

# Distributed Data Mining in Credit Card Fraud Detection

Philip K. Chan, Florida Institute of Technology  
Wei Fan, Andreas L. Prodromidis, and Salvatore J. Stolfo, Columbia University

**C**REDIT CARD TRANSACTIONS CONTINUE to grow in number, taking an ever-larger share of the US payment system and leading to a higher rate of stolen account numbers and subsequent losses by banks. Improved fraud detection thus has become essential to maintain the viability of the US payment system. Banks have used early fraud warning systems for some years.

Large-scale data-mining techniques can improve on the state of the art in commercial practice. Scalable techniques to analyze massive amounts of transaction data that efficiently compute 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 nonuniform cost per error, both of which have not been widely studied in the knowledge-discovery and data-mining community.

In this article, we survey and evaluate a number of techniques 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.

*THIS SCALABLE BLACK-BOX APPROACH FOR BUILDING EFFICIENT FRAUD DETECTORS CAN SIGNIFICANTLY REDUCE LOSS DUE TO ILLEGITIMATE BEHAVIOR. IN MANY CASES, THE AUTHORS' METHODS OUTPERFORM A WELL-KNOWN, STATE-OF-THE-ART COMMERCIAL FRAUD-DETECTION SYSTEM.*

## Our approach

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 credit cards accounted for approximately US \$13 billion in Internet sales during 1998. This figure is expected to grow rapidly each year. However, 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 because of credit card fraud, cardholders pay for all of that loss through higher interest rates, higher fees, and reduced benefits. Hence, it is in both the banks' and the

cardholders' interest to reduce illegitimate use of credit cards by early fraud detection. For many years, the credit card industry has studied computing models for automated detection systems; recently, these models have been the subject of academic research, especially with respect to e-commerce.

The credit card fraud-detection domain presents a number of challenging issues for data mining:

- There are millions of credit card transactions processed each day. Mining such massive amounts of data requires highly efficient techniques that scale.
- 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 that all

## 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 internal heuristics based on training accuracy, not cost. This leads us to investigate new algorithms that employ internal metrics of *misclassification cost* when computing hypotheses to predict fraud.

Here, we describe AdaCost<sup>1</sup> (a variant of AdaBoost<sup>2,3</sup>), which reduces both fixed and variable misclassification costs more significantly than AdaBoost. We follow the generalized analysis of AdaBoost by Robert Schapire and Yoram Singer.<sup>3</sup> Figure A shows the algorithm. Let  $S = (x_1, c_1, y_1), \dots, (x_m, c_m, y_m)$  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  $R^+$ , and each label  $y_i$  belongs to a finite label space  $\mathcal{Y}$ . We only focus on binary classification problems in which  $\mathcal{Y} = \{-1, +1\}$ .  $h$  is a weak hypothesis—it has the form  $h: \mathcal{X} \rightarrow R$ . 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  $\sum D_t(i) = 1$  is the chosen parameter as a weight for weak hypothesis  $h_t$  at the  $t$ th round. We assume  $\alpha_t > 0$ .  $\beta(\text{sign}(y_i h_t(x_i)), c_i)$  is a cost-adjustment function with two arguments:  $\text{sign}(y_i h_t(x_i))$  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  $\beta(\text{sign}(y_i h_t(x_i)), c_i)$  in the weight-updating rule.

Given:  $(x_1, c_1, y_1), \dots, (x_m, c_m, y_m): x_i \in \mathcal{X}, c_i \in R^+, y_i \in \{-1, +1\}$   
 Initialize  $D_1(i)$  (such as  $D_1(i) = c_i / \sum c_i$ )  
 For  $t = 1, \dots, T$ :

1. Train weak learner using distribution  $D_t$ .
2. Compute weak hypothesis  $h_t: \mathcal{X} \rightarrow R$ .
3. Choose  $\alpha_t \in R$  and  $\beta(i) \in R^+$
4. Update

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

where  $\beta(\text{sign}(y_i h_t(x_i)), c_i)$  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:

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

Figure A. AdaCost.

