Submitted to *IEEE Intelligent Systems' Special Issue on Data Mining, 1999*

Distributed Data Mining in Credit Card Fraud Detection

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

Wei Fan, Andreas Prodromidis, and Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York, NY 10027
{wfan,andreas,sal}@cs.columbia.edu

April 6, 1999

Abstract

Credit card transactions continue to grow in number, taking a larger share of the US payment system, and have led to a higher rate of stolen account numbers and subsequent losses by banks. Hence, improved fraud detection has become essential to maintain the viability of the US payment system. Banks have been fielding early fraud warning systems for some years. We seek to improve upon the state-of-the-art in commercial practice via large scale data mining. Scalable techniques to analyze massive amounts of transaction data to compute efficient fraud detectors in a timely manner is an important problem, especially for e-commerce. Besides scalability and efficiency, the fraud detection task exhibits technical problems that include skewed distributions of training data and non-uniform cost per error, both of which have not been widely studied in the KDD/DM community. In this article we survey and evaluate a number of techniques that we have proposed and implemented that address these three main issues concurrently. Our proposed methods of combining multiple learned fraud detectors under a "cost model" are general and demonstrably useful; our empirical results demonstrate that we can significantly reduce loss due to fraud through distributed data mining of fraud models.

1 Introduction

In today's increasingly electronic society and with the rapid advances of electronic commerce on the Internet, the use of credit cards for purchases has become convenient and necessary. Credit card transactions have become the de facto standard for internet and web-based e-commerce. The US Government estimates that there were approximately $13 Billion in sales on the internet, all by credit card, in 1998. This figure is expected to grow rapidly each year. The growing number of credit card transactions provides more opportunity for thieves to steal credit card numbers and subsequently commit fraud. When banks lose money due to credit card fraud, card holders entirely pay for the loss through higher interest rates, higher fees, and reduced benefits. Hence, it is both the banks' and card holders' interest to reduce illegitimate use of credit cards by early detection of credit card fraud. Computing "models" of credit card fraud for automated detection systems has been studied by the credit card industry for many years, and recently has been the subject matter of academic research especially with respect to e-commerce.

The credit card fraud detection domain presents a number of challenging issues to data mining. First, there are millions of credit card transaction processed each day. Mining such massive amounts of data needs highly efficient techniques that scale. Second, the data are highly skewed—many more transactions are legitimate than fraudulent. Typical accuracy-based mining techniques can generate highly accurate fraud detectors by simply predicting all transactions are legitimate, though this is equivalent to not detecting fraud at all. Third, each transaction record has a different dollar amount and hence has a variable potential loss, rather than a fixed misclassification cost (per error type), as is commonly assumed in "cost-based" mining techniques.

We address the efficiency and scalability issues in several ways. We divide a large data set of labeled transactions (either fraudulent or legitimate) into smaller subsets, apply mining techniques to generate classifiers in parallel, and combine the resultant "base" models by "meta-learning" [1] from the classifiers' behavior to generate a "meta-classifier." Our approach treats the classifiers as "black boxes" so that a variety of learning algorithms can be employed. Besides extensibility, combining multiple models computed over all available data produces meta-classifiers that can offset the loss of predictive performance that usually occurs when mining from data subsets or sampling. Another parallel approach focuses on parallelizing a particular algorithm on a particular parallel architecture. That is, a new algorithm or architecture requires substantial amount of parallel programming work. Although our architecture and algorithm-independent approach is not as efficient as the fine-grained parallelization approaches, it allows different "off the shelf" learning programs to be "plugged" into a parallel and distributed environment with relative ease and eliminates the requirement of expensive parallel hardware. Furthermore, since our approach could generate a potentially large number of classifiers from the concurrently processed data subsets, and hence potentially require more computational resources during detection, we investigate pruning methods that identify "redundant" classifiers and remove them from the ensemble without significantly degrading the predictive performance. This pruning technique can be shown to increase the computational performance and throughput of the learned detectors.

The issue of skewed distributions has not been studied widely because many of the data sets used in research do not exhibit this characteristic. We address skewness by partitioning the data set into subsets with a"desired" distribution, applying mining techniques to the subsets, and combining the mined classifiers by meta-learning (as discussed above). Some work by other researchers attempts to remove unnecessary instances from the majority class. Instances that are in the borderline region (noise or redundant exemplars) are candidates for removal. In contrast our approach keeps all the data for mining and does not change the underlying mining algorithms.

