaches to scalability and efficiency. Section 4 addresses the portability issues, and Section 5 describes our methods for overcoming the obstacles posed by databases with schema differences. In Sections 6 and 7, we detail our techniques for incorporating newly computed models and for extending JAM with new machine learning technologies and in Section 8 we elaborate on the effectiveness of data mining systems. Section 9 presents our results in the credit card fraud detection task and finally, Section 10, concludes the paper.

2 Inductive learning and Meta Learning

Inductive learning (or *learning from examples* [42]) is the task of identifying regularities in some given set of training examples with little or no knowledge about the domain from which the examples are drawn. Given a set of training examples, i.e. $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$, for some unknown function $y = f(\mathbf{x})$, with each $\mathbf{x_i}$ interpreted as a set of attribute (feature) vectors x_i of the form $\{x_{i1}, x_{i2}, ..., x_{ik}\}$ and with each y_i representing the class label associated with each vector ($y_i \in \{y_1, y_2, ..., y_m\}$), the task is to compute a *classifier* or *model* $\hat{f}$ that approximates f and correctly labels any feature vector drawn from the same source as the training set. It is common to call the body of knowledge that classifies data with the label y as the *concept* of *class* y.

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 unseen 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 unseen records (examples). Decision trees are used in ID3 [58], 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 [13] are if-then expressions, where the antecedent is a pattern expression and the consequent is a class label. Each version space learned in VS [44] 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 [36]. The learned distance functions in exemplar-based learning algorithms (or nearest neighbor algorithms) define a similarity or "closeness" measure between two instances [64]. In genetic algorithms, hypotheses are usually represented by application-specific bit strings. These algorithms search for the most appropriate hypotheses [29, 17] by simulating evolution, i.e. they generate successor hypotheses by repeatedly mutating

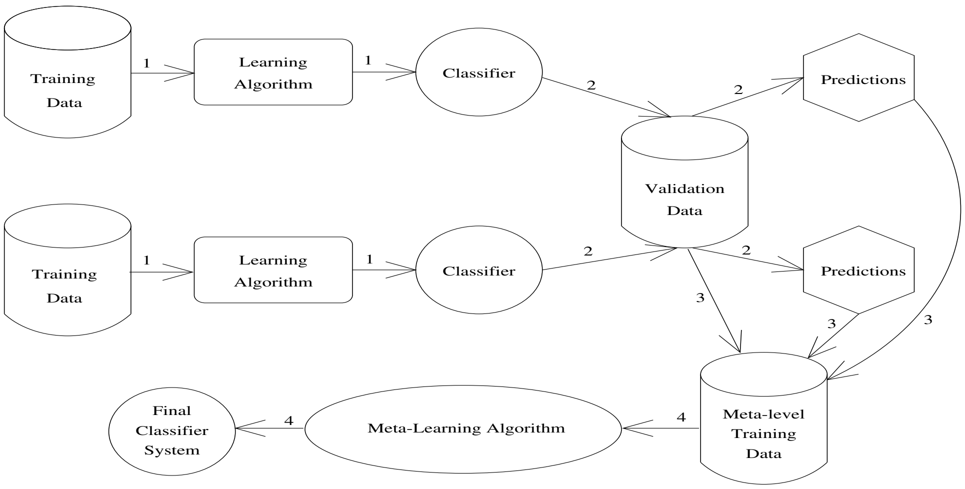

Figure 1: Meta-learning.

and recombining (crossover) parts of the best currently known hypothesis [28, 18]. 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 [12]. Implicit decision rules classify according to maximal probabilities.

Meta-learning [7] is loosely defined as learning from learned knowledge. In this case, we concentrate on learning from the output of concept learning systems. This is achieved by learning from the *predictions* of these classifiers on a common *validation data set*. Thus, we are interested in the output of the classifiers, not the internal structure and strategies of the learning algorithms themselves. Moreover, in some of the schemes defined, the data presented to the learning algorithms may also be available to the *meta-learner*.

Figure 1 depicts the different stages in a simplified meta-learning scenario:

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

2. Predictions are generated by the learned classifiers on a separate validation set.

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

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

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

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

2.1 Voting, Arbitrating and Combining

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

We distinguish between *base classifiers* and *combiners/arbiters* as follows. A base classifier is the outcome of applying a learning algorithm directly to "raw" training data. The base classifier is a program that given a test datum provides a prediction of its unknown class. A *combiner* or *arbiter*, is a program generated by a learning algorithm that is trained on the predictions produced by a set of base classifiers on raw data. The arbiter/combiner is also a classifier, and hence other arbiters or combiners can be computed from the set of predictions of other combiners/arbiters in a hierarchical manner.

Briefly, an arbiter [8] 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 2, depicts how the final prediction is made with input predictions from two base classifiers and a single arbiter. The rest of this paper, concentrates on the combiner strategy.

Before we detail the combiner strategy, for concreteness, we define the following notation. Let $\mathbf{x}$ be an instance whose classification we seek, and $C_1(\mathbf{x})$, $C_2(\mathbf{x})$,..., $C_k(\mathbf{x})$ be the predicted classifications of $\mathbf{x}$ from the k base classifiers,

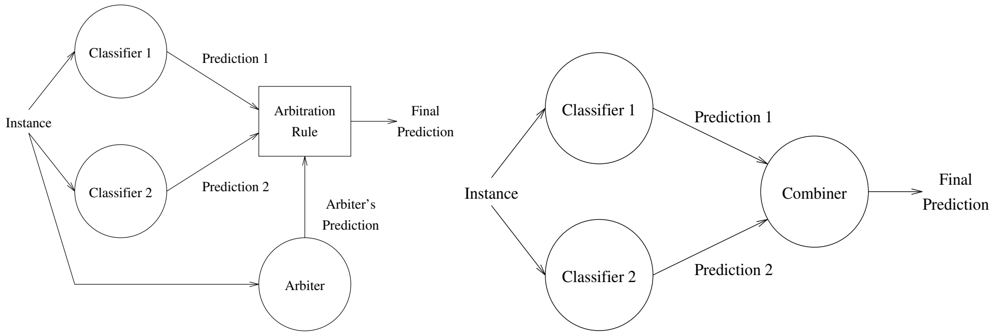

Figure 2: Left: An arbiter with two classifiers. Right: A combiner with two classifiers

C_i, $i = 1, 2, ..., k$. $class(\mathbf{x})$ and $attrvec(\mathbf{x})$ denote the correct classification and attribute vector of example $\mathbf{x}$, respectively.

