Classifying Data Streams with Skewed Class Distributions and Concept Drifts

Classification is an important data analysis tool that uses a model built from historical data to predict class labels for new observations. More and more applications are featuring data streams, rather than finite stored data sets, which are a challenge for traditional classification algorithms. Concept drifts and skewed distributions, two common properties of data stream applications, make the task of learning in streams difficult. The authors aim to develop a new approach to classify skewed data streams that uses an ensemble of models to match the distribution over under-samples of negatives and repeated samples of positives.

Jing Gao, Bolin Ding, and Jiawei Han
University of Illinois at Urbana-Champaign

Wei Fan
IBM T.J. Watson Research Center

Philip S. Yu
University of Illinois at Chicago

Many real applications, such as network traffic monitoring, credit-card fraud detection, and Web click streams, generate continuously arriving data known as data streams.[1,2] In general, knowledge discovery from stream data is challenging; the data are usually massive and arrive with high speed, making either storing all the historical data or scanning it nearly impossible. Moreover, stream data often evolve considerably over time. In many applications, the response time usually must be short. Classification can help people making decisions by predicting class labels for given data on the basis of past records. Researchers have studied classification on stream data extensively in recent years, developing many interesting algorithms.[3,4]

However, most studies on stream classification don't address skewed distributions, which are common in data stream applications. The changes of underlying distributions, called *concept drifts*, present additional challenges. Here, we propose an effective and efficient algorithm to classify data streams with skewed class distributions, employing both sampling and ensemble techniques.

Stream Classification Basics

Figure 1 depicts the classification model in data streams applied to the problem of credit-card fraud detection. As Figure 1a shows, data chunks C_1, C_2, ..., C_i arrive one by one, and each chunk contains positive instances P_i and negative instances Q_i. Suppose C_1, C_2, ..., C_m are labeled. At time stamp m +

1089-7801/08/$25.00 © 2008 IEEE

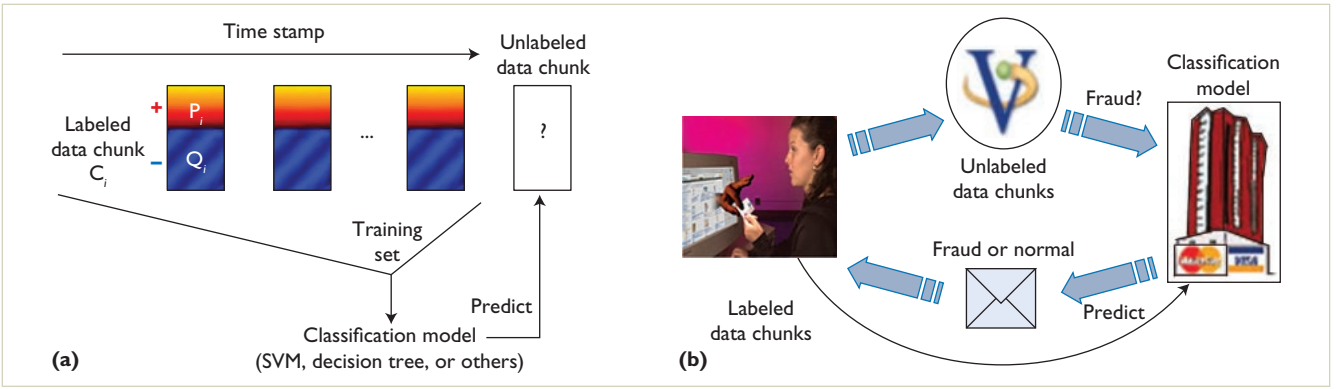

Figure 1. Classification model in data streams. (a) The classification model in data streams (b) applied to credit-card fraud detection can enable the credit-card company to build a model for predicting whether a transaction request is fraudulent.

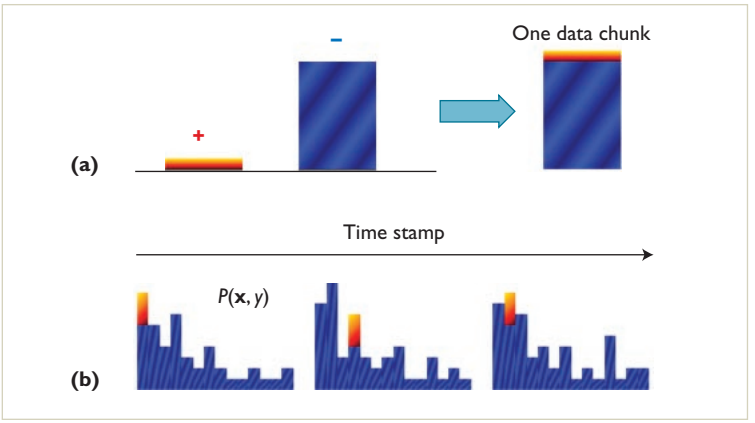

Figure 2. Skewed distributions and concept drifts. (a) In data with skewed distributions, each data chunk has many fewer positive instances than negative instances. (b) In concept drifting, the distribution of instances and the underlying labeling rules might change over time.

1, when an unlabeled data chunk C_{m+1} arrives, the classification model predicts the labels of instances in C_{m+1} on the basis of previously labeled data chunks. When experts give the true class labels of instances in C_{m+1}, the chunk can join the training set, resulting in more and more labeled data chunks. Because of the storage constraints, it's critical to wisely select labeled examples that can represent the current distribution well.

Figure 1b shows an example of stream classification. The credit-card company receives the credit-card transaction requests continuously (unlabeled data chunks). Based on historical records (labeled data chunks), the company builds a classification model to predict whether a transaction request is fraudulent. The prediction informs the decision of whether to accept the request. After some time, the company will send bank statements to the credit-card users, and the users will submit a claim to the company if any transaction is fraudulent. Therefore, the system can add the transactions that received their labels into the set of labeled data chunks to update the classification model.