We address the issue of non-uniform cost by developing the appropriate cost model for the credit card fraud domain and biasing our methods toward reducing cost. The "desired" distribution mentioned previously is determined by this cost model. AdaCost (a cost-sensitive version of AdaBoost) relies on the cost model for updating weights in the training distribution. Naturally, this cost model also defines the primary evaluation criterion for our techniques. Furthermore, we investigate techniques to improve the cost performance of a fraud detector used by a bank by importing (remote) classifiers from other banks and combining this remotely learned knowledge with locally stored classifiers. Banks are restricted by law (and competitive reasons) from sharing information about their customers with other banks. However, they may share "black-box" fraud detection models. Our distributed data mining approach provides a direct and efficient solution to sharing knowledge without sharing data. We also address (possible) incompatibility of data schemata among different banks.

We designed and developed an agent-based distributed environment to demonstrate our distributed and parallel data mining techniques. The *JAM* (Java Agents for Meta-learning) system not only provides distributed data mining capabilities, it also allows a user to monitor and visualize the various "learning agents" and derived models in real time. A variety of algorithms and techniques have been studied by many researchers that aim to combine multiple computed models. The *JAM* system provides generic features to easily implement any of these combining techniques (as well as a large collection of base-learning algorithms) and it has been broadly available for use. For the interested reader, the *JAM* system is described in detail in [11], and is available for download at http://www.cs.columbia.edu/~sal/JAM/PROJECT.

This article provides a unified treatment and an overview of the most important results achieved on our project that are general and of interest to a researcher or data miner studying other data mining applications. (Relevant details can be found in prior publications that are cited throughout the paper.) We describe the credit card fraud data and cost models in Section 2. Details of our multi-classifier approach to skewed distributions are presented in Section 3. Section 4 details the AdaCost algorithm that incorporates a cost model in updating weights in the training distribution. Section 5 discusses our "bridging" techniques in sharing classifiers generated from different schemata. Pruning techniques for improving detection efficiency are described in Section 6 while Section 7 summarizes our contributions and mentions the impact this work has had in the related domain of intrusion detection in networked information systems.

2 Credit Card Data and Cost Models

Chase and First Union Banks, members of the FSTC (Financial Services Technology Consortium) provided us with real credit card data for this study. The two data sets contain credit card transactions labeled as fraudulent or legitimate. Each bank supplied .5 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 schemata 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 [11]. The records have a fixed length of 137 bytes each and about 30 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.

The features in this data defined by the banks essentially describe the "usage behavior" of an individual credit card account. The data is rich enough to allow alternative modeling strategies. For example, it is possible to segment the data into classes of accounts based upon "payment history"[1] It is well known that there are at least two classes of card holders, "transactors" who pay their bills in full each month, and "revolvers" who pay their minimum charge and roll over balances and interest charges to the next month. Models can thus be built for each of these market segments to determine finer distinctions between account transaction behaviors. In this work, we do no such segmentation and partitioning, but rather we compute models of "fraudulent transaction" for the entire asset base. We believe the results achieved in this study can be improved had we segmented the data[2].

Most of the machine learning literature concentrates on model accuracy (either training error or generalization error on hold out test data computed as overall accuracy, True Positive/False Positive rates, or ROC analysis). This domain provides a considerably different metric to evaluate performance of learned models; models are evaluated and rated by a "cost model."

Due to the different dollar amounts of each credit card transaction and other factors, the cost of failing to detect a fraud varies with each transaction. Hence, the cost model for this domain is based on the sum and average of loss caused by fraud. We define:

$$CumulativeCost = \sum_{i}^{n} Cost(i) \tag{1}$$

and

$$AverageCost = \frac{CumulativeCost}{n} \tag{2}$$

where $Cost(i)$ is the cost associated with transactions i and n is the total number of transactions.

After consulting with a bank representative, we jointly settled on a simplified cost model that closely reflects reality. Since it takes time and personnel to investigate a potential fraudulent transaction, an *overhead* is incurred for each investigation. That is, if the amount of a transaction is smaller than the overhead, it is not worthwhile to investigate the transaction

[1] The reader is encouraged to do a personal study of their own checking account behavior. You will likely find the same type of payments from month to month in your own checking account. Variations or distinct payments in any particular month are likely to reoccur in the same month in the prior year. Such repetitive behaviors can be regarded as a normal profile for the checking account. Significant variations from this normal profile possibly indicate fraud.

