Learning with Non-uniform Class and Cost Distributions: Effects and a Distributed Multi-classifier Approach

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

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

Abstract

Many factors influence a learning process and the performance of a learned classifier. In this paper we investigate the effects of class distribution in the training set on performance. We also study different methods of measuring performance based on cost models and the performance effects of training class distribution with respect to the different cost models. Observations from these effects help us devise a distributed multi-classifier meta-learning approach to learn in domains with skewed class distributions, non-uniform cost per error, and large amounts of data. One such domain is credit card fraud detection and our empirical results indicate that our approach can significantly reduce loss due to illegitimate transactions.

Introduction

Inductive learning research has been focusing on devising algorithms that generate highly accurate classifiers. Many factors contribute to the success of a learning process and hence the quality of the learned classifier. One factor is the class distribution in the training set. Using the same algorithm, different training class distributions can generate classifiers of different quality. That is, using the "natural" (or "given") class distribution might not yield the most effective classifier. How do we characterize the effects of training class distribution on performance? Furthermore, in some domains the class distribution is highly skewed; for instance, the number of fraudulent cellular phone calls is much smaller that legitimate ones. Using the "natural" (highly skewed) class distribution for training, some learning algorithms might treat the minority class as noise or simply produce a classifier that always predict the majority class. Hence, given a learning algorithm, how do we select a class distribution that can produce the most effective classifier?

The most common measure for evaluating performance is the error rate (or accuracy), which assumes that each mistake is equally important. However, in many real-world domains different types of mistakes or different individual errors have varying severity. For example, in medical diagnosis an error that misses a disease diagnosis could be fatal, while a mistake on diagnosing an illness the patient does not have could be much less serious. How can we characterize the cost of each type of error (or each individual error) and evaluate a learning system based on those characteristics? And how do we generate more effective classifiers based on different error or cost models?

Furthermore, with the wide proliferation of computer automation and rapid advance in communication networks, data have become increasingly abundant. For instance, millions of phone calls are made each day and valuable traffic and customer characteristics can be discovered. How can we efficiently learn from large amounts of data?

Answers to the above questions could help us devise methods to efficaciously and efficiently learn from domains with skewed class distributions, non-uniform cost per error, and massive amounts of data. Our approach is based on creating data subsets with the appropriate class distribution, applying learning algorithms to the subsets independently and in parallel, and integrating to optimize cost performance of the classifiers by learning (meta-learning) from their classification behavior. Empirical results from the credit card fraud detection domain that we have been studying indicate that our approach can reduce loss due to illegitimate transactions in an efficient manner.

The paper is organized as follows. We first investigate the effects of training class distribution on classifier performance. Performance metrics based on different cost models are detailed next. We study the effects of training class distribution in the credit card fraud detection domain. We then describe and evaluate our multi-classifier meta-learning approach. In closing, we discuss related work and summarize our observations and future directions.

Class Distribution during Training

We hypothesized that the distribution of examples with different classifications influences the performance of classifiers generated by learning algorithms. For highly skewed data sets, a learning algorithm could treat the minority class instances as noise or generate classifiers that always predict the majority class. We tested this hypothesis.

Experiments were performed on four learning algorithms (C4.5, CART, RIPPER, and BAYES) and three bimodal (two-class) data sets (Credit Card Fraud, Adult Census, and Protein Coding Regions). We obtained C4.5 (Quinlan 1993) and CART (Breiman *et al.* 1984) as part of the IND package (Buntine & Caruana 1991) from NASA Ames Research Center; both algorithms compute decision trees. RIPPER (Cohen 1995) is a rule learning algorithm and was obtained from W. Cohen. BAYES is a naive Bayesian learning algorithm that is based on computing conditional probabilities using the Bayes Rule as described in (Clark & Niblett 1989). The credit card fraud data set was obtained from the Chase Manhattan Bank and contains half a million transactions from 10/95 to 9/96, about 20% of which are fraudulent (the real distribution is much more skewed (*fortunately*)—the 20:80 distribution is what we were given). The adult census data set (courtesy of R. Kohavi and B. Becker) contains records from the 1994 Census. We use a subset which has 45,000 records, 25% of which has more than \$50K in income. The protein coding region data set (courtesy of Craven and Shavlik (1993)) contains 20,000 records of DNA nucleotide sequences and their binary classifications (*coding* (50%) or *non-coding* (50%)).