Most studies on stream mining assume relatively balanced and stable data streams. However, many applications (including the credit-card fraud detection problem) can involve concept-drifting data streams with skewed distributions. As Figure 2a shows, in data with skewed distributions, each data chunk has many fewer positive instances than negative instances. For example, the online credit-card fraud (positive) rate in the US was just 2 percent in 2006. At the same time, the loss functions associated with positive and negative classes are also unbalanced. For example, misclassifying an instance of credit-card fraud as a normal transaction will cost a bank thousands of dollars. Other scenarios of skewed data streams include intrusion detection, airline no-show prediction, and targeted marketing.

Figure 2b illustrates an example of concept drifts. The histogram shows the distribution of feature values. Negative instances are blue, and positive instances are yellow-red. As time elapses, we might observe changes in the joint probability $P(\mathbf{x},y)$, where y is the class label of given feature vector $\mathbf{x}$. In other words, the distributions of instances and the underlying labeling rules might change over time.

Many researchers have addressed separately the deficiencies in existing classification methods on either skewed or concept-drifting data streams (see the "Related Work in Classifica-

Related Work in Classification

Many researchers have tackled skewed class distributions and concept drifts as separate issues that challenge classification models.

Skewed Class Distributions

Handling class imbalance has become an important research problem in recent years because more people have realized that imbalance in class distribution causes suboptimal classification performance.[1] Proposed solutions to this problem include preprocessing data, transforming algorithms, or postprocessing models.

Among the solutions, balancing training set distribution is the most popular approach. Specifically, many sampling algorithms either under-sample majority examples or over-sample minority examples.[2] These methods might improve the prediction accuracy of minority classes, but they have great difficulty with stream applications and their infinite data flow and continuous concept drifts.

Concept Drifts

Classification in stream data has been the focus of extensive study in recent years, resulting in many important algorithms. Much of the previous work aimed to effectively update the classification model when stream data flows in.[3,4] After some period, these methods throw out the old examples or smoothly fade them out by decreasing their weights as time elapses. Alternatively, other researchers explore some sophisticated methods to select old examples to help train a better model, rather than using the most recent data alone.[5–7] These algorithms select either old examples or old models with respect to how well they match the current data. Among these methods, ensemble approaches[5,6] proved highly accurate. Despite these studies, a general framework for dealing with both skewed class distribution and concept drifts in data streams is in great demand.

References

1. N.V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special Issue on Learning from Imbalanced Data Sets," *SIGKDD Explorations*, vol. 6, no. 1, 2004, pp. 1–6.
2. G.E.A.P.A. Batista, R.C. Prati, and M.C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," *SIGKDD Explorations*, vol. 6, no. 1, 2004, pp. 20–29.
3. G. Hulten, L. Spencer, and P. Domingos, "Mining Time-Changing Data Streams," *Proc. Conf. Knowledge Discovery in Data* (KDD 01), ACM Press, 2001, pp. 97–106.
4. C.C. Aggarwal et al., "On Demand Classification of Data Streams," *Proc. Conf. Knowledge Discovery in Data* (KDD 04), ACM Press, 2004, pp. 503–508.
5. H. Wang et al., "Mining Concept-Drifting Data Streams Using Ensemble Classifiers," *Proc. Conf. Knowledge Discovery in Data* (KDD 03), ACM Press, 2003, pp. 226–235.
6. J. Kolter and M. Maloof, "Using Additive Expert Ensembles to Cope with Concept Drift," *Proc. Int'l Conf. Machine Learning* (ICML 05), ACM Press, 2005, pp. 449–456.
7. G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Machine Learning*, vol. 23, 1996, pp. 69–101.

tion" sidebar). It's challenging to predict correctly when the training examples collected from previous time stamps can't represent the current concepts well. Skewed distribution adds complexity to the stream-classification problem. Many classification methods that perform well on balanced data sets would perform poorly on skewed data sets. Traditional classification algorithms aim to minimize classification error rate. Therefore, they might completely ignore the small number of positive examples and predict every example as negative – definitely an undesirable outcome. Our approach attempts to address some of these challenges, but first we'll look at what successful data stream classification requires.

Classifying Data Streams

Successful stream classification faces two important problems: identifying the source of concept drifts and selecting the best type of classification model.

Source of Concept Drifts

Because stream data are evolving, the class labels or the target values we want to predict might change over time, an evolution called concept drifts. Formally, concept drifts are changes in the joint probability $P(\mathbf{x},y) = P(y|\mathbf{x}) \cdot P(\mathbf{x})$, where y is the class label of feature vector $\mathbf{x}$. The formula comprises feature probability $P(\mathbf{x})$ and class label conditional probability $P(y|\mathbf{x})$, so the change in joint probability is better understood through the changes in either of these two components. So, three possible sources generate concept drifts:

- *Feature change.* $P(\mathbf{x})$ changes, but $P(y|\mathbf{x})$ remains the same. In this case, some previously infrequent feature vectors become more frequent, and vice versa. For example, during some period, a particular type of credit-card transaction might appear more frequently.
- *Conditional change.* $P(\mathbf{x})$ remains the same, but $P(y|\mathbf{x})$ changes. In this case, the rele-

vance of features and class labels changes. For example, the feature of a fraud might change from time to time.
- *Dual change*. Both $P(\mathbf{x})$ and $P(y|\mathbf{x})$ change (a combination of feature change and conditional change).

It's unlikely that stream data's underlying distribution remains unchanged. However, predicting when and how the concept changes is difficult. When any of these three changes occurs, a decrease in classification accuracy usually occurs because the training data the model is built on would be carrying out-of-date concepts. In reality, the distribution usually evolves gradually, so the most recent data normally carry concepts closer to the current ones. We define "the most recent data" as the data held in memory at the current time among continuously arriving stream data. It's optimal to always update the model according to the most recent data no matter how concepts evolve.

Classification Model Selection

Although many approaches to stream classification exist, certain types of models and outputs produce the best results.

Discriminative model vs. generative model. Discriminative and generative methods are two major categories for stream classification algorithms. Generative methods assume that $P(y|\mathbf{x})$ follows a certain form of distributions, whose parameters are estimated from the training set. Conversely, discriminative methods make few or no assumptions about the true form of $P(y|\mathbf{x})$. Rather, they try to obtain the most discriminative boundaries between classes. When applied to real stream data, discriminative models should be more accurate than generative models.