Where it is clear in context, we use either  $\beta(i)$  or  $\beta(c_i)$  as a shorthand for  $\beta(\text{sign}(y_i h_t(x_i)), c_i)$ . Furthermore, we use  $\beta_+$  when  $\text{sign}(y_i h_t(x_i)) = +1$  and  $\beta_-$  when  $\text{sign}(y_i h_t(x_i)) = -1$ . 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 nondecreasing with respect to  $c_i$ ,  $\beta_+(c_i)$  to be nonincreasing, and both are nonnegative. We proved that AdaCost reduces cost on the training data.<sup>1</sup>

Logically, we can assign a cost factor  $c$  of *transamt – overhead* to frauds and a factor  $c$  of overhead to nonfrauds. This reflects how the prediction errors will add to the total cost of a hypothesis. Because 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  $\beta$  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 let us form 10 such pairs of training and test sets. We ran both AdaBoost and AdaCost to the 50th round. We used Ripper as the “weak” learner because it provides an easy way to change the training set’s distribution. Because using the training set alone usually overestimates a rule set’s accuracy, 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 cost model, we determine which algorithm has achieved the lowest cumulative misclassification cost. We wish to know in 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.

Figure B plots the results from the Chase Bank’s credit card data. Figure B1 shows the average reduction of 10 months in percentage cumulative loss (defined as  $\text{cumulative loss} / \text{maximal loss} - \text{least loss} * 100\%$ ) for AdaBoost and AdaCost for all 50 rounds with an overhead of \$60 (results for other overheads are in other work<sup>1</sup>). We can clearly see that there is a consistent reduction. The absolute amount of reduction is around 3%

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 might not need to run AdaCost for many rounds. Figure B2 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 Figure B2, implying that AdaCost has lower cumulative loss in an overwhelming number of cases.

## References

1. W. Fan et al., “Adacost: Misclassification Cost-Sensitive Boosting,” Proc. 16th Int’l Conf. Machine Learning, Morgan Kaufmann, San Francisco, 1999, pp. 97–105.

transactions are legitimate, although this is equivalent to not detecting fraud at all.

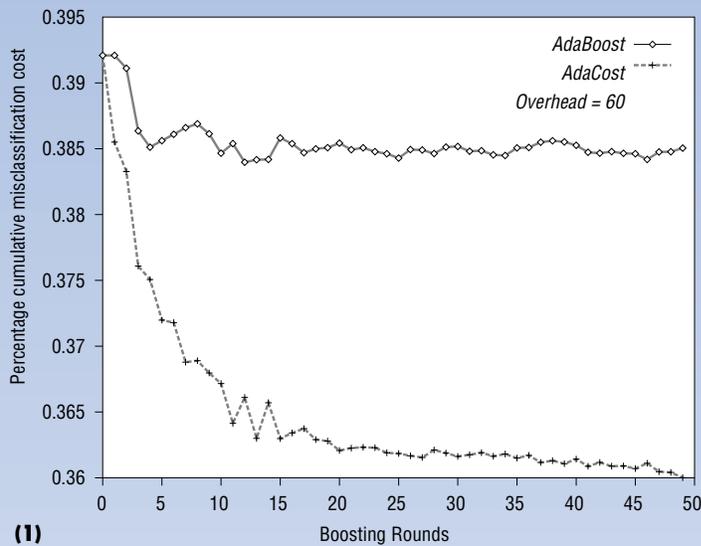
- Each transaction record has a different dollar amount and thus has a variable potential loss, rather than a fixed misclassification cost per error type, as is commonly assumed in cost-based mining techniques.