In this set of experiments the training class distribution varied from 10% to 90% and 10-fold CV was used. Since the natural distribution is fixed, generating distributions from 10% to 90% is achieved by excluding some instances from training in each CV fold. We also want to fix the training set size for each distribution. Using the fraud data as an example, we describe how we calculate the fixed training set size that allows the distribution to vary from 10% to 90%. Let n be the total number of examples, the training set at each fold has $.9n$ records and hence there are $.2 \times .9n$ fraudulent records (20% are fraudulent). To allow up to 90% of fraudulent records, the training set size is $\frac{.2\times.9n}{.9}$, which is $.2n$. That is, at each fold, only $.2n$ records out of $.9n$ records are used for training and $.1n$ records are for testing and has the "natural" 20:80 distribution.

In Figure 1 each plot depicts the results from a learning algorithm and a data set. The x-axis represents the percentage of minority class in the training set. Considering:

[Number of instances]	Actual Positive (fraudulent)	Actual Negative (legitimate)
Predicted Positive (fraudulent)	True Positive (*Hit*) [*a*]	False Positive (*False Alarm*) [*b*]
Predicted Negative (legitimate)	False Negative (*Miss*) [*c*]	True Negative (*Normal*) [*d*]

the false-negative rate ($\mathcal{FN}$) is defined as $\frac{c}{a+c}$ and the false-positive rate ($\mathcal{FP}$) is defined as $\frac{b}{b+d}$. The y-axis indicates $\mathcal{FN}$, $\mathcal{FP}$, $\mathcal{FN}+\mathcal{FP}$, and the error rate (1 - accuracy).

As expected, our empirical results demonstrate that the error rates vary with the class distributions in the training set. In the fraud domain (only 29K records from Jan 96 were used) the error rate generally increases with the amount of fraud (minority class) in the training set, but minimizes at around 20% fraud (natural percentage). Similarly, in the census domain the error rate generally increases with the amount of minority class instances, but minimizes at around 20% (close to the natural percentage). In the protein domain, which has the same number of instances in each class, the error rate minimizes at around 50% (natural percentage). That is, from our experiments, if the error rate is the measurement of performance, using the natural distribution in training would generally yield the most accurate classifiers. This is intrinsic in the design of most learning algorithms since error rate is commonly used to measure performance and tune algorithms. (Note that in the fraud and census domains BAYES is not quite sensitive to the varying class distributions—the reasons have yet to be determined; we suspect the way we are treating attributes with real values in BAYES is not appropriate for the fraud domain.)

Is error rate (or accuracy) always an appropriate metric for performance evaluation? If the minority class percentage is 10% in training and the learned classifier always predicts the majority class, it has an error rate of 10%, which is quite low. However, the classifier does not carry much information. In the fraud domain, this translates to predicting all transactions to be legitimate and missing all the fraudulent ones (which is the same as not using fraud detection). In order words, the false-positive (or false-alarm) rate is zero, but the false-negative (or miss) rate is one. (For the rest of the paper, let the minority class be positive.) In Figure 1, as expected, the false-negative rate decreases when the negative (minority) percentage increases. That is, increasing the number of minority instances in training produces fewer errors on the minority instances. Conversely, the false-positive rate has the opposite trend. In the fraud domain BAYES is quite insensitive to the class distributions—the FN rate remains high, the FP rate stays low, and the error rate remains at around .2. This suggests that, in this case, the BAYES classifier tends to predict the majority class.

Figure 2 plots $\mathcal{FN}$ versus $\mathcal{FP}$ in the three domains and each curve, composed of data points from different distributions, represents a learning algorithm. These curves are the same as the ROC curves (Provost & Fawcett 1997) since $\mathcal{TP}$ is $1-\mathcal{FN}$ (we chose to use $\mathcal{FN}$ because $\mathcal{TP}$ is not used for our discussion in this paper). The plot allows us to compare the $\mathcal{FN}$ and $\mathcal{FP}$ performance of different algorithms. The ideal classifier would appear at the lower left corner (0,0). In the fraud domain we observe that BAYES is less sensitive to training class distributions since its curve is shorter and CART generally performs better than the others since its curve is closer to the lower left corner. BAYES is also less sensitive to distributions in the census domain, but the curves are too close together to delineate. Moreover, in the fraud domain, just by varying