2.2 Combiner strategy

In the combiner 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 training examples for the meta-learner. From these examples, the meta-learner generates a meta-classifier, that we call a *combiner*. In classifying an instance, the base classifiers first generate their predictions. Based on the same composition rule, a new instance is generated from the predictions, which is then classified by the combiner (see right diagram of Figure 2). The aim of this strategy is to "coalesce" the predictions from the base classifiers by learning the relationship or correlation between these predictions and the correct prediction. A combiner computes a prediction that may be entirely different from any proposed by a base classifier, whereas an arbiter chooses one of the predictions from the base classifiers and the arbiter itself.

We experimented with three schemes for the composition rule (more details are in [6]. First, the predictions, $C_1(\mathbf{x})$, $C_2(\mathbf{x})$, ... $C_k(\mathbf{x})$, for each example $\mathbf{x}$ in the validation set of examples, E, are generated by the k base classifiers. These predicted classifications are used to form a new set of "meta-level training instances," T, which is used as input to a learning algorithm that computes a combiner. The manner in which T is computed varies as defined below:

class-combiner The meta-level training instances consist of the correct classification and the predictions; i.e., $T = \{(class(\mathbf{x}), C_1(\mathbf{x}), C_2(\mathbf{x}), ...C_k(\mathbf{x})) \mid \mathbf{x} \in$

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

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

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

Figure 3: Sample training sets generated by the class-combiner and class-attribute-combiner schemes with two base classifiers.

$E\}$. This "stacking" scheme was also proposed by Wolpert [74]. See Figure 3 for a sample training set.

class-attribute-combiner The meta-level training instances are formed as in class-combiner with the addition of the attribute vectors; i.e., $T = \{(class(\mathbf{x}), C_1(\mathbf{x}), C_2(\mathbf{x}), ...C_k(\mathbf{x}), attrvec(\mathbf{x})) \mid \mathbf{x} \in E\}$. See Figure 3 for a sample training set.

binary-class-combiner The meta-level training instances are composed in a manner similar to that in the class-combiner scheme except that each prediction, $C_i(\mathbf{x})$, has l binary predictions, $C_{i_1}(\mathbf{x}), \ldots, C_{i_l}(\mathbf{x})$, where l is the number of classes. Each prediction, $C_{i_j}(\mathbf{x})$, is produced from a binary classifier, which is trained on examples that are labeled with classes j and $\neg j$. In other words, we are using more specialized base classifiers and attempting to learn the correlation between the binary predictions and the correct prediction. For concreteness, $T = \{(class(\mathbf{x}), C_{1_1}(\mathbf{x}), \ldots, C_{1_l}(\mathbf{x}), C_{2_1}(\mathbf{x}), \ldots, C_{2_l}(\mathbf{x}), \ldots C_{k_1}(\mathbf{x}), \ldots, C_{k_l}(\mathbf{x})) \mid \mathbf{x} \in E\}$. See Figure 4 for a sample training set.

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

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

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

Figure 4: Sample training set generated by the binary-class-combiner scheme with two base classifiers.

2.3 Benefits of Meta-learning

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 predictive performance by combining different learning systems each having different *inductive bias* (e.g representation, search heuristics, search space) [44]. 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. Furthermore, meta-learning constitutes a scalable machine learning method since it can be generalized to hierarchical multi-level meta-learning.

Most of the methods reported in the Machine Learning and KDD literature that compute, evaluate and combine ensembles of classifiers [19] can be considered as weighted voting among hypothesis (models). Furthermore, most of these algorithms, generate their classifiers by applying the same learning algorithms on variants of the same data set [3, 24, 30, 32, 34, 48, 49, 62, 68]. Breiman [4] and LeBlanc and Tibshirani [35], for example, acknowledge the value of using multiple predictive models to increase accuracy, but they rely on cross-validation data and analytical methods, (e.g. least squares regression), to compute the best linear combination of the available hypothesis. Instead, the combiner methods apply arbitrary learning algorithms to discover the correlations among the available models and compute non-linear relations among the classifiers (at the expense, perhaps, of generating less intuitive representations).

Other methods for combining multiple models, include Merz and Pazzani's PCR^* [41] and Merz's SCANN [40] algorithms. The first integrates ensembles of regression models for improving regression estimates while the latter for improving classification performance. Both rely on methods similar to principal components analysis to map the estimates of the models into a new representation space upon which they compute a higher level model. PCR^* and SCANN are sophisticated and effective combining algorithms, but too computationally expensive to combine models within domains with many models and large data sets. SCANN, in fact, is cubic in the number of available models.

Meta-learning is particularly suitable for distributed data mining applications, such as fraud detection in financial information systems. Financial institutions today typically develop custom fraud detection systems targeted to their own asset bases. Recently though, banks have come to search for unified and global approaches that would also involve the periodic sharing with each other of information about attacks.

The key difficulties in this approach are: financial companies avoid sharing their data for a number of (competitive and legal) reasons; the databases that companies maintain on transaction behavior are huge and growing rapidly; real-time analysis is highly desirable to update models when new events are detected and easy distribution of models in a networked environment is essential to maintain up to date detection capability. Meta-learning is a general strategy that provides the means of learning how to combine and integrate a number of classifiers or models learned separately at different financial institutions. JAM allows financial institutions to share their models of fraudulent transactions that each computes separately, while not disclosing their own proprietary data.