[2] However, such a strategy would delve dangerously close to industry trade secrets.

even if it is suspicious. For example, if it takes ten dollars to investigate a potential loss of one dollar, it is more economical not to investigate. Therefore, assuming a fixed *overhead*, we devised the following cost model for each transaction:

Outcome	Cost
Miss (False Negative, FN)	$tranamt$
False Alarm (False Positive, FP)	$overhead$ if $tranamt > overhead$ or 0 if $tranamt \leq overhead$
Hit (True Positive, TP)	$overhead$ if $tranamt > overhead$ or $tranamt$ if $tranamt \leq overhead$
Normal (True Negative, TN)	0

where *tranamt* is the amount of a credit card transaction. The overhead threshold, for obvious reasons, is a closely guarded secret and varies over time. The range of values used here are probably reasonable levels as bounds for this data set, but are probably significantly lower. All the empirical studies we conducted are evaluated using this cost model.

3 Skewed Distributions

Given a skewed distribution, we would like to generate a training set of labeled transactions with a "desired distribution" without removing any data, that maximizes classifier performance. In this domain, we have found that determining the "desired" distribution is an experimental art and requires extensive empirical tests to find the most effective training distribution.

Our approach is to first create data subsets with the "desired" distribution (determined by extensive sampling experiments). Then we generate classifiers from these subsets, and combine them by meta-learning from their classification behavior. For example, if the natural skewed distribution is 20:80 and the desired distribution for generating the best models is 50:50, we randomly divide the majority instances into 4 partitions and 4 data subsets are formed by merging the minority instances with each of the 4 partitions containing majority instances. That is, the minority instances are replicated across 4 data subsets to generate the desired 50:50 distribution for each distributed training set.

For concreteness, 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 $\frac{y}{x} \times \frac{u}{v}$. 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, each subset can be distributed to different processors and each learning process can be run in parallel. For massive amounts of data, substantial improvement in speed can be achieved for super-linear-time learning algorithms.

The generated classifiers are combined by meta-learning from their classification behavior. Several meta-learning strategies are described in [1]. To simplify our discussion, we only describe the *class-combiner* (or *stacking* [12]) strategy. In this strategy a meta-level training set is composed by using the (base) classifiers' predictions on a validation set as attribute values and the actual classification as the class label. This training set is then used to train

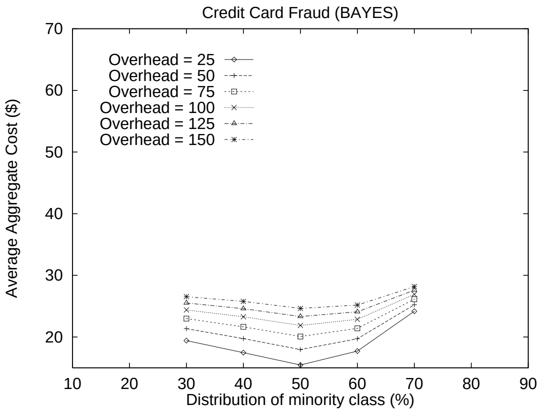

Figure 1: Training distribution vs. the credit card fraud cost model

a meta-classifier. For integrating subsets, the class-combiner strategy has been found to be more effective than the voting-based techniques.

3.1 Experiments and Results

To evaluate our multi-classifier meta-learning approach to skewed class distributions, we performed a set of experiments using the credit card fraud data from Chase. We used transactions from the first 8 months (10/95 - 5/96) for training, the ninth month (6/96) for validating, and the twelfth month (9/96) for testing[3]. Based on the empirical results from the effects of class distributions [2], the desired distribution is 50:50. Since the natural distribution is 20:80, four subsets are generated from each month for a total of 32 subsets. We applied four learning algorithms (C4.5, CART, RIPPER, and BAYES) to each subset and generated 128 base classifiers. Based on our experience with training meta-classifiers, BAYES is generally more effective and efficient, therefore BAYES is the meta-learner for all the experiments reported here.

Results from different amounts of overhead are plotted in Figure 1. Each data point is the average of ten runs using a different random seed. To demonstrate that 50:50 is indeed the desired distribution, we also ran experiments on other distributions and plotted the results in the figure. As expected, the cost is minimized when the fraud percentage is 50%. Surprisingly, 50% is the desired distribution for any of the overhead amounts. This is different from the results obtained from previous experiments when meta-learning was not used. (Note that the desired distribution is empirically determined based on the cost model and does not have to be 50%; for instance with this data set, it is 30% when the given distribution is 10:90 [2].)