Our approach addresses the efficiency and scalability issues in several ways. We divide

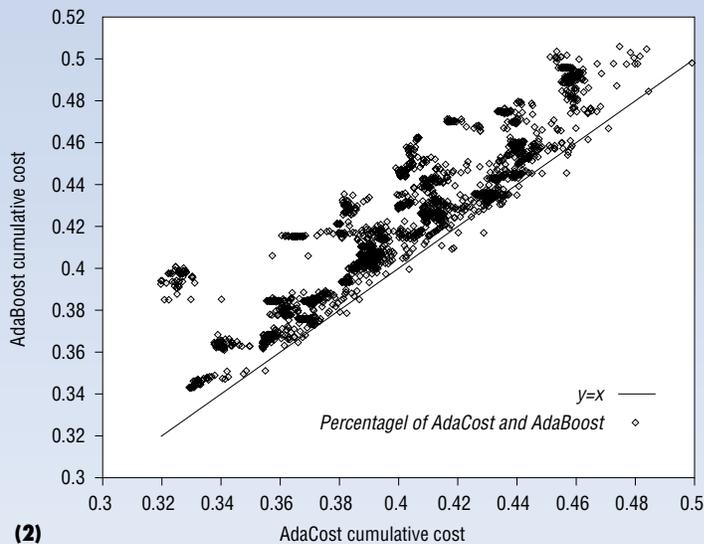
a large data set of labeled transactions (either fraudulent or legitimate) into smaller subsets, apply mining techniques to generate classifiers in parallel, and combine the resultant base models by metalearning from the classifiers’ behavior to generate a metaclassifier.<sup>1</sup> Our approach treats the classifiers as black boxes so that we can employ a variety of learning algorithms. Besides extensibility, combining multiple models computed over all available data produces metaclassifiers that can offset

the loss of predictive performance that usually occurs when mining from data subsets or sampling. Furthermore, when we use the learned classifiers (for example, during transaction authorization), the base classifiers can execute in parallel, with the metaclassifier then combining their results. So, our approach is highly efficient in generating these models and also relatively efficient in applying them.

Another parallel approach focuses on parallelizing a particular algorithm on a particu-



(1)



(2)

Figure B. Cumulative cost versus rounds (1); AdaBoost versus AdaCost in cumulative cost (2).

2. Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm," *Proc. 13th Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1996, pp. 148–156.
3. R. Schapire and Y. Singer, "Improved Boosting Algorithms Using Confidence-Rated Predictions," *Proc. 11th Conf. Computational Learning Theory*, ACM Press, New York, 1998.

lar parallel architecture. However, a new algorithm or architecture requires a substantial amount of parallel-programming work. Although our architecture- and algorithm-independent approach is not as efficient as some fine-grained parallelization approaches, it lets users plug different off-the-shelf learning programs into a parallel and distributed environment with relative ease and eliminates the need for expensive parallel hardware.

Furthermore, because our approach could

generate a potentially large number of classifiers from the concurrently processed data subsets, and therefore 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 increases the learned detectors' computational performance and throughput.

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 metalearning (as we have already discussed). Other researchers attempt 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 nonuniform cost by developing the appropriate cost model for the credit card fraud domain and biasing our methods toward reducing cost. This cost model determines the desired distribution just mentioned. AdaCost (a cost-sensitive version of AdaBoost) relies on the cost model for updating weights in the training distribution. (For more on AdaCost, see the "AdaCost algorithm" sidebar.) Naturally, this cost model also defines the primary evaluation criterion for our techniques. Furthermore, we investigate techniques to improve the cost performance of a bank's fraud detector by importing remote classifiers from other banks and combining this remotely learned knowledge with locally stored classifiers. The law and competitive concerns restrict banks 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 Metalearning) system not only provides distributed data-mining capabilities, it also lets users monitor and visualize the various learning agents and derived models in real time. Researchers have studied a variety of algorithms and techniques for combining 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. The JAM system is available for download at <http://www.cs.columbia.edu/~sal/JAM/PROJECT>.<sup>2</sup>

## Credit card data and cost models

Chase Bank and First Union Bank, members of the Financial Services Technology Consortium (FSTC), 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 500,000 records spanning one year with 20% fraud and 80% nonfraud distribution for Chase Bank and 15% versus 85% for First Union Bank. In practice, fraudulent transactions are much less frequent than the 15% to 20% observed in the data given to us. These data might have been cases where the banks have difficulty in determining legitimacy correctly. In some of our experiments, we deliberately create more skewed distributions to evaluate the effectiveness of our techniques under more extreme conditions.