Next, we describe how meta-learning is incorporated in JAM. We detail the fundamental issues of scalability, efficiency, portability, compatibility, adaptivity, extensibility and effectiveness of distributed data mining systems. We overview our solutions to these issues within the JAM framework, and provide empirical evidence that JAM constitutes an effective system as exemplified by the credit card fraud detection domain.

3 Scalability and efficiency

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

3.1 System architecture level

First we focus on the 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, we designed a protocol that allows the data sites to collaborate efficiently without hindering their progress. JAM is architected as a distributed computing construct that supports the launching of learning and meta-learning agents to distributed database sites.

First, local or imported *learning agents* execute on the local database to compute the data site's local classifiers. Then, each data site may import (remote) classifiers from its peer data sites and combine these with its own local classifier using *meta-learning agents*. Again, the meta-learning agents can be either local or imported from other data sites. Finally, once the base and meta-classifiers are computed, the JAM system manages the execution of these modules to classify data sets of interest. These actions may take place at all data sites simultaneously and independently. The JAM system can thus be viewed as a coarse-grain parallel application where the constituent sites function autonomously and (occasionally) exchange classifiers with each other.

The configuration of the distributed system is maintained by the Configuration Manager (CM), an independent server, much like a simple name server, that is responsible for keeping the state of the system up-to-date. The logical architecture of the JAM meta-learning system is presented in Figure 5. In this example, three JAM sites Orange, Mango and Strawberry exchange their base classifiers to share their local view of the learning task. The user of the data site controls the learning task by setting the parameters of the user configuration file, e.g. the algorithms to

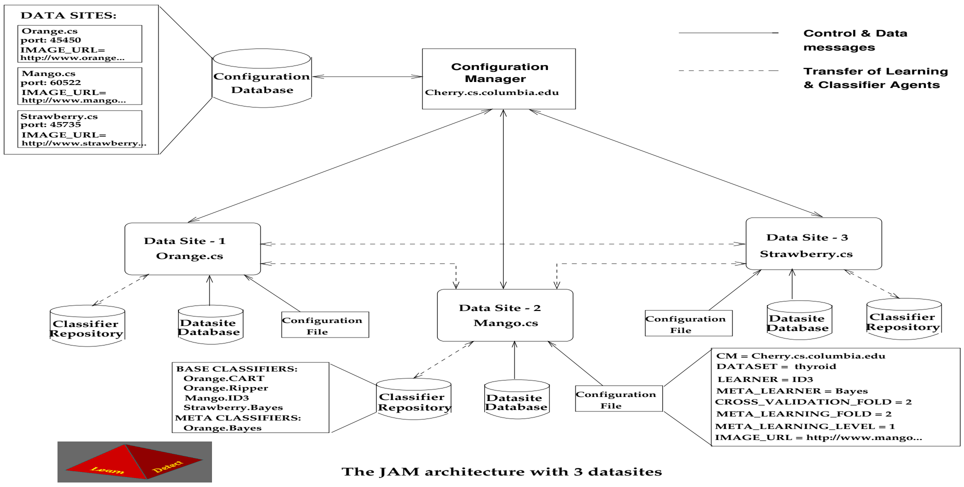

Figure 5: The architecture of the meta-learning system.

be used, the database to learn, the images to be used by the animation facility, the folding parameters, etc. In this example, the CM runs on Cherry and the Mango site ends up with four base classifiers (one local plus the three imported classifiers) and a single Bayesian meta-classifier (imported from Orange).

JAM is designed with asynchronous, distributed communication protocols that enable the participating database sites to operate independently and collaborate with other peer sites as necessary, thus eliminating centralized control and synchronization points. Each JAM site is organized as a layered collection of software components shown in Figure 6. In general, the system can be decomposed into four separate subsystems, the User Interface, the JAM Engine and the Client and Server subsystems. The User Interface (upper tier) materializes the front end of the system, through which the owner can define the data mining task and drive the JAM Engine. The JAM Engine constitutes the heart of each JAM site by managing and evaluating the local agents, by preparing/processing the local data sets and by interacting with the Database Management System (DBMS), if one exists. Finally, the Client and Server subsystems compose the network component of JAM and are responsible for interfacing with other JAM sites to coordinate the transport of their agents. Each site is developed on top of the JVM (Java Virtual Machine), with the possible exception of some agents that may be used in a native form and/or depend on an underlying DBMS. A Java agent, for instance, may be able to access a DBMS though JDBC (Java Database Connectivity). The RMI registry component displayed in Figure 6 corresponds to an independent Java process that is used indirectly by the JAM server component. For the interested reader, the JAM system is

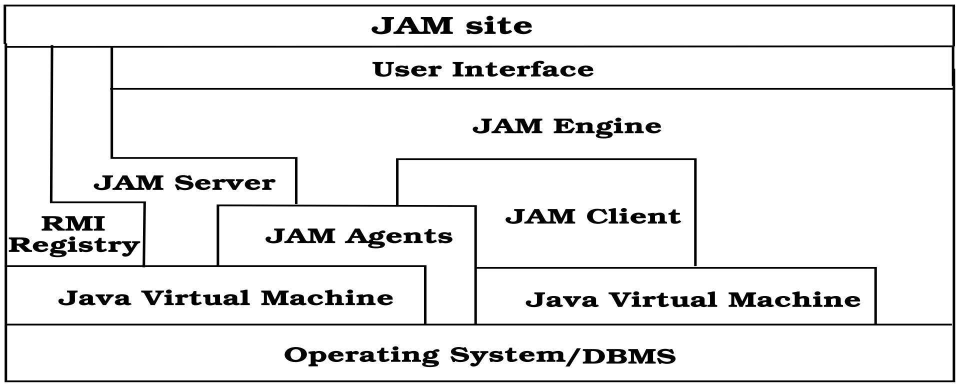

Figure 6: JAM site layered model.

described in detail in [67] and [51].