First, for many stream applications, we have little or no prior knowledge about the data distribution, so predefining a unified form of $P(y|\mathbf{x})$, such as Gaussian distribution, is difficult. Also, the distribution form might evolve as data continuously arrive; we never know when and how it changes. A wrong assumption about the distribution form can lead to generative models' poor performance. Second, the training data might not be balanced, resulting in insufficient examples for the minority class. It's not easy to learn parameters accurately from limited examples for generative models. The generative model likely would over-fit the data and wouldn't generalize well on future unseen examples.

Thus, discriminative methods, which don't depend on any parametric family of distributions and can easily adapt to the distribution change, provide a more favorable solution to stream-classification problems.

Probability estimation vs. label prediction. We can classify test examples in two ways: directly classify them into several categories (that is, predict their class labels) or estimate $P(y|\mathbf{x})$ and choose the best threshold to optimize on some given criteria. We favor the second approach.

First, in many problems, the labels we observe aren't deterministic; they follow a certain distribution $P(y|\mathbf{x})$. An example could apply to several classes with different probabilities. For example, if there's a 30 percent chance of precipitation, then there's a 70 percent chance that no rain falls. In this case, estimating $P(y|\mathbf{x})$ is more reasonable than predicting class labels. Second, probability estimates provide some information about classification uncertainties, which can be useful in decision making. We would have high confidence in the prediction of an example with a 99 percent probability estimate, but we wouldn't be sure about the prediction of an example with a 50 percent probability estimate.

Overall, when classifying concept-drifting data streams, we prefer discriminative models, which assign labels to test examples on the basis of probability estimates $P(y|\mathbf{x})$. Although the observations in this section apply to many stream classification situations, we extend these ideas to design and evaluate a general framework for classifying skewed concept-drifting data streams. An ensemble of multiple models is a discriminative classifier, even if the base models are from the generative category. This is true because ensemble learning's focus is to learn the decision boundary, not the model's parameters. The proposed method's outputs are probability estimates of $P(y|\mathbf{x})$.

Stream Ensemble Framework

We propose a simple strategy that can effectively classify data streams with skewed distributions. In stream applications, the incoming data stream arrives in sequential chunks, C_1, C_2, ...,

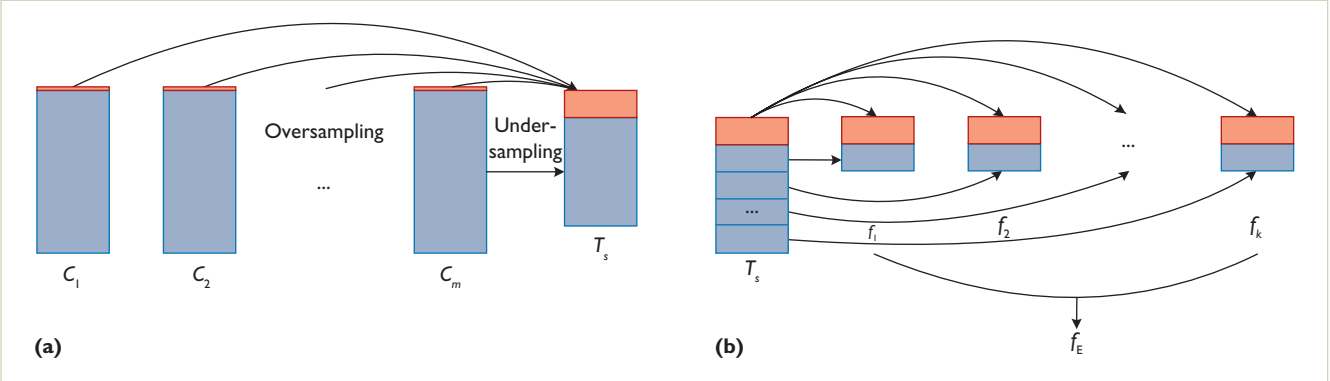

Figure 3. Stream ensemble framework. In the sampling method, (a) we select the positive examples in all the data chunks (over-sampling) and a subset of the negative examples in the most recent data chunk (under-sampling). In the ensemble method, (b) we propagate each positive example to each of the k *sets and each negative example to one of the* k *sets. We train* k *classifiers on the* k *sets and average their outputs.*

C_m, where C_m is the most up-to-date chunk. The data chunk that arrives next is C_{m+1}, and, for simplicity, we denote it as T. The aim is to train a classifier on the basis of the data that has already arrived to predict how likely each example in T falls into one of the predefined categories. We further assume that data come from two classes (positive and negative) and that the number of examples in the negative class is much greater than the number in the positive class. We need only estimate the probability of an example $\mathbf{x}$ belonging to the positive class, $P(+|\mathbf{x})$; that of the negative class would then be $1 - P(+|\mathbf{x})$. For accurate probability estimation, we propose using both sampling and ensemble techniques in the framework.

Sampling

We split each chunk C into two parts: P, which contains positive examples in C, and Q, which contains negative examples in C. The size of P is much smaller than that of Q. Also, note that some uncertainties are associated with classification in certain problems. An example $\mathbf{x}$ could appear in both the positive and negative sets several times. Then, the count of $\mathbf{x}$ in each class will contribute to the calculation of the probability $P(+|\mathbf{x})$.

In stream classification, we can't use all the observed data chunks as the training data. The stream data are huge, so it's usually impossible to store them all. Moreover, a huge training set will slow classification. Also, the concepts that stream data carry are usually changing, thus a model built on the data sets with out-of-date concepts can't make accurate predictions for the new data. A model trained on the most recent data chunk reduces the classification error. However, in the skewed stream-classification problem, the positive examples in the most recent data chunk are far from sufficient to train an accurate model. Therefore, such a classification model will perform poorly on the positive class.

To enhance the set of positive examples, we propose collecting all positive examples and keeping them in the training set. Specifically, the positive examples in the training set are $\{P_1, P_2, ..., P_m\}$. Conversely, we randomly under-sample the negative examples from the last data chunk Q_m to balance the class distribution. As Figure 3a shows, we form a new data set T_s that contains all the positives and sampled negatives.