Bank personnel developed the schemata of the databases over years of experience and continuous analysis to capture important information for fraud detection. We cannot reveal the details of the schema beyond what we have described elsewhere.<sup>2</sup> The records of one schema have a fixed length of 137 bytes each and about 30 attributes, including the binary class label (fraudulent/legitimate transaction). Some fields are numeric and the rest categorical. Because account identification is not present in the data, we cannot group transactions into accounts. Therefore, instead of learning behavior models of individual customer accounts, we build overall models that try to differentiate legitimate transactions from fraudulent ones. Our models are customer-independent and can serve as a second line of defense, the first being customer-dependent models.

Most machine-learning literature concentrates on model accuracy (either training error or generalization error on hold-out test data computed as overall accuracy, true-positive or false-positive rates, or return-on-cost analysis). This domain provides a considerably different metric to evaluate the learned models' performance—models are evaluated and rated by a cost model. Due to the different dollar amount 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 relies on the sum and average of loss caused by fraud. We define

$$\text{CumulativeCost} = \sum_i^n \text{Cost}(i)$$

and

$$\text{AverageCost} = \frac{\text{CumulativeCost}}{n}$$

where  $\text{Cost}(i)$  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. Because it takes time and personnel to investigate a potentially fraudulent transaction, each investigation incurs an *overhead*. Other related costs—for example, the operational resources needed for the fraud-detection system—are consolidated into *overhead*. So, if the amount of a transaction is smaller than the overhead, investigating the transaction is not worthwhile even if it is suspicious. For example, if it takes \$10 to investigate a potential loss of \$1, it is more economical not to investigate it. Therefore, assuming a fixed *overhead*, we devised the cost model shown in Table 1 for each transaction, 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. We evaluated all our empirical studies using this cost model.

### 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, which maximizes classifier performance. In this domain, we found that determining the desired distribution is an experimental art and requires extensive empirical tests to find the most effective training distribution.

In our approach, we first create data subsets with the desired distribution (determined by extensive sampling experiments). Then we generate classifiers from these subsets and combine them by metalearning from their classification behavior. For example, if the given skewed distribution is 20:80 and the desired distribution for generating the best models is 50:50, we randomly divide the

majority instances into four partitions and form four data subsets by merging the minority instances with each of the four partitions containing majority instances. That is, the minority instances replicate across four 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  $Nx \times v/u$ . 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  $Ny$  divided by  $Nxv/u$  or  $y/x \times u/v$ . So, we have  $y/x \times u/v$  subsets, each of which has  $Nx$  minority instances and  $Nxv/u$  majority instances.

The next step is to apply a learning algorithm or algorithms to each subset. Because the learning processes on the subsets are independent, the subsets can be distributed to different processors and each learning process can run in parallel. For massive amounts of data, our approach can substantially improve speed for superlinear time-learning algorithms. The generated classifiers are combined by metalearning from their classification behavior. We have described several metalearning strategies elsewhere.<sup>1</sup>

To simplify our discussion, we only describe the *class-combiner* (or *stacking*) strategy.<sup>3</sup> This strategy composes a metalevel training set by using the base classifiers' predictions on a validation set as attribute values and the actual classification as the class label. This training set then serves for training a metaclassifier. For integrating subsets, the class-combiner strategy is more effective than the voting-based techniques. When the learned models are used during online fraud detection, transactions feed into the learned base classifiers and the metaclassifier then combines their predictions. Again, the base classifiers are independent and can execute in parallel on different processors. In addition, our approach can prune redundant base classifiers without affecting the cost performance, making it relatively efficient in the credit card authorization process.

Table 1. Cost model assuming a fixed overhead.

OUTCOME	COST
Miss (false negative—FN)	<i>Tranamt</i>
False alarm (false positive—FP)	<i>Overhead</i> if <i>tranamt</i> > <i>overhead</i> or 0 if <i>tranamt</i> ≤ <i>overhead</i>
Hit (true positive—TP)	<i>Overhead</i> if <i>tranamt</i> > <i>overhead</i> or <i>tranamt</i> if <i>tranamt</i> ≤ <i>overhead</i>
Normal (true negative—TN)	0

Table 2. Cost and savings in the credit card fraud domain using class-combiner (cost  $\pm$  95% confidence interval).