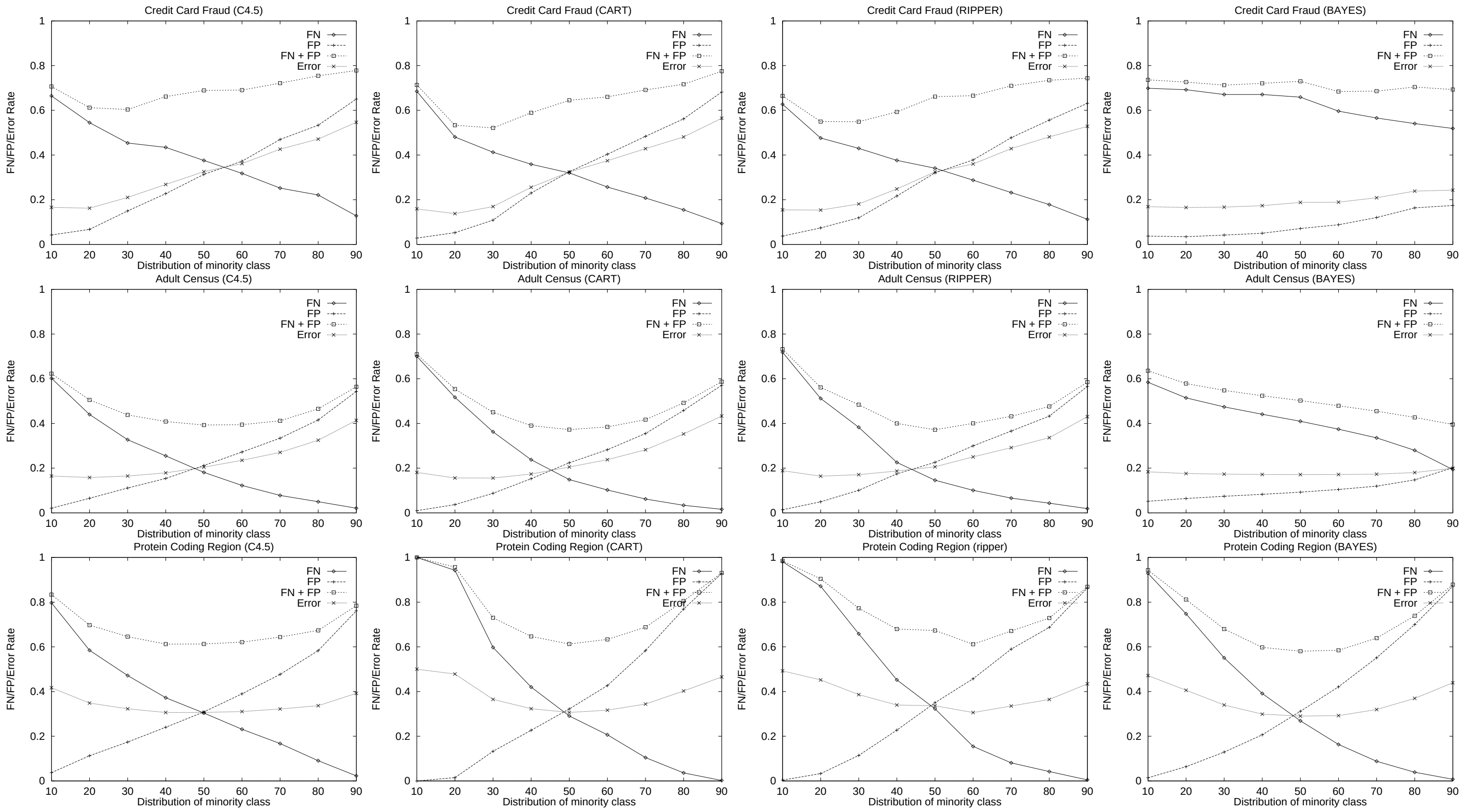

Figure 1: Percentage of minority class in training vs. $\mathcal{FN}/\mathcal{FP}/\mathcal{FN}+\mathcal{FP}$/error rate

the class distribution, a lower $\mathcal{FN}$ is harder to achieve than a lower $\mathcal{FP}$ since more data points "bundle" on the left than at the bottom.

An alternative metric to error rate is the sum of $\mathcal{FN}$ and $\mathcal{FP}$. This sum considers both types of errors. From Figure 1, in the fraud domain, except for BAYES, the sum minimizes at around 30% (close to the natural percentage). In the protein domain, the sum minimizes at around 50% (natural percentage). More interestingly, in the census domain, the sum minimizes at 50%, which is not the natural distribution. In other words, if the sum of $\mathcal{FN}$ and $\mathcal{FP}$ is used as the performance metric, using the natural distribution might not yield the best classifier.

Next we discuss how the error rate, the sum of $\mathcal{FN}$ and $\mathcal{FP}$, and some other performance metrics can be unified in a set of cost models.