3.2 Meta-learning level

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

Meta-classifiers can be defined recursively as collections of classifiers structured in multi-level trees [9]. Such structures, however, can be unnecessarily complex, meaning that many classifiers may be redundant, wasting resources and reducing system throughput. (Throughput here denotes the rate at which a stream of data items can be piped through and labeled by a meta-classifier.) We study the efficiency of meta-classifiers by investigating the effects of pruning (discarding certain base classifiers) on their performance. Determining the optimal set of classifiers for meta-learning is a combinatorial problem. Hence, the objective of pruning is to utilize heuristic methods to search for partially grown meta-classifiers (meta-classifiers with pruned subtrees) that are more efficient and scalable and at the same time achieve comparable or better predictive performance results than fully grown (unpruned) meta-classifiers. To this end, we introduced two stages for pruning meta-classifiers, the *a priori* pruning or *pre-training* pruning and the *a posteriori* pruning or *post-training* pruning stages. Both levels are essential and complementary to each other with respect to the improvement of accuracy and efficiency of the system.

A priori pruning or pre-training pruning refers to the filtering of the classifiers before they are combined. Instead of combining classifiers in a brute force manner, with pre-training pruning we introduce a preliminary stage for analyzing the available classifiers and qualifying them for inclusion in a combined meta-classifier. Only those classifiers that appear (according to one or more pre-defined metrics) to be most “promising” participate in the final meta-classifier. Here, we adopt a “black-box” approach which evaluates the set of classifiers based only on their input and output behavior, not their internal structure. Conversely, a posteriori pruning or post-training pruning, denotes the evaluation and pruning of constituent base classifiers after a complete meta-classifier has been constructed.

We have implemented and experimented with three pre-training pruning and two post-training pruning algorithms each with different search heuristics. The first pre-training pruning algorithm is a metric-based algorithm, i.e. it ranks and selects the best k classifiers based on their individual performance on a separate validation set or via cross-validation. The second algorithm is a diversity-based algorithm that has preference towards classifiers with diverse predictive behavior.[1] Finally, the third pre-training pruning algorithm concentrates on sets of specialized classifiers (i.e. classifiers that are good in predicting specific classes) that achieve high coverage. [2] The pre-training pruning algorithms are described in detail in [53]. The post-training pruning algorithms are based, the first on the mapping of the unpruned meta-classifiers as decision trees and their subsequent pruning (the nodes of such a decision tree represent base classifier; pruning a node corresponds to the pruning of a base classifier), and the second on the removal of the base classifiers that are least correlated (and hence the least trusted) to the unpruned meta-classifier. Both post-training pruning algorithms are detailed in [54].

There are two primary objectives for the pruning techniques:

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

2. to generate effective and efficient meta-classifiers.

These pre-training pruning techniques preceed the meta-learning phase and, as such, can be used in conjunction with most of the combining techniques examined in Section 2 (e.g. SCANN). Other methods for evaluating and pruning ensembles of classifiers have been studied by Provost and Fawcett and by Margineantu and Dietterich. Provost and Fawcett [55, 56] introduced the ROC convex hull method

[1] In general, the more diverse a set of classifiers is, the more room for improvement the meta-learning method has.

[2] Coverage denotes the fraction of instances of a validation set where at least one classifier makes a correct prediction.

for its intuitiveness and flexibility. The method evaluates and selects classification models by mapping them onto a True Positive/False Positive plane and by allowing comparisons under different metrics (TP/FP rates, accuracy, cost, etc.). The intent of these algorithms, however, is to select the best classifier (not a group of classifiers) under a specific performance criterion (which could be adjusted in the ROC space). In this work, the focus is on methods with the potential to form effective ensembles of classifiers [19, 27]. In fact, the performance of sub-optimal yet diverse models can be substantially improved when combined together and even surpass that of the best single model.

Margineantu and Dietterich [38] studied the problem of pruning the ensemble of classifiers (i.e. the set of hypothesis (classifiers)) obtained by the boosting algorithm ADABOOST [25]. According to their findings, by examining the diversity and accuracy of the available classifiers, it is possible for a subset of classifiers to achieve similar levels of performance as the entire set. Their research however, was restricted to computing all classifiers by applying the same learning algorithm on many different subsets of the same training set. In JAM we consider the more general setting where ensembles of classifiers can be obtained by applying, possibly, different learning algorithms over (possibly) distinct databases. Furthermore, instead of voting (ADABOOST) over the predictions of classifiers for the final classification, we adopt meta-learning as a more general framework for combining predictions of the individual classifiers.

Our pre-training and post-training pruning methods have been tested in the credit card fraud detection domain. The results are presented in Section 9.

4 Portability

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

The JAM system presented in this paper, is a distributed computing construct designed to extend the OS environments to accommodate such requirements. As implied by its name (Java Agents for Meta-learning), portability is inherent within JAM. The "Java" part denotes that we have used Java technology to build the composing parts of the system including the underlying infrastructure, the specific operators that generate and spawn agents, the graphical user interface, the animation facilities to monitor agent exchanges and the meta-learning process and the learning and classifier agents. The "meta-learning" term refers to the system's methods for combining classifier agents. It constitutes a unifying machine learning approach that can be applied to large amounts of data in wide area computing networks for a range of different applications. It has the advantage of being 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.

The learning agents are the basic components for searching for patterns within the data and the classifier agents are the units that capture the computed models and can be shared among the data sites. The platform independence of Java makes it easy for each JAM site to delegate its agents to any participating site. As a result, JAM has been successfully tested on the most popular platforms including Solaris, Windows and Linux simultaneously, i.e. JAM sites imported and utilized classifiers that were computed over different platforms.

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

5 Compatibility

Combining multiple models has been receiving increased attention in the literature [19, 11]. In much of the prior work on combining multiple models, it is assumed that all models originate from different subsets (not necessarily distinct) of a single data set as a means to increase accuracy, (e.g. by imposing probability distributions over the instances of the training set, or by stratified sampling, sub-sampling, etc.) and not as a means to integrate distributed information. Although the JAM system, as described in Section 3 addresses the later by employing meta-learning techniques, integrating classification models derived from distinct and distributed databases may not always be feasible.