Furthermore, to investigate if our approach is indeed fruitful, we ran experiments on the class-combiner strategy directly applied to the original data sets from the first 8 months

[3]Since credit card transactions have a natural "two month business cycle" (the time to bill a customer is one month, followed by a one month payment period), the true label of a transaction cannot be determined in less than two months time. Hence, building models from data in one month cannot be rationally applied for fraud detection in the next month. We therefore test our models on data that is at least two months newer.

Table 1: Cost and savings in the credit card fraud domain using meta-learning (Cost ± 95% confidence interval)

		Overhead = $50			Overhead = $75			Overhead = $100		
Method	fraud%	Cost	%saved	$saved	Cost	%saved	$saved	Cost	%saved	$saved
Class combiner	50%	17.96 ± .14	51%	761K	20.07 ± .13	46%	676K	21.87 ± .12	41%	604K
Single CART	50%	20.81 ± .75	44%	647K	23.64 ± .96	36%	534K	26.05 ± 1.25	30%	437K
Class combiner	natural	22.61	39%	575K	23.99	35%	519K	25.20	32%	471K
Avg. single classifier	natural	27.97 ± 1.64	24%	360K	29.08 ± 1.60	21%	315K	30.02 ± 1.56	19%	278K
COTS	N/A	25.44	31%	461K	26.40	29%	423K	27.24	26%	389K

(i.e., they have the natural 20:80 distribution). We also evaluate how individual classifiers generated from each month perform without class-combining.

Table 1 shows the cost and savings from class-combiner using the 50:50 distribution (128 base classifiers), the average of individual CART classifiers generated using the desired distribution (10 classifiers), class-combiner using the natural distribution (32 base classifiers—8 months × 4 learning algorithms), and the average of individual classifiers using the natural distribution (the average of 32 classifiers). (We did not perform experiments on simply replicating the minority instances to achieve 50:50 in one single data set because this approach increases the training set size and is not appropriate in domains with large amounts of data—one of the three primary issues we address here.) Compared to the other three methods, class-combining on subsets with a 50:50 fraud distribution clearly achieves a significant increase in savings—at least $110K for the month (6/96). When the overhead is $50, more than half of the losses were prevented. Surprisingly, we also observe that when the overhead is $50, a classifier ("Single CART") trained from one's month data with the desired 50:50 distribution (generated by ignoring some data) achieved significantly more savings than combining classifiers trained from all eight months' data with the natural distribution. This reaffirms the importance of employing the appropriate training class distribution in this domain. Meta-learning (class-combiner) also contributed to the improvement in performance. Consequently, utilizing the desired training distribution and meta-learning provide a synergistic approach to data mining with non-uniform class and cost distributions.

Perhaps more importantly, how do our techniques perform compared to the bank's existing fraud detection system? We label the current system COTS (commercial off-the-shelf system) in Table 1. COTS achieved significantly less savings than our techniques in the three overhead amounts we reported in the table. We note that this comparison might not be entirely accurate since COTS has much more training data than we have and it might be optimized to a different cost model (which might even be the simple error rate). Nevertheless, COTS' performance on the test data provides some indication of how the existing fraud detection systems behave in the real world. Our techniques are effective up to a certain degree of skewness because the subsets can get too small for learning effective base classifiers; details of our results from more skewed distributions are reported in [2].

4 The AdaCost Algorithm

One of the most important results of our experimental work on this domain was the realization that each of the "base" learning algorithms employed in our experiments utilized

- Given: $(x_1, c_1, y_1), \ldots, (x_m, c_m, y_m); x_i \in \mathcal{X}, c_i \in \mathbb{R}^+, y_i \in \{-1, +1\}$
- Initialize $D_1(i)$ (such as $D_1(i) = c_i / \sum_j^m c_j$)
- For $t = 1, \ldots, T$:
  1. Train weak learner using distribution D_t.
  2. Compute weak hypothesis $h_t : \mathcal{X} \to \mathbb{R}$.
  3. Choose $\alpha_t \in \mathbb{R}$ and $\beta(i) \in \mathbb{R}^+$.
  4. Update

$$D_{t+1}(i) = \frac{D_t(i)\exp\Big(-\alpha_t y_i h_t(x_i) \boxed{\beta\Big(\mathrm{sign}(y_i h_t(x_i), c_i)\Big)}\Big)}{Z_t}$$

  where $\beta(\mathrm{sign}(y_i h_t(x_i)), c_i)$ is a cost-adjustment function. Z_t is a normalization factor chosen so that D_{t+1} will be a distribution.