Ensemble

Instead of training a single model on this selected training set T_s, we propose generating multiple samples from the training set and computing multiple models from these samples. Suppose we generate k data sets from T_s, as Figure 3b illustrates. Each negative example in T_s randomly propagates to exactly one of the k sets, hence the negative examples in the k data sets are completely disjointed. As for positive examples, they propagate to all the k sets because positives are rare and must be used to balance the distribution. Then, we train a series of classifiers on the k data sets, each of which outputs $f_i(\mathbf{x})$, which approximates $P(+|\mathbf{x})$. We use simple averaging to combine the outputs from k models:

$$f_E(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} f_i(\mathbf{x}).$$

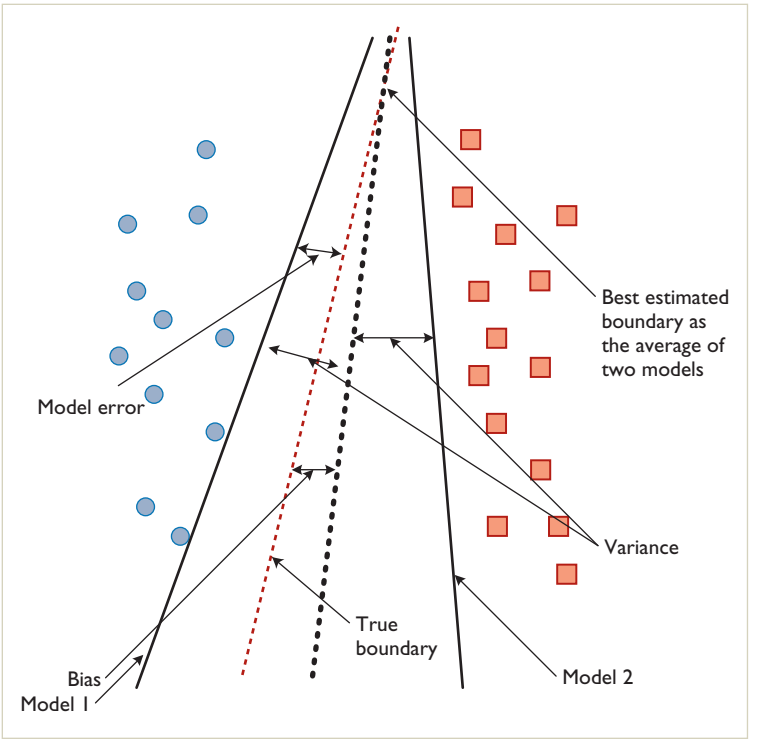

Figure 4. Bias-variance decomposition. The classification error can decompose into bias (the difference between the expected and the true boundaries) and variance (the difference between the single model's boundary and the expected one).

This framework's design considers two facts. First, the ensemble learning methods, which average outputs of multiple models, usually generate higher accuracy compared with a single model. Furthermore, the examples in the ensemble learning training sets should be as uncorrelated as possible so that the base classifiers would make uncorrelated errors that averaging can eliminate.

Infinite Stream

The sampling strategy works when the incoming positive examples are limited. When all positive examples can't be held in memory, we might need to conduct sampling from the pool of positive examples. Given that old data might represent out-of-date concepts, we choose to sample from the l most recent data chunks: C_{m-l+1}, ..., C_m. We can use the following two sampling schemes:

Fixed sampling. From each data chunk C_i, we select a fixed number of positive examples, together with sampled negative examples, to form the training set. In other words, a global sampling rate r exists and has nothing to do with the time stamp. For example, if each data chunk contains N_p positive examples, then each chunk contributes approximately $N_p \cdot r$ positive examples to the training set.

Fading sampling. In this sampling scheme, the sampling rate r is higher when the time stamp is closer to m. Because of memory constraints, this sampling scheme should generate the same number of positive examples in the training set as compared with fixed sampling. Therefore, in the more recent data chunks, we select more positive examples. A simple way to set r is to use the linear equation $r_i = 1 - (m - i + 1)^* s_r$, where i indicates the time stamp, m is the current time stamp, and s_r represents the difference in sampling rate between nearby data chunks.

Because the number of selected positive examples from current data chunks will decrease only when new data chunks flow in, the raw data need not be stored. However, it's necessary to keep the time information of each positive example in the training set. When a new data chunk arrives, we must discard some old positive examples and add new positive examples to the training set.

Framework Analysis

In completing our framework analysis, we looked at error decomposition, error reduction, and efficiency.

Error Decomposition

Researchers have analyzed the error decomposition problem[5,6] in an attempt to explain why ensemble methods reduce classification errors.[7] We follow the decomposition scheme in Kagan Tumer and Joydeep Ghosh's work[5] in the following analysis. We expect that a well-trained classifier could approximate the true $P(+|x)$. The difference between the true and estimated $P(+|x)$ is the classification error that we wish to minimize. This two-class classification problem is equivalent to finding the optimal decision boundary between two classes. So, we're interested in the boundary error b — that is, the error a model makes when discriminating between two classes.

As Figure 4 shows, b can decompose into bias β_b and variance η_b. The bias measures the difference between the expected boundary and the true boundary, whereas the variance mea-

sures the changes in estimated boundary using varied training sets. The expected boundary (dotted line) represents the boundary resulting from averaging the model space. We could use the bias-variance decomposition to explain many basic principles in classification.

Usually, bias represents a classification algorithm's intrinsic ability to characterize the true decision boundary. In stream classification, we might need to update the model frequently on the basis of recent data because the model trained on old data could have a large bias. However, if the sample size is too small, we'll observe a large variance because a slight change in one example might lead to a completely different model, thus causing performance to fluctuate significantly.

This decomposition explains why we keep all the positives and sample negatives from the recent data chunk. Previous studies show that the variance η_b is a dominating factor of the classification error.[5] So, if we consider only η_b and assume β_b is fixed, we can regard η_b as a random noise with zero mean and variance σ_η^2. Then b is normally distributed with mean β_b and variance σ_b^2, where

$$\sigma_b^2 = \frac{\sigma_+^2 + \sigma_-^2}{s^2},$$

s is a constant independent of the trained model, and σ_+^2 and σ_-^2 are variances of classification models associated with positive and negative classes, respectively.