In all cases considered so far, all classification models are assumed to originate from databases of identical schemas. Since classifiers depend directly on the format of the underlying data, minor differences in the schemas between databases derive incompatible classifiers, i.e. a classifier cannot be applied on data of different formats. Yet these classifiers may target the same concept. We seek to bridge these disparate classifiers in some principled fashion.

Assume, for instance, we acquire two data sets of credit card transactions (labeled fraud or legitimate) from two different financial institutions (i.e. banks). The learning problem is to distinguish legitimate from fraudulent use of a credit card.

Both institutions seek to be able to exchange their classifiers and hence incorporate in their system useful information that would otherwise be inaccessible to both. Indeed, for each credit card transaction, both institutions record similar information, however, they also include specific fields containing important information that each has acquired separately and which provides predictive value in determining fraudulent transaction patterns. In a different scenario where databases and schemas evolve over time, it may be desirable for a single institution to be able to combine classifiers from both past accumulated data with newly acquired data. To facilitate the exchange of knowledge and take advantage of incompatible and otherwise useless classifiers, we need to devise methods that "bridge" the differences imposed by the different schemas.

Integrating the information captured by such classifiers is a non-trivial problem that we have come to call, *"the incompatible schema"* problem. (The reader is advised not to confuse this with Schema Integration over Federated/Mediated Databases.)

5.1 Database Compatibility

The "incompatible schema" problem impedes JAM from taking advantage of all available databases. 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:

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

$$Schema(DB_B) = \{B_1, B_2, ..., B_n, B_{n+1}, C\} \tag{2}$$

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

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

$$Schema(DB_A) = \{A_1, A_2, ..., A_n, A_{n+1}, C\} \tag{3}$$

$$Schema(DB_B) = \{B_1, B_2, ..., B_n, C\} \tag{4}$$

   where we assume that attribute B_{n+1} is not present in DB_B.[3]

[3] The dual problem has DB_B composed with B_{n+1} but A_{n+1} is not available to A.

2. $A_{n+1} \approx B_{n+1}$: The two attributes are of similar type but slightly different semantics that is, there may be a map from the domain of one type to the domain of the other. For example, 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).

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

5.2 Bridging methods

There are several approaches to address the problem depending upon the learning task and the characteristics of the different or missing attribute A_{n+1} of DB_B. The details of these approaches can be found in [52].

- A_{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 [5], locally weighted regression [1], linear regression fit [47], MARS [31]) 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 $A_1, ..., A_n$. Then it can send all its local (base and bridging) classifiers to data site B. At the other side, data site B can deploy the auxiliary classifiers ${C_{Aj}}'$ to estimate the values of the missing attribute A_{n+1} and present to classifiers C_{Aj} a new database ${DB_B}'$ with schema $\{A_1, ..., A_n, \hat{A}_{n+1}\}$. From this point on, meta-learning and meta-classifying proceeds normally.

- A_{n+1} **is missing and cannot be predicted:** Computing a model for a missing attribute assumes a correlation between that attribute and the rest. Nevertheless, such a hypothesis may be unwarranted in which case we adopt one of the following strategies:

– **Classifier agent C_{Aj} supports missing values:** If the classifier agent C_{Aj} originating from DB_A can handle attributes with missing values, data site B can simply include null values in a fictitious A_{n+1} attribute added to DB_B. The resulting $DB_B{}'$ database is a database compatible with the C_{Aj} classifiers. Different classifier agents treat missing values in different ways. Some machine learning algorithms, for instance, treat them as a separate category, others replace them with the average or most frequent value, while more 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:

$$DB_A{}' = PROJECT\ (A_1, ..., A_n)\ FROM\ DB_A \tag{5}$$

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

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

These approaches address the "incompatible schema" problem and meta-learning over these models should proceed in a straightforward manner. The idea of requesting missing definitions from remote sites (i.e. missing attributes) first appeared in [37]. In that paper, Maitan, Raś and Zemankova define a query language and describe a scheme for handling and processing global queries (queries that need to access multiple databases at more than one site) within distributed information systems. According to this scheme, each site compiles into rules some facts describing the data that belong to other neighboring sites which can subsequently by used to interpret and correctly resolve any non-standard queries posed (i.e. queries with unknown attributes).

More recently Zbigniew Raś in [60] further elaborated this scheme and developed an algebraic theory to formally describe a query answering system for solving non-standard DNF queries in a distributed knowledge based system (DKBS). Given a non-standard query on a relational database with categorical or partially ordered set of attributes, his aim is to compute rules consistent with the distributed data to resolve unknown attributes and retrieve all the records of the database that satisfy it. Our approach, however, is more general, in that, it supports both categorical and continuous attributes, and it is not limited to a specific syntactic case or the consistency of the generated rules. Instead, it employs machine learning techniques to compute models for the missing values.

6 Adaptivity

Most data mining systems operate in environments that are almost certainly bound to change, a phenomenon known as *concept drift*. For example, medical science evolves, and with it the types of medication, the dosages and treatments, and of course the data included in the various medical database; lifestyles change over time and so do the profiles of customers included in credit card data; new security systems are introduced and new ways to commit fraud or to break into systems are devised. Most traditional data mining systems are static.

The classifiers deployed in the traditional classification systems are obtained by applying machine learning programs over historical databases DB_i. 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 [69, 70, 71], an incremental version of ID3) or nearest neighbor algorithms. Incremental machine learning programs denote machine learning programs that are not constrained to retain all training examples in main memory; instead they examine one instance at a time and tune the model accordingly. Hence, the classifiers initially trained over DB_i can be updated later by resuming their training on the new database DB_j once it becomes available. On the other hand, these algorithms do not provide a means for removing irrelevant knowledge gathered in the past. Furthermore, updating a model on every instance may not be accurate in a noisy domain. This shortcoming can be avoided by employing incremental batch learning methods [14, 20, 75], i.e. methods that update models using subsets of data. The problem with these approaches is that they are not general enough; instead they rely on specific algorithms, model representation and implementations.