- Output the final hypothesis:

$$H(x) = \mathrm{sign}(f(x)) \quad \text{where} \quad f(x) = \left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Figure 2: AdaCost

"internal heuristics" based upon training accuracy, not cost. This leads us to investigate new algorithms that employs internal metrics of "misclassification cost" when computing hypotheses to predict fraud.

Here, we describe AdaCost (first reported in [3], a variant of AdaBoost [4], that reduces both fixed and variable misclassification costs more significantly than AdaBoost. We follow the generalized analysis of AdaBoost by Schapire and Singer [10]. The algorithm is shown in Figure 2. Let $S = ((x_1, c_1, y_1), \ldots, (x_m, c_m, y_m))$ be a sequence of training examples where each *instance* x_i belongs to a *domain* $\mathcal{X}$, each *cost factor* c_i belongs to the *non-negative real domain* $\mathbb{R}^+$, and each *label* y_i belongs to a finite *label space* $\mathcal{Y}$. For now, we focus on binary classification problems in which $\mathcal{Y} = \{-1, +1\}$. h is a weak hypothesis. It has the form $h : \mathcal{X} \to \mathbb{R}$. The sign of $h(x)$ is interpreted as the predicted label and the magnitude $|h(x)|$ is the "confidence" in this prediction. Let t be an index to show the round of boosting and $D_t(i)$ be the weight given to (x_i, c_i, y_i) at the t-th round. $0 \leq D_t(i) \leq 1$ and $\sum D_t(i) = 1$. α_t is a chosen parameter as a weight for weak hypothesis h_t at the t-th round. We assume $\alpha_t > 0$. $\beta(\mathrm{sign}(y_i h_t(x_i)), c_i)$ is a cost adjustment function with two arguments: $\mathrm{sign}(y_i h_t(x_i))$ to show if $h_t(x_i)$ is correct, and the cost factor c_i.

The difference between AdaCost and AdaBoost is the additional cost adjustment function $\beta(\mathrm{sign}(y_i h_t(x_i)), c_i)$ in the weight updating rule (highlighted by a box in Figure 2). Where it is clear in context, we use either $\beta(i)$ or $\beta(c_i)$ as a shorthand for $\beta(\mathrm{sign}(y_i h_t(x_i)), c_i)$. Furthermore, we use β_+ when $\mathrm{sign}(y_i h_t(x_i)) = +1$ and β_- when $\mathrm{sign}(y_i h_t(x_i)) = -1$. For an instance with a higher cost factor, $\beta(i)$ increases its weights "more" if the instance is misclassified, but decreases its weight "less" otherwise. Therefore, we require $\beta_-(c_i)$ to be non-decreasing with respect to c_i, $\beta_+(c_i)$ to be non-increasing, and both are non-negative. We proved, in [3], that AdaCost reduces cost on the training data. Next we discuss our empirical results on unseen test data.

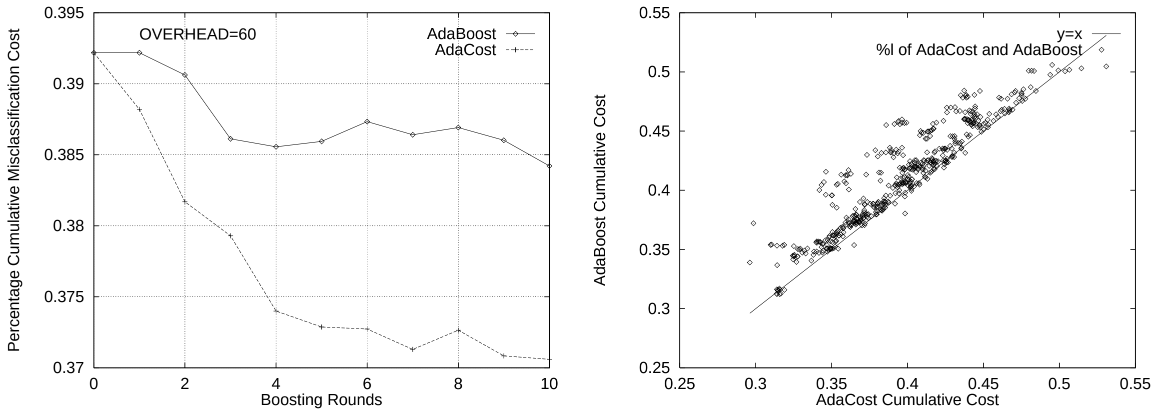

Figure 3: a. Cumulative cost versus rounds; b. AdaBoost versus AdaCost in cumulative cost