Error Reduction

The proposed sampling approach could reduce the variance σ_b^2 without increasing the bias β_b. If we use only the current data chunk to train the model, the positive examples are so limited that the classifier error would mainly come from the variance. Adding positive examples from previous time slots would reduce the high variance σ_+^2 that insufficient data causes. The negative examples of the training set are from the current data chunk, which we assume is sufficient and reflects the current concept. Therefore, the boundary between the two classes can't be biased much by incorporating old positive examples. Even if the bias β_b increases, the variance reduction is dominant, and the overall generalization accuracy improves.

Using ensemble techniques could further reduce single classifiers' variance. Research has proven that, if the noise error of each of the k classifiers is independent, the ensemble's boundary variance is only $1/k^2$ of the original.[5] This result represents the ideal case in which the base classifiers are completely uncorrelated. When the classifiers are related, the variance could still be reduced as long as we can average out some uncorrelated errors. Because we put different sets of negative examples into k sets, there's a high chance that the errors are uncorrelated.

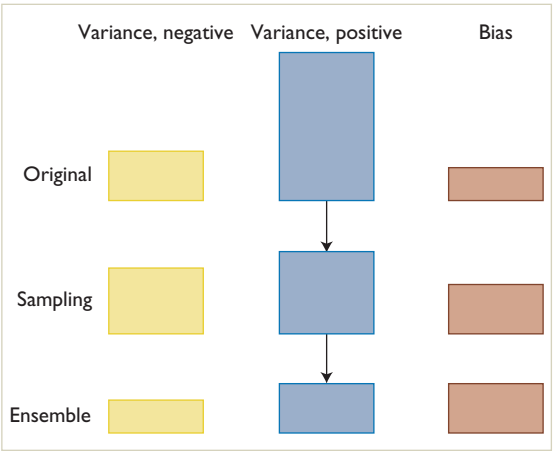

Figure 5. Error reduction. Both sampling and ensemble methods contribute to error reduction in skewed stream classification. (Bar height represents the error rate.)

Figure 5 illustrates this idea. A model's classification error comprises bias and variance on two classes. The bar's height represents the error rate. If the model is trained on the most recent data chunk alone, the variance on the positive class will be high because the model uses only a few positive examples (see the top row). After sampling, we can significantly reduce the variance on the positive class. But the variance on the negative class and the bias might increase slightly because the number of negative examples drops, and some old examples are incorporated (see the middle row). However, in this case, the reduction on the positive variance is dominant. Finally, using ensemble techniques could reduce the variance on both classes even more (see the bottom row). Therefore, both sampling and ensemble techniques contribute to error reduction in skewed stream classification.

Efficiency Analysis

Compared with a single model, our proposed ensemble framework is more efficient. It sounds counterintuitive that training multiple models

Table 1. Data set descriptions.

Data set	Two classes	Number of instances	Number of features	Number of rare class instances	Number of chunks	Number of examples in a chunk
Thyroid1	Class 1 vs. Class 3	6,832	21	166	6	1,138
Thyroid2	Class 2 vs. Class 3	7,034	21	368	6	1,172
Opt	Each class vs. rest	5,620	64	554–572	6	936
Letter	Each class vs. rest	20,000	16	131	6	3,332
Covtype	Class 2 vs. Class 4	28,604	54	274	11	2,599

is more efficient than training a single model. But, in fact, because the multiple models are trained on small subsets of the original training set, the time savings is great.

Suppose the base learner is a decision tree and the data dimensionality is d. The original training set's size is n, and the set is roughly divided into k parts. So, each set in the ensemble learning has size n/k. A decision tree's time complexity would be $O(dn \log(n))$ for the single model and $O(1/k\ dn \log(n/k))$ for each base learner in the ensemble. If these base learners execute sequentially (serial ensembles), the training time for multiple models would be $O(dn \log(n/k))$, which is less than that of a single model.

In general, for example, if a model's training time is expressed as $O(n^a)$, then ensembles of k base models would have a training time of $O(n^a k^{1-a})$. Only the most efficient learning algorithm can have $a = 1$, which indicates a linear scan of the training examples. Usually, $a > 1$, so ensembles are more efficient than a single model. Because each classifier in the ensemble is independent, we can compute them in parallel. The gain in efficiency would be more significant for such parallel ensembles.

Experiments

We conducted thorough experiments on both synthetic and real data sets and found four key results. First, our proposed method would perform well no matter how the concept changes. Second, for a series of real data sets in which the distribution forms and concept changes are both unknown, the simple stream ensemble method can greatly improve prediction accuracy. Third, when the evolution pattern remains stable, the fading sampling method performs better than uniform sampling. Finally, besides gains in accuracy, the ensemble framework also improves efficiency.

Data Collection

Our experiments involved synthetically generated data sets as well as a series of real data sets.

Synthetic data. We generated synthetic data streams with different kinds of concept changes. The data is of the form $(\mathbf{x},y)$, where $\mathbf{x}$ is a multidimensional feature vector, and $y \in \{0,1\}$ is the class label. We then simulated $P(\mathbf{x})$, $P(y|\mathbf{x})$, and their changes. $\mathbf{x}$ follows a Gaussian distribution with mean μ and variance σ. The feature change is simulated through the mean vector's change. Let μ_i change to $\mu_i s_i(1 + t)$ for each data chunk. Specifically, t is between 0 and 1, representing the magnitude of changes. $s_i \in \{-1,1\}$ specifies the direction of changes and could be reversed with a probability of 10 percent.

We generated the decision boundary using function $g(\mathbf{x}) = \sum_{i=1}^{d} a_i x_i x_{d-i+1} - a_0$. If $\mathbf{x}$ satisfies $g(\mathbf{x}) < 0$, it's labeled positive; otherwise, it's labeled negative. Weights a_i $(1 \le i \le d)$ are initialized by random values in the range of [0,1]. We set the value of a_0 so that the number of positive examples is much smaller than that of negative examples. The concept change in $P(y|\mathbf{x})$ is represented by the change in weight a_i, which is changed to $a_i s_i$ $(1 + t)$ for every data chunk. We generate b data chunks of size n for each kind of concept drift: feature change, conditional change, and dual change.