We describe a new possibility, a mechanism that takes advantage of the capabilities and architecture of the JAM system to integrate new information. New information is treated in a fashion similar to the information imported from remote data sites in JAM. Instead of combining classifiers from remote data sites (integration over space), adaptive learning systems combine classifiers acquired over different time periods (integration over time). We employ meta-learning techniques to design learning systems capable of incorporating into their accumulated knowledge (existing classifiers) the new classifiers that capture emerging patterns learned over new data sources.

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

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

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

a straight-forward manner.

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

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

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

5. It can be used in conjunction with existing pruning techniques. Normally, incorporating new classifiers in a meta-classifier hierarchy continuously would eventually result in large and inefficient tree-based 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 new patterns.

This strategy opens a new research direction 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 allows JAM to extend and incorporate new information without discarding or depreciating the knowledge it has accumulated over time from previous data mining.

We have not treated the issue of what to do about previously computed and unused (pruned) classifiers or models. Retaining and managing older unused classifiers or models is an interesting open question that has not been adequately addressed in our work.

7 Extensibility

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

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 [58], Ripper [15], CART [5], Bayes [21], WPEBLS [16], CN2 [13], etc.) are 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 with. 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:

1. A `constructor` with no arguments. JAM can then instantiate the agent, provided it knows its name (which can be supplied by the owner of the data site 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 arguments to the agent. Arguments include the names of the training and test datasets, 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 dataset.

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

The class hierarchy (only methods are shown) for five different learning agents is presented in Figure 7. ID3, Bayes, WPEBLS, CART and Ripper inherit the methods `initialize()` and `getClassifier()` from their parent learning

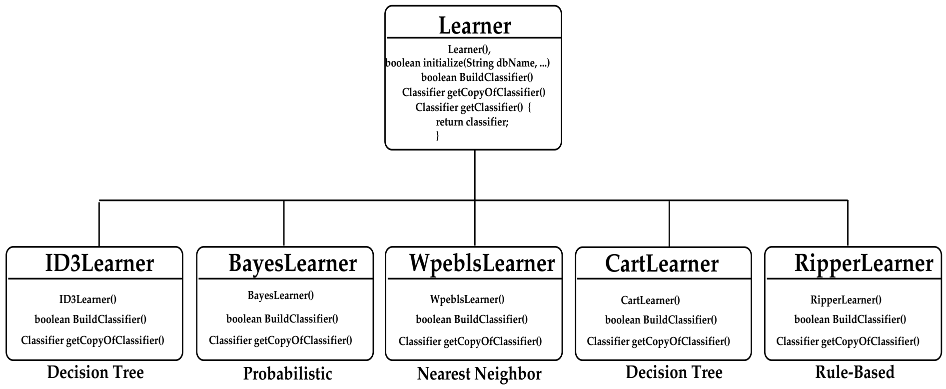

Figure 7: The class hierarchy of learning agents.

agent class. The MetaLearning, Classifier and MetaClassifier classes are defined in similar class hierarchies.

JAM's infrastructure is independent 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 a powerful and extensible data mining facility. In the latest version of JAM, for example, ID3 and CART are full Java agents, whereas Bayes, WPEBLS and Ripper are stored locally as native applications. It is exactly this feature that allows users to employ native programs within the Java agents if computational speed is crucial (see also Section 4).

8 Effectiveness

We seek effective measures to evaluate the predictive accuracy of classification systems. Contrary to most studies on comparing different learning algorithms and classification models, predictive accuracy does not mean only overall accuracy (or minimal error rate). Instead, we must consider alternative and more realistic "optimality criteria". Other measures of interest include True Positive (TP) and False Positive (FP) rates for (binary) classification problems, ROC analysis and problem specific cost models are all different criteria relevant to different problems and learning tasks. A detailed study against the use of accuracy estimation for comparing induction algorithms can be found in [57]. In the credit card fraud domain, for example, overall predictive accuracy is inappropriate as the single measure of predictive performance. If 1% of the transactions are fraudulent, then a model that

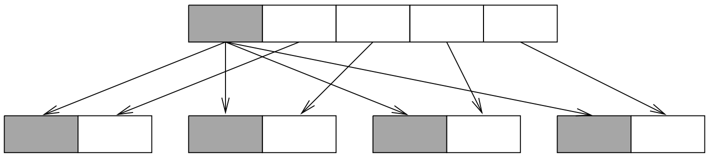

Figure 8: Generating four 50:50 data subsets from a 20:80 data set

always predicts "non-fraud" will be 99% accurate. Hence, TP rate is more important. Of the 1% fraudulent transactions, we wish to compute models that predict 100% of these, yet produce no false alarms (i.e. predict no legitimate transactions to be fraudulent). Hence, maximizing the TP-FP spread may be the right measure of a successful model. Yet, one may find a model with TP rate of 90%, i.e. it correctly predicts 90% of the fraudulent transactions, but here it may correctly predict the lowest cost transactions, being entirely wrong about the top 10% most expensive frauds. Therefore, a cost model criterion may be the best judge of success, i.e a classifier whose TP rate is 10% may be the best cost performer.

Furthermore, using the natural class distribution of a data set might not yield the most effective classifiers (particularly when the distribution is highly skewed). Given a skewed distribution, we would like to generate the desired distribution without removing any data. Our approach is to create data subsets with the desired distribution, generate classifiers from these subsets, and integrate them by meta-learning their classification behavior. If, for example, we aim to change a naturally skewed 20:80 distribution of a binary classification problem into a 50:50 distribution, we can randomly divide the majority instances into four partitions and form four data subsets by by merging the minority instances with each of the four partitions containing majority instances. That is, the minority instances are replicated across four data subsets to generate the desired 50:50 distribution. Figure 8 depicts this process. Our empirical results indicate that our multi-classifier meta-learning approach using a 50:50 distribution in the data subsets for training can significantly reduce the amount of dollar loss due to illegitimate transactions. Details of our techniques and results are in [10].