Cost Models

The regular error rate measures the percentage of incorrectly classified instances and has the implicit assumption that each error is equally important. To consider the different costs of false-negative's and false-positive's, we define the *average cost* (or *expected cost*) function as:

$$AverageCost = P(P) \times Cost(FN) \times \mathcal{FN} + P(N) \times Cost(FP) \times \mathcal{FP}, \tag{1}$$

where $P(P)$ and $P(N)$ are the probabilities of positive's and negative's, $Cost(FN)$ and $Cost(FP)$ are the respective costs of a false-negative and a false-positive, $\mathcal{FN}$ and $\mathcal{FP}$ are the respective FN and FP rates. This function was also defined by Provost and Fawcett (1997). We further define the *cost ratio* as:

$$CostRatio = \frac{Cost(FN)}{Cost(FP)} \tag{2}$$

Three cases can be derived from Equation 1:

1. **Error rate**:

$$ErrorRate = P(P) \times \mathcal{FN} + P(N) \times \mathcal{FP} \tag{3}$$

From Equations 1 and 3:

$$P(P) \times Cost(FN) = P(P), P(N) \times Cost(FP) = P(N)$$

$$\Rightarrow Cost(FN) = Cost(FP) = 1$$

$$\Rightarrow \frac{Cost(FN)}{Cost(FP)} = 1$$

That is, error rate assumes each error incurs the same cost.

2. $\mathcal{FN} + \mathcal{FP}$:

$$\mathcal{FN} + \mathcal{FP} \tag{4}$$

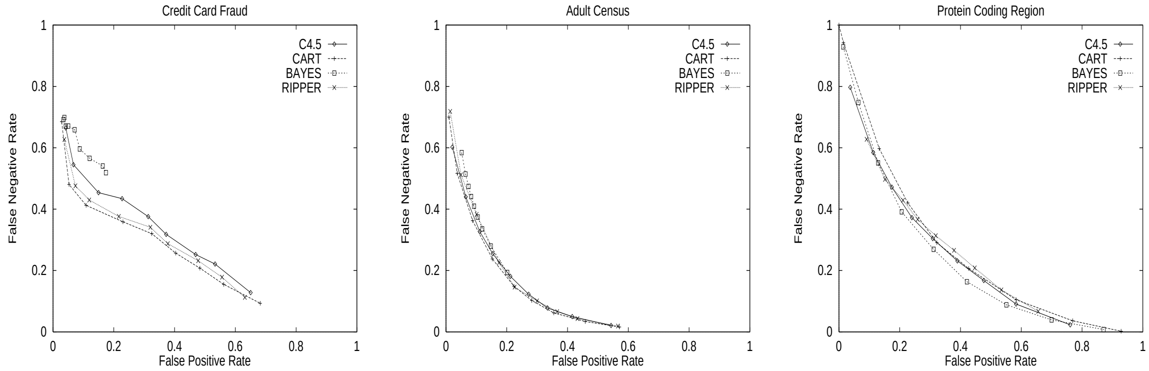

Figure 2: $\mathcal{FN}$ versus $\mathcal{FP}$

From Equations 1 and 4:

$$P(P) \times Cost(FN) = P(N) \times Cost(FP) = 1$$

$$\Rightarrow \frac{Cost(FN)}{Cost(FP)} = \frac{P(N)}{P(P)}$$

That is, the relative cost of each type of error is determined by the relative prior class distribution. Since $\frac{P(N)}{P(P)}$ is 4 in the credit card domain, $\mathcal{FN} + \mathcal{FP}$ implicitly assumes that the cost ratio is 4.

3. **Weighted** $\mathcal{FN} + \mathcal{FP}$:

$$weighted\mathcal{FN} + \mathcal{FP} = w_{FN} \times \mathcal{FN} + w_{FP} \times \mathcal{FP} \quad (5)$$

where w_{FN} and w_{FP} are the weights on a false-negative and a false-positive respectively. From Equations 1 and 5:

$$P(P) \times Cost(FN) = w_{FN}, P(N) \times Cost(FP) = w_{FP}$$

$$\Rightarrow \frac{w_{FN}}{w_{FP}} = \frac{P(P) \times Cost(FN)}{P(N) \times Cost(FP)}$$

$$\Rightarrow \frac{Cost(FN)}{Cost(FP)} = \frac{w_{FN}}{w_{FP}} \times \frac{P(N)}{P(P)}$$

That is, the relative cost of each type of error is determined by the relative prior class distribution multiplied by a weight ratio. Therefore, in the credit card domain $weighted\mathcal{FN} + \mathcal{FP}$ assumes that the cost ratio is 4 $\times$ the weight ratio ($\frac{w_{FN}}{w_{FP}}$).

Experiments were performed to observe the effects of training class distributions on cost measured by the different models described in the previous section. As we learned from the previous experiments, cost varies with the class distribution used in training and the classifier with the lowest cost might not be trained from the "natural" class distribution in the data.