METHOD	FRAUD (%)	OVERHEAD = \$50			OVERHEAD = \$75			OVERHEAD = \$100		
		COST	SAVED (%)	SAVED (\$K)	COST	SAVED (%)	SAVED (\$K)	COST	SAVED (%)	SAVED (\$K)
Class combiner	50	17.96 $\pm$ .14	51	761	20.07 $\pm$ .13	46	676	21.87 $\pm$ .12	41	604
Single CART	50	20.81 $\pm$ .75	44	647	23.64 $\pm$ .96	36	534	26.05 $\pm$ 1.25	30	437
Class combiner	Given	22.61	39	575	23.99	35	519	25.20	32	471
Average single classifier	Given	27.97 $\pm$ 1.64	24	360	29.08 $\pm$ 1.60	21	315	30.02 $\pm$ 1.56	19	278
COTS	N/A	25.44	31	461	26.40	29	423	27.24	26	389

**Experiments and results.** To evaluate our multiclassifier class-combiner approach to skewed class distributions, we performed a set of experiments using the credit card fraud data from Chase.<sup>4</sup> We used transactions from the first eight months (10/95–5/96) for training, the ninth month (6/96) for validating, and the twelfth month (9/96) for testing. (Because 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.)

Based on the empirical results from the effects of class distributions, the desired distribution is 50:50. Because the given 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, so it is the metalearner for all the experiments reported here.

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 eight months (that is, they have the given 20:80 distribution). We also evaluated how individual classifiers generated from each month perform without class-combining.

Table 2 shows the cost and savings from the class-combiner strategy 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 given distribution (32 base classifiers—8 months  $\times$  4 learning algorithms), and the average of individual classifiers using the given 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 \$110,000 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 month’s 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 given distribution. This reaffirms the importance of employing the appropriate training class distribution in this domain. Class-combiner also contributed to the performance improvement. Consequently, utilizing the desired training distribution and class-combiner provides a synergistic approach to data mining with nonuniform 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 2. COTS achieved significantly less savings than our techniques in the three overhead amounts we report in this table.

This comparison might not be entirely accurate because 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). Furthermore, unlike COTS, our techniques are general for problems with skewed distributions and do not utilize any domain knowledge in detecting credit card fraud—the only exception is the cost model used for evaluation and search guidance. Nevertheless, COTS’ performance on the test data provides some indication of how the existing fraud-detection system behaves in the real world.

We also evaluated our method with more skewed distributions (by downsampling minority instances): 10:90, 1:99, and 1:999.

As we discussed earlier, the desired distribution is not necessarily 50:50—for instance, the desired distribution is 30:70 when the given distribution is 10:90. With 10:90 distributions, our method reduced the cost significantly more than COTS. With 1:99 distributions, our method did not outperform COTS. Both methods did not achieve any savings with 1:999 distributions.

To characterize the condition when our techniques are effective, we calculate  $R$ , the ratio of the overhead amount to the average cost:  $R = \text{Overhead}/\text{Average cost}$ . Our approach is significantly more effective than the deployed COTS when  $R < 6$ . Both methods are not effective when the  $R > 24$ . So, under a reasonable cost model with a fixed overhead cost in challenging transactions as potentially fraudulent, when the number of fraudulent transactions is a very small percentage of the total, it is financially undesirable to detect fraud. The loss due to this fraud is yet another cost of conducting business.

However, filtering out “easy” (or low-risk) transactions (the data we received were possibly filtered by a similar process) can reduce a high overhead-to-loss ratio. The filtering process can use fraud detectors that are built based on individual customer profiles, which are now in use by many credit card companies. These individual profiles characterize the customers’ purchasing behavior. For example, if a customer regularly buys groceries at a particular supermarket or has set up a monthly payment for phone bills, these transactions are close to no risk; hence, purchases of similar characteristics can be safely authorized without further checking. Reducing the overhead through streamlining business operations and increased automation will also lower the ratio.