Real data. We used a series of real data sets from the University of California, Irvine's Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.html). Although these data sets don't correspond directly to skewed data mining problems, we can convert them into rare class problems by taking one small class as the rare class and the remaining records, or the biggest remaining class, as the majority class.

For the "thyroid" data set, we selected either Class 1 or Class 2 as the rare class and Class 3 as the majority class. Similarly, for the "covtype" data set, the biggest class (Class 2) is the majority class and the smallest class (Class 4) is the rare class. Data sets "letter" and "opt" are used to recognize 26 letters and 10 digits, respectively. For these two data sets, we simply

chose one class to represent the rare class and collapsed the remaining classes into one majority class. So, from the original data sets, we generated 26 and 10 skewed data sets and averaged the results over these data sets.

To simulate a data stream, we randomly partitioned the data into several chunks with skewed distribution maintained in each chunk. Table 1 summarizes the data sets.

Measures and Baselines

We evaluated the proposed method from two perspectives: Are the probability estimates accurate? Is the classification accurate? We can use many standard measures to evaluate probability estimation's quality. A popular one is mean squared error of the estimated $P(y|\mathbf{x})$ from the true $P(y|\mathbf{x})$. Because in skewed mining problems, rare class is more interesting, we focus on the results of positive class.

In skewed mining problems, classification error isn't a good measure because the examples in the majority class will dominate the result. Thus, the following evaluation metrics are typical: *precision* (detection rate), *recall*, and *false-alarm rate*. Precision in this context is the percentage of correct predictions among all predicted positive examples. Recall is the percentage of correct predicted positive examples among all examples with the true class label "positive." False-alarm rate is the percentage of false predictions among all predicted positive examples. We then used both the receiving operating characteristic (ROC) curve and recall-precision plot to demonstrate the experimental results. The ROC curve represents the trade-off between the detection rate and false-alarm rate, and the recall-precision plot shows the correlation between precision and recall. The ideal case will have 0 percent false alarm rate, 100 percent recall, and 100 percent detection rate.

Given that sampling and ensemble techniques could help reduce the classification error, the baseline methods we compared are the following:

- *No sampling + single model* (*NS*). This method trains with only the current data chunk, which is highly skewed. The method trains a single model on the training set.
- *Sampling + single model* (*SS*). The training set is the same as that used in the proposed ensemble method and is generated by keeping all positive examples seen so far and under-sampling negative examples in the current data chunk. But the method trains only a single model on this training set.
- *Sampling + ensemble* (*SE*). This set denotes our proposed model, which adopts both sampling and ensemble techniques.

In the experiments, the base learners include both parametric and nonparametric classifiers: decision tree, naïve Bayes, and logistic regression. We use the implementation in the Weka package,[8] a publicly available data mining software package.

Empirical Results

Experimental results verify our proposed method's effectiveness and efficiency.

Test on concept-drifting streams. We generated synthetic data streams with many types of concept drifts, including feature, conditional, and dual concept changes. Each data set has 10 dimensions and 11 chunks, each of which contains 1,000 examples. The portion of rare examples makes up 1 percent of each data set. The last chunk in the data set is recognized as the test data, and all other data chunks are for training.

We obtained results by calculating errors of 10 randomly generated data sets. Figures 6a, 6b, and 6c show the results. Clearly, no matter how the concept changes, the proposed method (SE) greatly improves the mean squared error of the positive class. NS performs badly, with error rates around 90 percent. The results indicate that the classifier classifies almost every positive example as a negative because the training set is highly skewed. The error of the positive class drops to about 50 percent after we over-sample positive examples and under-sample negative ones. However, classification variance would still cause a high error rate for the single model.

SE used exactly the same training sets as SS used. However, SE performs much better because the ensemble could reduce the classification variance by averaging the outputs. For our proposed method, the mean squared error on the positive class is usually less than 10 percent. On average, the error decreases around 40 percent after using sampling and further reduces to 10 to 20 percent if we use both sampling and en-