Formally, let n be the size of the data set with a distribution of $x : y$ (x is the percentage of the minority class) and $u : v$ be the desired distribution. The number of minority instances is $n \times x$ and the desired number of majority instances in a subset is $nx \times \frac{v}{u}$. The number of subsets is the number of majority instances ($n \times y$) divided by the number of desired majority instances in each subset, which is $\frac{ny}{\frac{nxv}{u}}$ or $\frac{y}{x} \times \frac{u}{v}$. (When it is not a whole number, we take the ceiling ($\lceil \frac{y}{x} \times \frac{u}{v} \rceil$) and replicate some majority instances to ensure all of the majority instances are in the subsets.) That is, we have $\frac{y}{x} \times \frac{u}{v}$ subsets, each of which has nx minority instances and $\frac{nxv}{u}$

majority instances.

The next step is to apply a learning algorithm(s) to each of the subsets. Since the subsets are independent, the learning process for each subset can be run in parallel on different processors. For massive amounts of data, substantial improvement in speed can be achieved for super-linear-time learning algorithms. The generated classifiers are combined by learning (meta-learning) from their classification behavior.

At this stage, pruning can be used as additional means to improve the predictive performance of the final classification model (meta-classifier). In general, learning algorithms are designed to compute classification models with as small error rate as possible. However, when the models are evaluated with respect to different metrics (e.g. TP, cost model), their results are bound to be sub-optimal, except perhaps by chance. In such cases, pruning can help discard from the ensemble the base classifiers that do not exhibit the desired property with positive impact on the predictive performance of the final meta-classifier.

9 Empirical Evaluation

JAM has been used to compute classifier and meta-classifier agents to forewarn of possibly fraudulent credit card transactions. This section describes our experimental setting, reports the results and compares the performance of the different approaches.

9.1 Experimental setting

Learning algorithms Five inductive learning algorithms are used in our experiments, Bayes, C4.5, ID3, CART and Ripper. ID3, its successor C4.5 [59], and CART are decision tree based algorithms, Bayes, described in [43], is a naive Bayesian classifier, and Ripper [15] is a rule induction algorithm.

Learning tasks Two data sets of real credit card transactions 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).

The two data sets contained credit card transactions labeled as fraudulent or legitimate. Each bank supplied half a million records spanning one year with 20% fraud and 80% non-fraud distribution for Chase bank and 15% versus 85% for First Union bank. The schemas of the databases were 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 [66]. The records have a fixed length of 137 bytes each and about thirty numeric attributes including the binary class label (fraudulent/legitimate transaction). Some of the fields are numeric and the rest categorical, i.e. numbers were used to represent a few discrete categories.

To evaluate and compare the meta-classifiers constructed, we adopted three metrics: the overall accuracy, the TP-FP spread and a cost model fit to the credit card fraud detection problem. Overall accuracy expresses the ability of a classifier to provide correct predictions, TP-FP[4] denotes the ability of a classifier to catch fraudulent transactions while minimizing false alarms, and finally, the cost model captures the performance of a classifier with respect to the goal of the target application (stop dollar loss due to fraud).

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

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

First we distribute the data sets across six different data sites (each site storing two months of data) and we prepared the set of candidate base classifiers, i.e. the original set of base classifiers the pruning algorithm is called to evaluate. We computed these classifiers by applying the five learning algorithms to each month of data, therefore creating sixty base classifiers (ten classifiers per data site). Next, we had each data site import the "remote" base classifiers (fifty in total) that were

[4]The TP-FP spread is an ad-hoc, yet informative and simple metric characterizing the performance of the classifiers. In comparing the classifiers, one can replace the TP-FP spread, which defines a certain family of curves in the ROC plot, with a different metric or even with a complete analysis [55] in the ROC space.

subsequently used in the pruning and meta-learning phases, thus ensuring that each classifier would not be tested unfairly on known data. Specifically, we had each site use half of its local data (one month) to test, prune and meta-learn the base-classifiers and the other half to evaluate the overall performance of the pruned or unpruned meta-classifier (more details can be found in [53, 54]). In essence, the setting of this experiment corresponds to a parallel six-fold cross validation.

Finally, we had the two banks exchange their classifier agents as well. In addition to its ten local and fifty "internal" classifiers (those imported from their peer data sites), each site also imported sixty external classifiers (from the other bank). Thus, each Chase data site was populated with sixty (ten local and fifty remote) Chase classifiers and sixty First Union classifiers and each First Union site was populated with sixty (ten local and fifty remote) First Union classifiers and sixty Chase classifiers. Again, the sites used half of their local data (one month) to test, prune and meta-learn the base-classifiers and the other half to evaluate the overall performance of the pruned or unpruned meta-classifier. To ensure fairness, the ten local classifiers were not used in meta-learning.

The two databases, however, had the following schema differences:

1. Chase and First Union defined a (nearly identical) feature with different semantics

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

For the first incompatibility, we had the values of the First Union data mapped to the semantics of the Chase data. For the second incompatibility, we deployed bridging agents to compute the missing values (for a detailed discussion, see [52]). When predicting, the First Union classifiers simply disregarded the real values provided at the Chase data sites, while the Chase classifiers relied on both the common attributes and the predictions of the bridging agents to deliver a prediction at the First Union data sites.

Tables 1 and 2 summarize our results for the Chase and First Union banks respectively. Table 1 reports the performance results of the best classification models on Chase data, while Table 2 presents the performance results of the best performers on the First Union data. Both tables display the accuracy, the TP-FP spread and savings for each of the fraud predictors examined and the best result in every category is depicted in bold. The maximum achievable savings for the "ideal" classifier, with respect to our cost model, is $1470K for the Chase and $1085K for the First Union data sets. The column denoted as "size" indicates the number of base-classifiers used in the classification system.

The first row of Table 1 shows the best possible performance of Chase's own COTS authorization/detection system on this data set. The next two rows present

Table 1: Performance results for the Chase credit card data set.