## Knowledge sharing through bridging

Much of the prior work on combining multiple models assumes that all models originate from different (not necessarily dis-

tinct) subsets of a single data set as a means to increase accuracy (for example, by imposing probability distributions over the instances of the training set, or by stratified sampling, subsampling, and so forth) and not as a means to integrate distributed information. Although the JAM system addresses the latter problem by employing metalearning techniques, integrating classification models derived from distinct and distributed databases might not always be feasible.

In all cases considered so far, all classification models are assumed to originate from databases of identical schemata. Because classifiers depend directly on the underlying data's format, minor differences in the schemata between databases derive incompatible classifiers—that is, 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 thus incorporate useful information in their system 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 that 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.

**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  $DB_B$ , having similar but not identical schemata. Without loss of generality, we assume that

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

where,  $A_i$  and  $B_i$  denote the  $i$ th attribute of  $DB_A$  and  $DB_B$ , respectively, and  $C$  the class label (for example, the fraud/legitimate label in the credit card fraud illustration) 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; that is,

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

where we assume that attribute  $B_{n+1}$  is not present in  $DB_B$ . (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 might 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 (that is,  $A_{n+1}$  might denote the number of times an event occurred within a window of half an hour and  $B_{n+1}$  might denote the number of times the same event occurred but within 10 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  $C_{A_j}$  derived from  $DB_A$  are not compatible with  $DB_B$ 's data and therefore 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 metalearning their classifier agents is to be able to combine information from different sources.

**Bridging methods.** There are two methods for handling the missing attributes.<sup>5</sup>

*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 might not be possible or desirable by  $DB_A$  (for example, in case the attribute is proprietary). 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 might 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) 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 metalearning 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 two databases' data sets, implies that the second classifier uses only the attributes that are common among the participating sites and no issue exists for its integration at other databases. But, 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 local data set's missing attributes.

Both approaches address the *incompatible schema problem*, and metalearning over these models should proceed in a straightforward manner. Compared to Zbigniew Ras's related work,<sup>6</sup> our approach is more general because 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.

## Pruning

An ensemble of classifiers can be unnecessarily complex, meaning that many classifiers might be redundant, wasting resources and reducing system throughput. (Throughput here denotes the rate at which a metaclassifier can pipe through and label a stream of data items.) We study the efficiency of metaclassifiers by investigating the effects of pruning (discarding certain base classifiers) on their performance. Determining the optimal set of classifiers for metalearning is a combinatorial problem. Hence, the objective of pruning is to utilize heuristic methods to search for partially grown metaclassifiers (metaclassifiers with pruned subtrees) that are more efficient and scalable and at the same time achieve comparable or better predictive performance

Table 3. Results on knowledge sharing and pruning.

CLASSIFICATION METHOD	CHASE		FIRST UNION	
	SIZE	SAVINGS (\$K)	SIZE	SAVINGS (\$K)
COTS scoring system from Chase	N/A	682	N/A	N/A
Best base classifier over entire set	1	762	1	803
Best base classifier over one subset	1	736	1	770
Metaclassifier	50	818	50	944
Metaclassifier	32	870	21	943
Metaclassifier (+ First Union)	110	800	N/A	N/A
Metaclassifier (+ First Union)	63	877	N/A	N/A
Metaclassifier (+ Chase – bridging)	N/A	N/A	110	942
Metaclassifier (+ Chase + bridging)	N/A	N/A	110	963
Metaclassifier (+ Chase + bridging)	N/A	N/A	56	962

results than fully grown (unpruned) meta-classifiers. To this end, we introduced two stages for pruning meta-classifiers; the pre-training and posttraining pruning stages. Both levels are essential and complementary to each other with respect to the improvement of the system's accuracy and efficiency.