Figure 3 depicts the effects of training class distribution on $weighted\mathcal{FN} + \mathcal{FP}$ using C4.5 on three data sets (due to space limitation, the plots for the other algorithms are not shown). From the figure, we observe that with the different cost models, the lowest-cost classifiers could be generated from training sets with class distributions different from the natural distribution. For instance, in the fraud domain, when the weight ratio ($\frac{w_{FN}}{w_{FP}}$) increases, $weighted\mathcal{FN} + \mathcal{FP}$ minimizes at a larger percentage of minority instances (positive's or fraudulent transactions). This is expected since a larger percentage of positives reduces the FN rate, which in turn lessens the effect of a heavier penalty (w_{FN}) on the FN's. In fact, if the weight ratio is higher than 2 (or 8 in cost ratio), $weighted\mathcal{FN} + \mathcal{FP}$ is minimized when the minority percentage is 90%. Using a similar analysis, 90% is most effective when the weight ratio is larger than 5 (5 in cost ratio) in the protein domain and larger than 5 (15 in cost ratio) in the census domain.

Cost models in the Credit Card Fraud Domain

Although the average cost equation (Eq. 1) allows different costs for different types of errors (false-negative's and false-positive's), it assumes each type of error incurs the same cost. However, due to the different dollar amount of each credit card transaction and other factors, the cost of failing to detect different fraudulent transactions is not the same. Hence we define:

$$AverageAggregateCost = \frac{1}{n} \sum_{i}^{n} Cost(i) \qquad (6)$$

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

After consulting with a bank representative, we settled on a simplified cost model (the cost model used by the bank is more complex and is still evolving). 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 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:

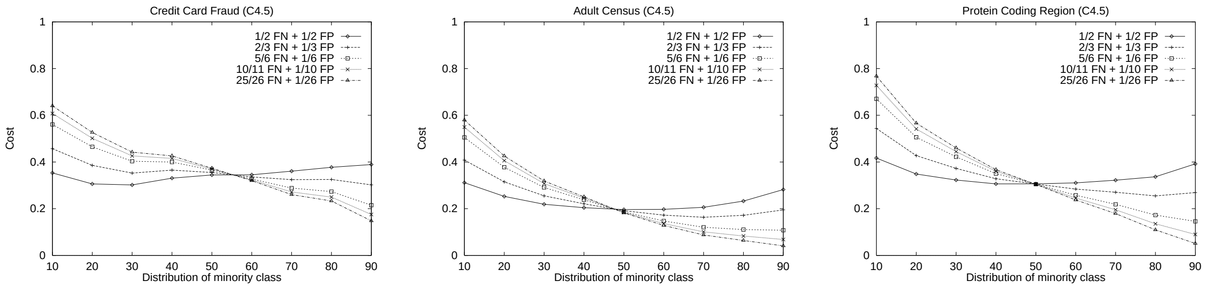

Figure 3: Percentage of minority class in training vs. weighted $\mathcal{FN} + \mathcal{FP}$

$Cost(FN)$	$tranamt$
$Cost(FP)$	$overhead$ if $tranamt > overhead$ or 0 if $tranamt \leq overhead$
$Cost(TP)$	$overhead$ if $tranamt > overhead$ or $tranamt$ if $tranamt \leq overhead$
$Cost(TN)$	0

where *tranamt* is the amount of a credit card transaction. Based on this cost model, we next study the effects of training class distributions on performance.

Experiments on the Credit Card Cost Model

Experiments were performed to study the effects of training class distributions on the credit card cost model. We use data from the first 10 months (10/95 - 7/96) for training and the 12th month (9/96) for testing. In order to vary the fraud distribution from 10% to 90% for each month, we limit the size of the training sets to 6,400 transactions, which are sampled randomly without replacement. The results are plotted in Figure 4. Each data point is an average of 10 classifiers, each of which generated from a separate month. Each curve represents a different amount of overhead.

From Figure 4, as expected, the larger overhead leads to higher cost. More importantly, we observe that when the overhead is smaller, the cost minimizes at a larger percentage of fraudulent transactions (minority class) in the training set. When the overhead is smaller, the bank can afford to send larger number of transactions for investigation. That is, the bank can tolerate more false-alarms (a higher FP rate) and aim for fewer misses (a lower FN rate), which can be achieved by a larger percentage of positive's (as we discussed in the section on the effects of class distributions). Conversely, if the overhead is larger, the bank should aim for fewer false-alarms (a lower FP rate) and tolerate more misses (a higher FN rate), which can be obtained by a smaller percentage of positive's. (Note that, at some point, the overhead can be large enough making fraud detection economically unattractive.)

The test set (from 9/96) has 40,038 transactions and 17.5% of them are fraudulent. If fraud detection is not available, on the average, \$36.96 is lost per transaction. Table 1 shows the maximum savings of each algorithm with the most effective fraud percentage. *Cost* is the dollars lost per transaction; *fraud%* denotes the most

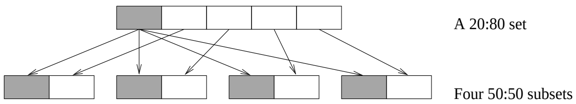

Figure 5: Generating four 50:50 data subsets from a 20:80 data set

effective fraud percentage for training; *%saved* represents the percentage of savings from the average loss of \$36.96; *\$saved* shows the total dollars saved for the month (9/96).