4.1 Experiments and results

Logically, we can assign a cost factor c of $tranamt - overhead$ (Section 2) to frauds and a factor c of $overhead$ to non-frauds. This reflects how the prediction errors will add to the total cost of a hypothesis. Since the actual $overhead$ is a closely guarded trade secret and is unknown to us, we chose to set $overhead \in \{60, 70, 80, 90\}$ to run four sets of experiments. We normalized each c_i to $[0, 1]$. The cost adjustment function β is chosen as: $\beta_-(c) = 0.5 \cdot c + 0.5$ and $\beta_+(c) = -0.5 \cdot c + 0.5$.

As in previous experiments, we use training data from one month and data from two months later for testing. Our data set allowed us to form 10 such pairs of training and test sets. Both AdaBoost and AdaCost were run to the 10^{th} round. We used RIPPER as the "weak" learner since it provides an easy way to change the distribution of the training set. Since using the training set alone usually over estimates the accuracy of a rule set, we used the Laplace estimate to generate the confidence for each rule.

We are interested in comparing RIPPER (as a baseline), AdaCost and AdaBoost in several dimensions. First, for each data set and each cost model, we determine which algorithm has achieved the lowest cumulative misclassification cost. We wish to know how many cases AdaCost is the clear winner. Second, we also seek to know, quantitatively, the difference in cumulative misclassification cost of AdaCost from AdaBoost and the baseline RIPPER. It is interesting to measure the significance of these differences in terms of both reduction in misclassification loss and percentage of reduction. Finally, we are interested to know if AdaCost requires more computing power.

The results from the Chase credit card data are plotted in Figure 3. Figure 3a shows the average reduction of 10 months in percentage cumulative loss (defined as $\frac{cumulativeloss}{maximalloss - leastloss} * 100\%$) for AdaBoost and AdaCost for all 10 rounds with an overhead of \$60 (results for other overheads are in [3]). We can clearly see that there is a consistent reduction. The absolute amount of reduction is around 3%[4]. We also observe that the speed of reduction by AdaCost is quicker than that of AdaBoost. The speed is the highest in the first few rounds. This means that in practice, we may not need to run AdaCost for many rounds. Figure 3b

[4]In any real-world financial application domain, 3% or "300 basis points" is very profitable.

plots the ratio of cumulative cost by AdaCost and AdaBoost. We have plotted the results of all 10 pairs of training and test months over all rounds and overheads. Most of the points are above the "y=x" line in the right drawing, implying that AdaCost has lower cumulative loss in an overwhelming number of cases.

5 Knowledge Sharing via Bridging

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 addresses the latter problem 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 schemata. Since classifiers depend directly on the format of the underlying data, minor differences in the schemata 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.

The banks 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. To facilitate the exchange of knowledge (represented as remotely learned models) and take advantage of incompatible and otherwise useless classifiers, we need to devise methods that "bridge" the differences imposed by the different schemata.

5.1 Database Compatibility

The *"incompatible schema"* problem impedes JAM from taking advantage of all available databases. Let's consider two data sites A and B with databases DB_A and DB_B, respectively, with similar but not identical schemata. Without loss of generality, we assume that:

$$Schema(DB_A) = \{A_1, A_2, ..., A_n, A_{n+1}, C\} \tag{3}$$
$$Schema(DB_B) = \{B_1, B_2, ..., B_n, B_{n+1}, C\} \tag{4}$$

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.:

$$Schema(DB_A) = \{A_1, A_2, ..., A_n, A_{n+1}, C\} \quad (5)$$
$$Schema(DB_B) = \{B_1, B_2, ..., B_n, C\} \quad (6)$$

where we assume that attribute B_{n+1} is not present in DB_B.[5]

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 schemata.

5.2 Bridging methods

We describe two methods for handling the missing attributes. The details of these approaches can be found in [7].