*Pretraining pruning* refers to the filtering of the classifiers before they are combined. Instead of combining classifiers in a brute-force manner, we introduce a preliminary stage for analyzing the available classifiers and qualifying them for inclusion in a combined meta-classifier. Only those classifiers that appear (according to one or more pre-defined metrics) to be most promising participate in the final meta-classifier. Here, we adopt a black-box approach that evaluates the set of classifiers based only on their input and output behavior, not their internal structure. Conversely, *posttraining 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 pretraining and two posttraining pruning algorithms, each with different search heuristics.

The first pretraining pruning algorithm ranks and selects its classifiers by evaluating each candidate classifier independently (metric-based), and the second algorithm decides by examining the classifiers in correlation with each other (diversity-based). 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 and specialty-based). The first posttraining pruning algorithms are based on a cost-complexity pruning technique (a technique the CART decision-tree learning algorithm uses that seeks to minimize the cost and size of its tree while reducing the misclassification rate). The second is based on the correlation between the classifiers and the meta-classifier.<sup>7</sup>

Compared to Dragos Margineantu and Thomas Dietterich's approach,<sup>8</sup> ours 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 (such as AdaBoost) over the predictions of classifiers for the final classification, we use metalearning to combine the individual classifiers' predictions.

**Evaluation of knowledge sharing and pruning.** First, we distributed the data sets

across six different data sites (each site stores two months of data) and prepared the set of candidate base classifiers—that is, the original set of base classifiers the pruning algorithm is called to evaluate. We computed these classifiers by applying five learning algorithms (Bayes, C4.5, ID3, CART, and Ripper) to each month of data, creating 60 base classifiers (10 classifiers per data site). Next, we had each data site import the remote base classifiers (50 in total) that we subsequently used in the pruning and metalearning 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 metalearn the base classifiers and the other half to evaluate the pruned and unpruned meta-classifier's overall performance.<sup>7</sup> In essence, this experiment's setting corresponds to a parallel sixfold cross-validation.

Finally, we had the two simulated banks exchange their classifier agents. 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, we populated each simulated Chase data site with 60 (10 + 50) Chase classifiers and 60 First Union classifiers, and we populated each First Union site 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 metalearn the base classifiers and the other half to evaluate the pruned or unpruned meta-classifier's overall performance. To ensure fairness, we did not use the 10 local classifiers in metalearning.

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

- Chase and First Union defined a (nearly identical) feature with different semantics.
- Chase includes two (continuous) features not present in the First Union data.

For the first incompatibility, we mapped the First Union data values to the Chase data's semantics. For the second incompatibility, we deployed bridging agents to compute the missing values.<sup>5</sup> 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 3 summarizes our results for the Chase and First Union banks, displaying the savings for each fraud predictor examined. The column denoted as *size* indicates the number of base classifiers used in the ensemble classification system. The first row of Table 3 shows the best possible performance of Chase's own COTS authorization and 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. We did not use bridging agents in these experiments, because Chase defined all attributes used by the First Union classifier agents.

Table 3 records similar data for the First Union data set, with the exception of First Union's COTS authorization and 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. Most obviously, these experiments show the superior performance of metalearning over the single-model approaches and over the traditional authorization and 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 metalearning system's performance.

However, combining classifiers' agents from the two banks directly (without bridging) is not very effective, no doubt because 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.

This table also shows the invaluable contribution of pruning. In all cases, pruning succeeded in computing metaclassifiers with similar or better fraud-detection capabilities, while reducing their size and thus improving their efficiency. We have provided detailed description on the pruning methods and a comparative study between predictive performance and metaclassifier throughput elsewhere.<sup>7</sup>

 ONE LIMITATION OF OUR APPROACH to skewed distributions is the need to run preliminary experiments to determine the desired training distribution based on a defined cost model. This process can be automated, but it is unavoidable because the desired distribution highly depends on the cost model and the learning algorithm. Currently, for simplicity reasons, all the base learners use the same desired distribution; using an individualized training distribution for each base learner could improve the performance. Furthermore, because thieves also learn and fraud patterns evolve over time, some classifiers are more relevant than others at a particular time. Therefore, an adaptive classifier-selection method is essential. Unlike a monolithic approach of learning one classifier using incremental learning, our modular multiclassifier approach facilitates adaptation over time and removes out-of-date knowledge.