BAYES performed relatively poor (as observed in earlier experiments and discussed earlier) and is excluded from the following discussion. Considering the amount of overhead ranged from \$50 to \$100, the learning algorithms we used generally achieved at least 20% in savings or at least \$300K in savings. With an overhead of \$75 or less, at least half a million dollars in savings can be attained. More importantly, maximum savings were achieved 7 out of 9 times when the fraud percentage used in training is 50%. Since the natural distribution is 20:80, one way to achieve a 50:50 distribution is to ignore 75% of the legitimate transactions (or 60% of all the transactions), as we did in the experiments above. The following section discusses an approach that utilizes all the data.

A Distributed Multi-classifier Meta-learning Approach to Non-uniform Distributions

As we discussed earlier, using the natural class distribution might not yield the most effective classifiers, particularly when the distribution is highly skewed. Given a skewed distribution, we would like to generate the desired distribution without removing any data. Our approach is to create data subsets with the desired distribution, generate classifiers from the subsets, and integrate them by learning (meta-learning) from their classification behavior. For example, if the natural skewed distribution is 20:80 and the desired distribution 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. Figure 5 illustrates this

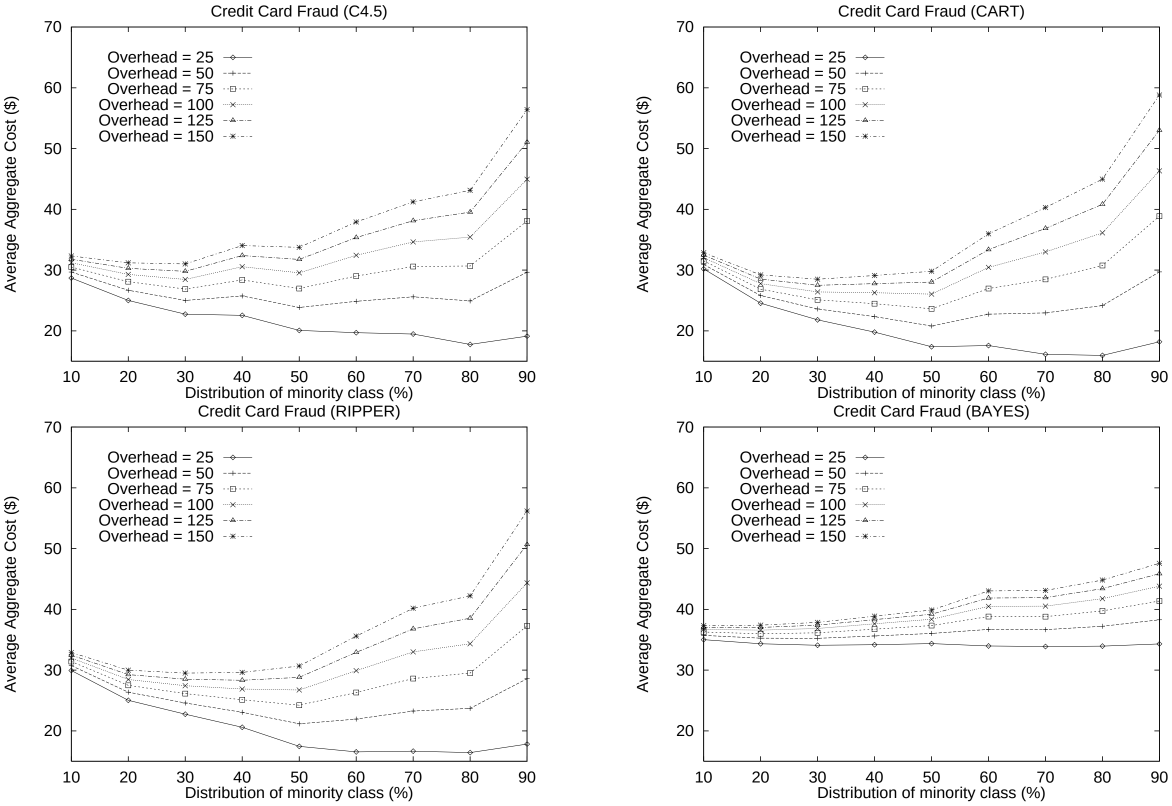

Figure 4: Training distribution vs. the credit card fraud cost model

Table 1: Cost and saving in the credit card fraud domain

Learning	Overhead = $50				Overhead = $75				Overhead = $100			
Alg.	Cost	fraud%	%saved	$saved	Cost	fraud%	%saved	$saved	Cost	fraud%	%saved	$saved
C4.5	23.85	50%	35%	525K	26.88	30%	27%	404K	28.46	30%	23%	341K
CART	20.80	50%	44%	647K	23.64	50%	36%	534K	26.05	50%	30%	437K
RIPPER	21.16	50%	43%	632K	24.23	50%	34%	510K	26.73	50%	28%	409K
BAYES	35.23	30%	5%	69K	35.99	20%	3%	39K	36.58	20%	1%	15K

process.