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

the performance of the best base classifiers over a single subset and over the largest possible [5] data subset, while the next four rows detail the performance of the unpruned (size of 50) and best pruned meta-classifiers for each of the evaluation metrics (size of 46 for accuracy, 27 for the TP-FP spread, and 4 for the cost model). Finally, the last four rows report on the performance of the unpruned (size of 110) and best pruned meta-classifiers (sizes of 65, 43, 52) according to accuracy, the TP-FP spread and the cost model respectively. The first four meta-classifiers combine only "internal" (from Chase) base classifiers, while the last four combine both internal and external (from Chase and First Union) base classifiers. Bridging agents were not used in these experiments, since all attributes needed by First Union agents, were already defined in the Chase data.

Similar data is recorded in Table 2 for the First Union set, with the exception of First Union's COTS authorization/detection performance (it was not made available to us), and the additional results obtained when employing special bridging agents from Chase to compute the values of First Union's missing attributes. (In Table 1 we do not report results using bridging agents. First Union classifiers do not require predictive bridging agents to estimate any additional values; instead they ignore the two extra attributes of the Chase data.)

The most apparent outcome of these experiments is the superior performance of meta-learning over the single model approaches and over the traditional authorization/detection systems (at least for the given data sets). The meta-classifiers outperformed the single base classifiers (local or global) in every category. More-

[5] Determined by the available system resources.

Table 2: Performance results for the First Union credit card data set.

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

over, by bridging the two databases, we managed to further improve the performance of the meta-learning system. Notice, however, that combining classifiers agents from the two banks directly (without bridging) is not very effective. This phenomenon can be easily explained from the fact that the attribute missing from the First Union data set is significant in modeling the Chase data set. Hence, the First Union classifiers are not as effective as the Chase classifiers on the Chase data, and the Chase classifiers cannot perform at full strength at the First Union sites without the bridging agents.

An additional result, evident from these tables, is the invaluable contribution of pruning. In all cases, pruning succeeded in computing meta-classifiers with similar or better fraud detection capabilities, while reducing their size and thus improving their efficiency. A comparative study between predictive performance and meta-classifier throughput can be found in [54].

10 Conclusions

Distributed data mining systems aim to discover and combine useful information that is distributed across multiple 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 classification models or classifiers. In this study, we concentrated on the problem of acquiring useful information, efficiently and accurately, from large and distributed databases. In this respect, we described 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 models computed 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 identified several issues related to the scalability, efficiency, portability, compatibility, adaptivity, extensibility and effectiveness of distributed data mining systems. We addressed the efficiency and scalability problems first by employing distributed and asynchronous protocols at the architectural level of JAM for managing the learning agents across the data sites of the system, and second by introducing special pruning algorithms at the data site level (meta-learning level) to evaluate and combine only the most essential classifiers. To preserve portability across heterogenous platforms we built JAM upon existing agent infrastructure available over the internet and to achieve compatibility we employed special bridging agents to resolve any differences in the schemata among the distributed databases. Adaptivity is attained by extending the meta-learning techniques to combine both existing and new classifiers while extensibility is ensured by decoupling JAM from the learning algorithms and by introducing plug-and-play capabilities through objects. Finally, to evaluate and improve the effectiveness of our data mining system, we investigated more appropriate metrics (some of which are task specific) and applied meta-learning and distribution manipulation techniques to tasks with skewed distributions.

The design and implementation of useful and practical distributed data mining systems requires extensive research on all these issues. The intent of this paper is not to provide a detailed exposition of techniques but to overview the important issues and our proposed approaches (detailed discussions of our techniques and findings appear in publications cited throughout this study). This area is open and active and these problems have not been fully explored. The proposed methods were empirically evaluated against real credit card transaction data provided by two separate financial institutions where the target data mining application was to compute predictive models that detect fraudulent transactions. Our experiments suggest that meta-learning, together with distributed protocols, the pruning methods and bridging techniques constitute a highly effective and scalable approach for mining distributed data sets with the potential to contribute useful systems with broad applicability.

11 Acknowledgments

This research is supported by the Intrusion Detection Program (BAA9603) from DARPA (F30602-96-1-0311), NSF (IRI-96-32225 and CDA-96-25374) and NYSSTF (423115-445). We wish to thank Adam Banckenroth of Chase Bank and Tom French of First Union Bank for their support of this work.

References

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

[2] Michael Belford. Information overload. In *Computer Shopper*, July 1998.

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

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

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

[12] 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.

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

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

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

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

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

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

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

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

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

[22] 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.

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

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

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

[26] R. Grossman, S. Baily, S. Kasif, D. Mon, and A. Ramu. The preliminary design of papyrus: A system for high performance. In P. Chan H. Kargupta, editor, *Work. Notes KDD-98 Workshop on Distributed Data Mining*, pages 37–43. AAAI Press, 1998.

[27] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Analysis and Mach. Itell.*, 12:993–1001, 1990.

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

[29] J. Holland. Escaping brittleness: The possiblilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*, pages 593–623. Morgan Kaufmann, Los Altos, CA, 1986.

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

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

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

[33] 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.

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

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

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

[37] Jacek Maitan, Zbigniew W. Ras, and Maria Zemankova. Query handling and learning in a distributed intelligent system. In Zbigniew W. Ras, editor, *Methodologies for Intelligent Systems, 4*, pages 118–127, Charlotte, North Carolina, October 1989. North Holland.

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

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

[40] C. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 1998. In press.

[41] C. Merz and M. Pazzani. A principal components approach to combining regression estimates. *Machine Learning*, 1998. In press.

[42] 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.

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

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

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

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

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

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

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

[50] 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).

[51] A. Prodromidis. *Management of Intelligent Learning Agents in Distributed Data Min ing Systems*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 1999.

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

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

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

[55] F. Provost and T. Fawcett. Analysis 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.

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

[57] F. Provost, T. Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proc. Fifteenth Intl. Conf. Machine Learning*, pages 445–553, Madison, WI, 1998.

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

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

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

[61] 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.

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

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

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

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

[66] 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. Available from http://www.cs.columbia.edu/∼sal/JAM/PROJECT, 1998.

[67] 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.

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

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

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

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

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

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

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

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