Method I: Learn a local model for the missing attribute and exchange.
Database DB_B imports, along with the remote classifier agent, a *bridging* agent from database DB_A that is trained to compute values for the missing attribute A_{n+1} in DB_B's data. Next, DB_B applies the bridging agent on its own data to estimate the missing values. In many cases, however, this may not be possible or desirable by DB_A (e.g. in case the attribute is propriety). The alternative for database DB_B is to simply add the missing attribute to its data set and fill it with null values. Even though the missing attribute may have high predictive value for DB_A, it is of no value to DB_B. After all, DB_B did not include it in its schema and presumably other attributes (including the common ones) do have predictive value.

Method II: Learn a local model without the missing attribute and exchange.
In this approach, database DB_A can learn two local models. One with the attribute A_{n+1} that can be used locally by the meta-learning agents, and one without it that can be subsequently exchanged. Learning a second classifier agent without the missing attribute, or with the attributes that belong to the intersection of the attributes of the data sets of the two

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

databases, implies that the second classifier makes use only of the attributes that are common among the participating sites and no issue exists for its integration at other databases. On the other hand, remote classifiers imported by database DB_A (and assured not to involve predictions over the missing attributes) can still be locally integrated with the original model that employs A_{n+1}. In this case, the remote classifiers simply ignore the missing attributes of the local data set.

Both approaches address the *incompatible schema problem* and meta-learning over these models should proceed in a straightforward manner. Compared to related work by Ras [9], our approach is more general since our techniques support both categorical and continuous attributes, and are not limited to a specific syntactic case or purely logical consistency of generated rule models. Instead, these bridging techniques can employ machine or statistical learning algorithms to compute arbitrary models for the missing values. We conducted experiments to evaluate our techniques and the results are reported in Section 6.1 in conjunction with experiments on pruning.

6 Pruning

An ensemble of classifiers 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 pre-training pruning and the 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.

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, *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 ranks and selects its classifiers by evaluating each candidate classifier independently (metric-based), the second algorithm decides by examining the classifiers in correlation with each other (diversity-based), while the third relies on the independent performance of the classifiers and the manner in which they predict with respect to each other and with respect to the underlying data set (coverage/specialtybased). The pre-training pruning algorithms are described in detail in [8]. The post-training pruning algorithms are based, the first on a cost-complexity pruning technique (a technique used by the CART decision tree learning algorithm that seeks to minimize the cost/size of its tree while reducing the misclassification rate) and the second on the correlation between the classifiers and the meta-classifier. Both algorithms are detailed in [8].

Compared to Margineantu and Dietterich's approach [6], our approach considers 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 (eg., AdaBoost) over the predictions of classifiers for the final classification, we use meta-learning to combine predictions of the individual classifiers.

6.1 Evaluation of knowledge sharing and pruning

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 5 learning algorithms (BAYES, C4.5, ID3, CART and RIPPER) to each month of data, therefore creating 60 base classifiers (10 classifiers per data site). Next, we had each data site import the "remote" base classifiers (50 in total) that were 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 [8]). In essence, the setting of this experiment corresponds to a parallel 6-fold cross validation.

Finally, we had the two "simulated banks" exchange their classifier agents as well. In addition to its 10 local and 50 "internal" classifiers (those imported from their peer data sites), each site also imported 60 external classifiers (from the other bank). Thus, each simulated Chase data site was populated with 60 (10+50) Chase classifiers and 60 First Union classifiers and each First Union site was populated with 60 (10+50) First Union classifiers and 60 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 10 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 [7]). 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.

Table 2: Results on knowledge sharing and pruning

Classification Method	Chase Size	Chase Savings	First Union Size	First Union Savings
COTS scoring system from Chase	N/A	$ 682K	N/A	N/A
Best base classifier over entire set	1	$ 762K	1	$ 803K
Best base classifier over one subset	1	$ 736K	1	$ 770K
Meta-classifier	50	$ 818K	50	$ 944K
Meta-classifier	32	$ 870K	21	$ 943K
Meta-classifier (+ First Union)	110	$ 800K	N/A	N/A
Meta-classifier (+ First Union)	63	$ 877K	N/A	N/A
Meta-classifier (+ Chase - bridging)	N/A	N/A	110	$ 942K
Meta-classifier (+ Chase + bridging)	N/A	N/A	110	$ 963K
Meta-classifier (+ Chase + bridging)	N/A	N/A	56	$ 962K

Table 2 summarizes our results for the Chase and First Union banks. Both tables display the savings for each of the fraud predictors examined. The column denoted as "size" indicates the number of base-classifiers used in the ensemble classification system. The first row of Table 2 shows the best possible performance of Chase's own COTS authorization/detection system on this data set. The next two rows present the performance of the best base classifiers over the entire set and over a single month's data, while the last four rows detail the performance of the unpruned (size of 50 and 110) and pruned meta-classifiers (size of 32 and 63). The first two of these meta-classifiers combine only "internal" (from Chase) base classifiers, while the last two combine both internal and external (from Chase and First Union) base classifiers. Bridging agents were not used in these experiments, since Chase bank defined all attributes used by the First Union classifier agents.