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

The next step is to apply a learning algorithm(s) to each of the subsets. Since the subsets are independent, 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 non-linear-time learning algorithms.

The generated classifiers are combined by learning (meta-learning) from their classification behavior. Several meta-learning strategies are described in (Chan & Stolfo 1993). To simplify our discussion, we only describe the *class-combiner* (or *stacking* (Wolpert 1992)) 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 a meta-classifier. For integrating subsets, class-combiner can be more effective than the voting-based techniques (Chan & Stolfo 1995).

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. We used

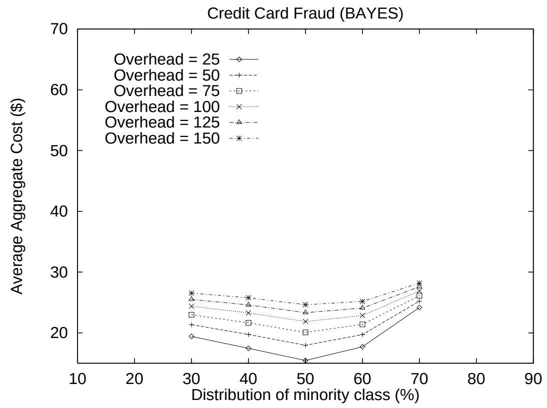

Figure 6: Training distribution vs. the credit card fraud cost model

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. Based on the empirical results from the effects of class distributions, 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. BAYES was used to train the meta-classifier.

Results from different amounts of overhead are plotted in Figure 6. 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 amount. This is different from the results obtained from previous experiments when meta-learning was not used.

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

Table 2 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 from Table 1), class-combiner using the natural distribution (32 base classifiers—8 months $\times$ 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 try to 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 throwing away 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.

Discussion and Concluding Remarks

The credit card fraud detection domain presents a number of challenging issues to machine learning—it has skewed class distributions, non-uniform cost per error, and large amounts of data, each of which has not been widely studied in the machine learning research community. Our approach attempts to address all three issues.

In Dec 96 Fawcett (1996) summarized the responses to his inquiry on learning with skewed class distributions. The number of responses was amazingly few given skewed distributions are not rare in the real world. Kubat and Matwin (1997) acknowledged the performance degradation effects of skewed class distribution and investigated techniques for removing unnecessary instances from the majority class. Instances that are in the borderline region, noisy, or redundant are candidates for removal. Cardie and Howie (1997) stated that skewed class distributions are "the norm for learning problems in natural language processing (NLP)." In a case-based learning framework, they studied techniques to extract relevant features from previously built decision trees and customize local feature weights for each case retrieval. Our approach keeps all examples and does not change the underlying learning algorithms.

Error rate (or accuracy) is commonly used in evaluating learning algorithms; cost-sensitive learning has not been well investigated. (In a bibliography collected by Turney (1998) on cost-sensitive learning, 34 articles were published between 1974 and 1997—an average of fewer than 1.5 articles per year.) Assuming the errors can be grouped into a few types and each type incurs the same cost, some studies (for example, (Pazzani *et al.* 1994)) proposed algorithms that aim to reduce the total cost. Another line of cost-sensitive research tries to reduce the cost in using a classifier. For instance, some sensing devices are costlier in the robotics domain (Tan 1993) and some medical tests are more expensive in the medical diagnosis domain. Fawcett and Provost (1997) considered non-uniform cost per error in their cellular phone fraud detection task and exhaustively searched (with a fixed increment) for the Linear Threshold Unit's threshold that minimizes the total cost. Without modifying the learning algorithms, our approach handles non-uniform cost per error and is cost-sensitive during the learning process.

Table 2: Cost and savings in the credit card fraud domain using meta-learning

Method	Overhead = \$50				Overhead = \$75				Overhead = \$100			
	Cost	fraud%	%saved	\$saved	Cost	fraud%	%saved	\$saved	Cost	fraud%	%saved	\$saved
Class combiner	17.96	50%	51%	761K	20.07	50%	46%	676K	21.87	50%	41%	604K
Single CART	20.80	50%	44%	647K	23.64	50%	36%	534K	26.05	50%	30%	437K
Class combiner	22.61	natural	39%	575K	23.99	natural	35%	519K	25.20	natural	32%	471K
Avg. single classifier	27.97	natural	24%	360K	29.08	natural	21%	315K	30.02	natural	19%	278K

Until recently, researchers in machine learning have been focused on small data sets. Efficiently learning from large amounts of data has been gaining attention due to the fast growing field of data mining, where data are abundant. Sampling (e.g., (Catlett 1991)) and parallelism (e.g., (Han, Karypis, & Kumar 1997; Provost & Aronis 1996)) are the two main directions in scalable learning. Much of the parallelism work 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.