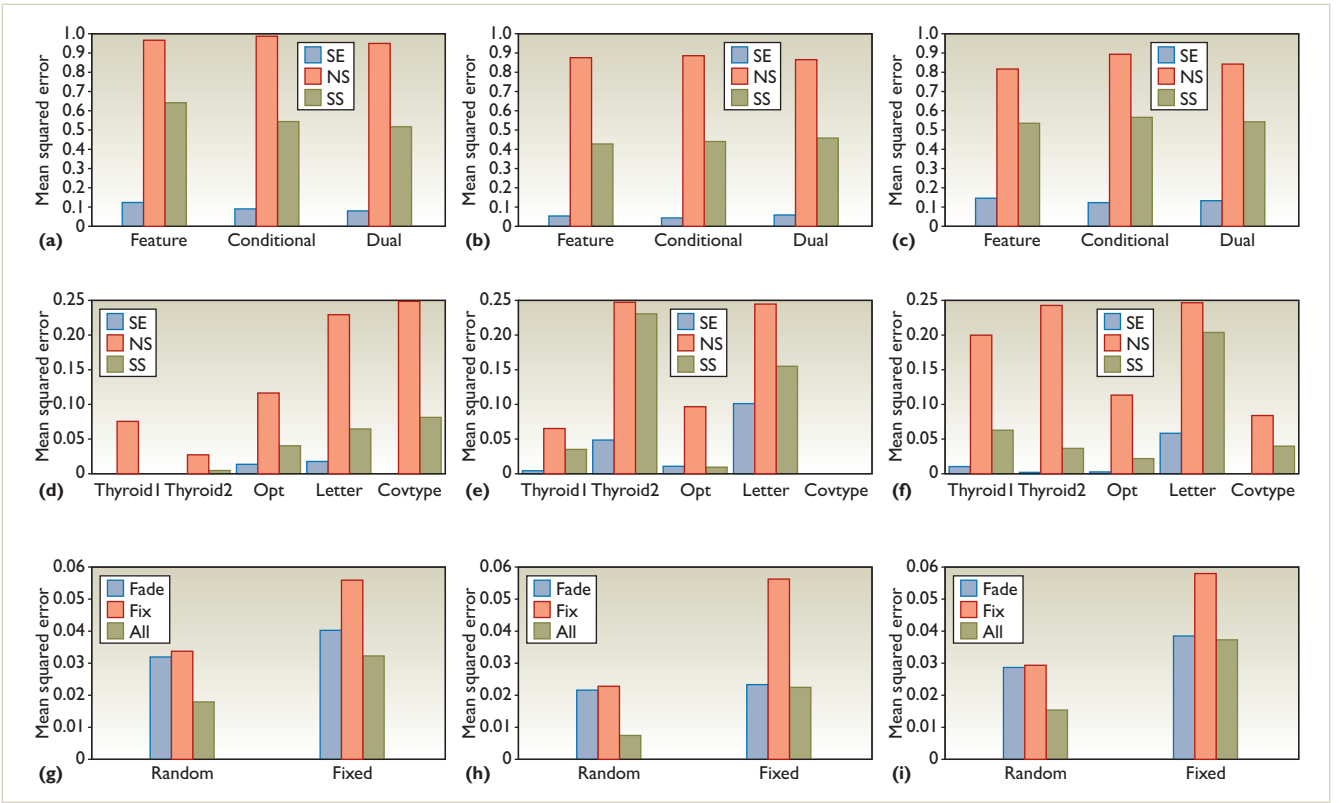

Figure 6. Mean squared error. MSE on synthetic stream data for (a) decision tree, (b) naive Bayes, and (c) logistic regression. MSE on real data for (d) decision tree, (e) naive Bayes, and (f) logistic regression. Our proposed method beats the two baselines in MSE. MSE on two sampling methods for (g) decision tree, (h) naive Bayes, and (i) logistic regression The fading sampling scheme is better than the fixed sampling scheme when we have infinite stream data.

semble. Naïve Bayes had the best performance on the synthetic data set because synthetic data are generated using a Gaussian distribution with a diagonal covariance matrix, which guarantees independence between features.

Test on real data. As Figures 6d, 6e, and 6f show, our proposed method SE consistently outperformed NS and SS on real-life data sets, but the improvements aren't as significant as those on the synthetic data sets. This disparity is probably owing to data sets' learning complexity.

Synthetic data sets' concepts change with every data chunk, which makes them harder to learn. In real data sets, the concept changes aren't intense, so the NS and SS's error rates are typically around 20 percent and less than 10 percent, respectively. This leaves little room for SE to improve. Nevertheless, SE successfully achieves an error rate less than one-tenth of the error rate NS generates most of the time. When NS and SS have high error rates of around 40 and 20 percent, SE could reduce the error rate to 5 percent.

Another interesting observation is that no base learner is globally optimal. Decision trees perform well on "thyroid" and "letter"; naive Bayes is suitable for "covtype"; and "opt" is best classified by logistic regression. This variation demonstrates that these real data sets have different characteristics with different sizes, skewness degrees, and distributions. Such a diverse testbed shows our proposed method's wide capabilities.

Sampling schemes comparison. The fading sampling method samples more examples in the more recent data chunks, whereas the fixed sampling method ignores the temporal information. We applied these sampling techniques on only positive examples; negative examples are sampled from the latest data chunk.

In the two sampling schemes, the number of sampled examples remains the same. We compared the performance of these two methods with the optimal performance in which the system doesn't do any sampling and keeps all the positive examples for training. In reality, this doesn't work because we use sampling when

not all the examples can be stored, and keeping all of them violates the storage requirements. However, in this simulation study, we used this method as a benchmark to test the performance of the two sampling schemes (Fade and Fix versus All).

We generated synthetic data streams with dual concept changes. Each data set has 10 dimensions and 11 chunks, each of which contains 2,500 examples. The portion of rare examples makes up 5 percent of each data set. We obtained the results by calculating errors of 10 randomly generated data sets. In the first experiment, the data evolved randomly (denoted as Random). Specifically, at each time stamp, the data generator randomly selects two dimensions, and their values will either increase or decrease 10 percent. Figures 6g to 6i show that, as expected, keeping all the positive examples achieves the best performance. The fading sampling technique performs slightly better than the fixed sampling technique in this scenario. The data are evolving randomly, so the advantage of using recent examples isn't obvious.

Therefore, we further compared the two methods in the scenario in which the evolving pattern remains stable. In other words, the change pattern is chosen at the first data chunk and occurs in all the following data chunks (denoted as Fixed). Fading sampling outperformed fixed sampling significantly and nearly reached optimal performance. Therefore, when the space isn't large enough to hold all the positive examples, the fading sampling technique is best for generating the most representative set of positive examples.

Model accuracy analysis. This experiment aims to compare model accuracy in terms of detection rate, recall, and false-alarm rates. Figures 7a to 7d show the ROC curves and recall-precision plots of a synthetic data set and one of the real data sets, Opt. The parameter setting is the same as for testing the mean squared error, and the base learner is C4.5, a decision tree classification algorithm. Clearly, in both synthetic and real applications, the proposed method consistently improves precision. The synthetic data set is difficult for the model to learn owing to highly skewed distribution and concept changes, so the precision a single model achieves is extremely low — about 2 percent in the recall range of [0.4, 0.6].