Similar data is recorded in Table 2 for the First Union data 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. 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 in every category. Moreover, by bridging the two databases, we managed to further improve the performance of the meta-learning system. Notice, however, that combining classifiers agents from the two banks directly (without bridging) 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 detailed description on the pruning methods and a comparative study between predictive performance and meta-classifier throughput can be found in [8].

7 Concluding Remarks

Credit card fraud detection poses very interesting technical problems for data mining, with complex real world constraints that invalidate some of the conventional assumptions in the research community.

We are confronted with three main issues: efficiency of highly distributed databases and detection systems, skewed distributions, and non-uniform cost per error. Our proposed methods attempt to solve these problems by training on subsets of distributed data with a "desired distribution" that maximizes performance under a "cost model", updates weights in a new boosting algorithm called AdaCost according to a realistic cost model, imports and combines classifiers among participating (and distributed) banks, and prunes redundant classifiers to improve detection efficiency and transaction throughput. We also designed and implemented a distributed environment called *JAM* to facilitate distributed and parallel data mining. Though our exposition here only highlights a subset of our results, these results represent encouraging news for future e-commerce fraud detection. Our scalable black-box approach can build fraud detectors efficiently and they can significantly reduce loss due to illegitimate behavior. In many cases, our methods outperformed a state-of-the-art commercial fraud detection system.

Our experience in credit card fraud detection has also impacted other important applications. Encouraged by our results in fraud detection[6], we shifted our attention to the growing problem of intrusion detection in (network- and host-) based computer systems. Here we seek to perform the same sort of task as in the credit card fraud domain. We seek to build models to distinguish between "bad" (intrusions or attacks) and "good" (normal) connections or processes. By first applying feature extraction algorithms, followed by the application of machine learning algorithms (eg., RIPPER) to learn and combine multiple models for different types of intrusions, remarkably good success has been achieved in detecting intrusions [5]. This work, as well as the results reported in this paper demonstrates convincingly that distributed data mining techniques, that combine multiple models, indeed produces effective fraud and intrusion detectors.

Acknowledgments

The authors would like to thank the efforts of past and current participants of this project: Charles Elkan, Wenke Lee, Shelley Tselepis, Alex Tuzhilin, and Junxin Zhang. We also appreciate the data and expertise provided by Chase and First Union for conducting this study. This research was partially supported by grants from DARPA (F30602-96-1-0311) and NSF (IRI-96-32225 & CDA-96-25374).

[6]and under "encouragement" from DARPA, our primary sponsor of this work...

References

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

[2] P. Chan and S. Stolfo. Learning with non-uniform distributions: Effects and a multi-classifier approach. Submitted to Machine Learning Journal, 1999.

[3] W. Fan, S. Stolfo, J. Zhang, and P. Chan. Adacost: Misclassification cost-sensitive boosting. In *Proc. 16th Intl. Conf. Machine Learning*, 1999. To appear.

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

[5] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In *Proc. IEEE Symp. Security and Privacy*, 1999. To appear.

[6] D. Margineantu and T. Dietterich. Pruning adaptive boosting. In *Proc. 14th Intl. Conf. Machine Learning*, pages 211–218, 1997.

[7] A. Prodromidis and S. Stolfo. Mining databases with different schemas: Integrating imcompatible classifiers. In *Proc. 4th Intl Conf. Knowledge Discovery and Data Mining*, pages 314–318, 1998.

[8] A. Prodromidis, S. Stolfo, and P. Chan. Pruning classifiers in a distributed meta-learning system. Submitted to Data Mining and Knowledge Discovery Journal, 1998.

[9] Z. Ras. Answering non-standard queries in distirbuted knowledge-based systems. In L. Polkowski and A. Skowron, editors, *Rough sets in Knowledge Discovery, Studies in Fuzziness and Soft Computing*, pages 98–108. Physica Verlag, 1998.

[10] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proc. 11th Conf. Computational Learning Theory*, 1998.

[11] 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.

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