Our experience in credit card fraud detection has also affected other important applications. Encouraged by our results, 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 (such as Ripper) to learn and combine multiple models for different types of intrusions, we have achieved remarkably good success in detecting intrusions.<sup>9</sup> This work, as well as the results reported in this article, demonstrates convincingly that distributed data-mining techniques that combine multiple models produce effective fraud and intrusion detectors. ■

## Acknowledgments

We thank the 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 the NSF (IRI-96-32225 and CDA-96-25374).

## References

1. P. Chan and S. Stolfo, "Metalearning for Multistrategy and Parallel Learning," *Proc. Second Int'l Workshop Multistrategy Learning*, Center for Artificial Intelligence, George Mason Univ., Fairfax, Va., 1993, pp. 150–165.
2. S. Stolfo et al., "JAM: Java Agents for Metalearning over Distributed Databases," *Proc. Third Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1997, pp. 74–81.
3. D. Wolpert, "Stacked Generalization," *Neural Networks*, Vol. 5, 1992, pp. 241–259.
4. P. Chan and S. Stolfo, "Toward Scalable Learning with Nonuniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection," *Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1998, pp. 164–168.
5. A. Prodromidis and S. Stolfo, "Mining Databases with Different Schemas: Integrating Incompatible Classifiers," *Proc. Fourth Intl Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1998, pp. 314–318.
6. Z. Ras, "Answering Nonstandard Queries in Distributed Knowledge-Based Systems," *Rough Sets in Knowledge Discovery, Studies*

*in Fuzziness and Soft Computing*, L. Polkowski and A. Skowron, eds., Physica Verlag, New York, 1998, pp. 98–108.

7. A. Prodromidis and S. Stolfo, "Pruning Meta-classifiers in a Distributed Data Mining System," *Proc. First Nat'l Conf. New Information Technologies*, Editions of New Tech. Athens, 1998, pp. 151–160.
8. D. Margineantu and T. Dietterich, "Pruning Adaptive Boosting," *Proc. 14th Int'l Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 1997, pp. 211–218.
9. W. Lee, S. Stolfo, and K. Mok, "Mining in a Data-Flow Environment: Experience in Network Intrusion Detection," *Proc. Fifth Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1999, pp. 114–124.

**Philip K. Chan** is an assistant professor of computer science at the Florida Institute of Technology. His research interests include scalable adaptive methods, machine learning, data mining, distributed and parallel computing, and intelligent systems. He received his PhD, MS, and BS in computer science from Columbia University, Vanderbilt University, and Southwest Texas State University, respectively. Contact him at Computer Science, Florida Tech, Melbourne, FL 32901; pkc@cs.fit.edu; www.cs.fit.edu/~pkc.

**Wei Fan** is a PhD candidate in computer science at Columbia University. His research interests are in machine learning, data mining, distributed systems, information retrieval, and collaborative filtering. He received his MSc and B.Eng from Tsinghua University and MPhil from Columbia University. Contact him at the Computer Science Dept., Columbia Univ., New York, NY 10027; wfan@cs.columbia.edu; www.cs.columbia.edu/~wfan.

**Andreas Prodromidis** is a director in research and development at iPrivacy. His research interests include data mining, machine learning, electronic commerce, and distributed systems. He received a PhD in computer science from Columbia University and a Diploma in electrical engineering from the National Technical University of Athens. He is a member of the IEEE and AAAI. Contact him at iPrivacy, 599 Lexington Ave., #2300, New York, NY 10022; andreas@iprivacy.com; www.cs.columbia.edu/~andreas.

**Salvatore J. Stolfo** is a professor of computer science at Columbia University and codirector of the USC/ISI and Columbia Center for Applied Research in Digital Government Information Systems. His most recent research has focused on distributed data-mining systems with applications to fraud and intrusion detection in network information systems. He received his PhD from NYU's Courant Institute. Contact him at the Computer Science Dept., Columbia Univ., New York, NY 10027; sal@cs.columbia.edu; www.cs.columbia.edu/sal; www.cs.columbia.edu/~sal.