This study demonstrates that the training class distribution affects the performance of the learned classifier. We also analyze the performance effects of training class distributions using a set of unified cost models. In the credit card fraud detection domain our empirical results indicate that our multi-classifier meta-learning approach using a training class distribution of 50:50 can significantly reduce the amount of loss due to illegitimate transactions. Furthermore, this approach provides a means for efficiently handling learning tasks with skewed class distributions, non-uniform cost per error, and large amounts of data. Not only is our method efficient, it is also scalable to larger amounts of data.

Although downsampling instances of the majority class is not new for handling skewed distributions (Breiman *et al.* 1984), our approach does not discard any data, allows parallelism for processing large amounts of data efficiently, and permits the usage of multiple "off-the-shelf" learning algorithms to increase diversity among the learned base classifiers. Besides, how the data are sampled is based on the cost model, which might dictate downsampling instances of the minority class instead of the majority class. Furthermore, we provide empirical evidence of the effectiveness of our approach.

One limitation of our approach is the need of running preliminary experiments to determine the desired distribution based on a defined cost model. This process can be automated but it is unavoidable since the desired distribution is highly dependent on the cost model and the learning algorithm.

Using four learning algorithms, our approach generates 128 classifiers from a 50:50 class distribution and eight months of data. We might not need to keep all 128 classifiers because some of them could be highly correlated and hence redundant. Also, more classifiers are generated when the data set is larger or additional learning algorithms are incorporated. Metrics for analyzing an ensemble of classifiers (e.g., diversity, correlated error, and coverage) can be used in pruning unnecessary classifiers (Margineantu & Dietterich 1997). Furthermore, the real distribution is more skewed than the 20:80 provided to us. We intend to investigate our approach with more skewed distributions. As with a large overhead, a highly skewed distribution can render fraud detection economically undesirable. More importantly, since 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 multi-classifier approach facilitates adaptation over time and removal of out-of-date knowledge.

Since transactions with amounts smaller than the overhead are generally automatically approved, they might contribute little to evaluating transactions with amounts larger than the overhead. Excluding the small transactions during training reduces the training time, which is particularly beneficial in this domain when large amounts of data are present. In addition, although banks do not share credit card data for fear of losing valuable customers to competitors or violating the customers' privacy, a bank can import "black-box" classifiers from other banks to improve its local performance (Chan & Stolfo 1996).

Acknowledgments

The authors benefited from discussions with Dave Fan, Andreas Prodromidis, and Wenke Lee. Part of this work was performed while the first author was visiting the Computer Science Dept. at Brigham Young University last summer. This research was partially supported by grants from DARPA (F30602-96-1-0311), NSF (IRI-96-32225 & CDA-96-25374), and NYSSTF (423115445).

References

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

Buntine, W., and Caruana, R. 1991. *Introduction to IND and Recursive Partitioning*. NASA Ames Research Center.

Cardie, C., and Howe, N. 1997. Improving minority class prediction using case-specific feature weights. In *Proc. 14th Intl. Conf. Mach. Learning*, 57–65.

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

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

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

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

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

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

Craven, M., and Shavlik, J. 1993. Learning to represent codons: A challenge problem for constructive induction. In *Proc. IJCAI-93*, 1319–1324.

Fawcett, T., and Provost, F. 1997. Adaptive fraud detection. *Data Mining and Knowledge Discovery* 1:291–316.

Fawcett, T. 1996. Learning with skewed class distributions–summary of responses. Machine Learning List, Vol. 8, No. 20.

Han, E.; Karypis, G.; and Kumar, V. 1997. Scalable parallel data mining for association rules. In *Proc ACM-SIGMOD-97*.

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

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

Pazzani, M.; Merz, C.; Murphy, P.; Ali, K.; Hume, T.; and Brunk, C. 1994. Reducing misclassification costs. In *Proc. 11th Intl. Conf. Machine Learning*, 217–225.

Provost, F., and Aronis, J. 1996. Scaling up inductive learning with massive parallelism. *Machine Learning* 23:33–46.

Provost, F., and Fawcett, T. 1997. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, 43–48.

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

Tan, M. 1993. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning* 13:7.

Turney, P. 1998. Cost-sensitive learning bibliography. http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html.

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