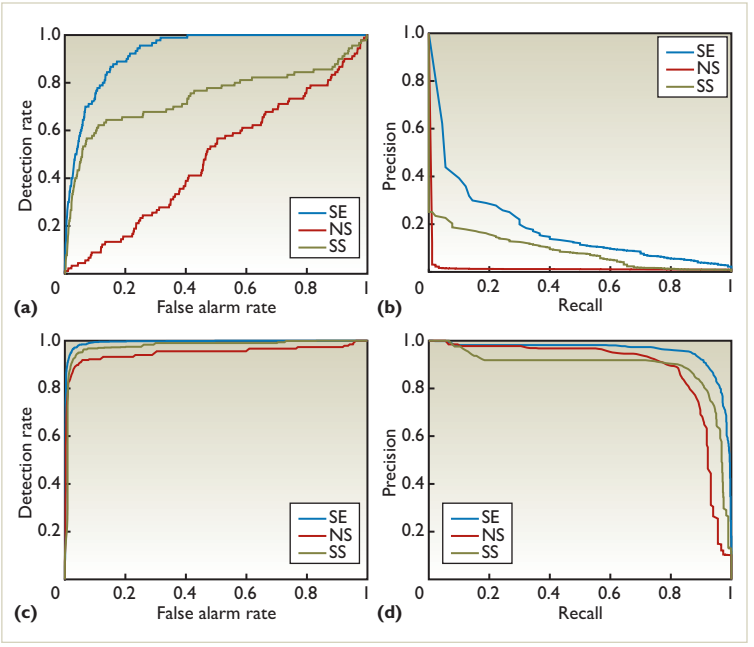

Figure 7. ROC curve and recall-precision plots. (a) ROC curve and (b) recall-precision plots for a synthetic data set. (c) ROC curve and (d) recall-precision plots for the Opt data set. The ideal case will have a 0 percent false alarm rate, 100 percent recall rate, and 100 percent detection rate.

Our proposed method has achieved nearly 10 times the precision of a single model and an ROC curve with an area near 100 percent. It justifies that the sampling and ensemble techniques we use in our proposed method effectively improve the probability estimates and thus lead to a better classification model. The improvements in the Opt data set aren't that significant because the data set isn't sharply skewed (nearly 10 percent of positive examples), so the single model has already obtained a high precision (nearly 90 percent). However, the proposed method succeeds in improving both ROC and recall-precision graphs with a 10 percent increase in precision because the classification variance is reduced.

Chunk size's effect. Chunk size, or the number of examples each chunk contains, is an important factor that can affect stream classification's performance. In real applications, the computation facilities, such as buffer size, determine chunk size. The data sets we used are the synthetic data set with 1 percent positive examples; the base learner is C4.5 classification algorithm. We vary the number of examples each chunk contains from 1,000 to 5,000. According to results in Figure 8, the classification errors of all

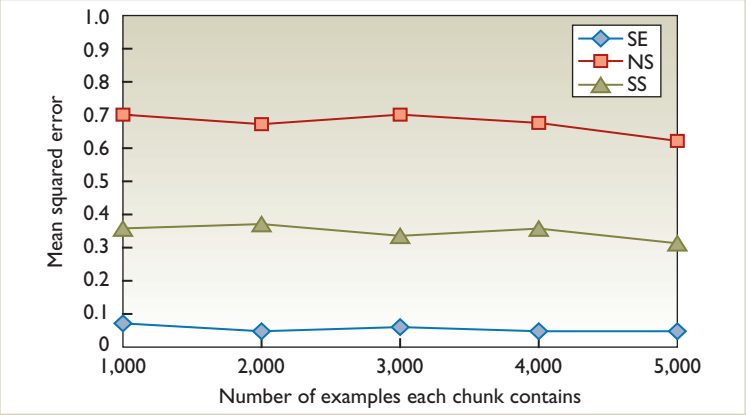

Figure 8. Chunk size's effect. Classification errors decrease as the number of examples each chunk contains increases.

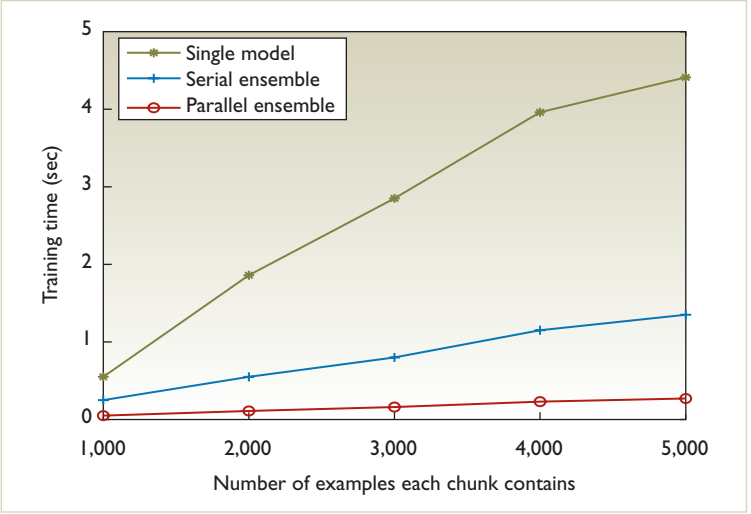

Figure 9. Training time. The ensemble model significantly outperformed the single model.

three methods decrease when chunk size increases because more training examples lead to a reduction in classification errors. SE always outperforms NS and SS when chunk size varies.

Training efficiency. Figure 9 shows the ensemble approach's time complexity compared with a single model. If the base models execute sequentially, the ensemble is referred to as a serial ensemble. If they execute in parallel, it's termed a parallel ensemble. We performed the experiments on synthetic data streams with 10 chunks and five models in the ensemble. As we expected, the ensemble significantly outperformed the corresponding single model. When each chunk has 1,000 examples, the serial ensemble reduces the training time by half. The reduction becomes more significant as chunk size increases. When the training set has a high cardinality, training a single model would cost a large amount of time. The ensemble method, however, divides the large training set into several small sets, thus saving substantial time. The gain in efficiency is more significant if the classification processes execute in parallel. The results verify that our proposed method could improve efficiency and scalability.

To design successful stream classification algorithms, we must understand concept drifts' origin. Also, we'd better choose a discriminative method that outputs accurate probability estimates. Particularly, our research has demonstrated that both sampling and ensemble techniques are effective for improving the classification accuracy on skewed data streams. Through our experiments, we reveal that the proposed method is both accurate and efficient. For the future, we are working on experimenting on the method's performance with respect to the number of base models the ensemble uses. We also want to test the method on various real applications that generate skewed stream data, such as network traffic data and credit-card transaction data.

References

1. B. Babcock et al., "Models and Issues in Data Stream Systems," *Proc. Symp. Principles of Database Systems* (PODS 02), ACM Press, 2002, pp. 1–16.
2. C. Aggarwal, *Data Streams: Models and Algorithms*, Springer, 2007.
3. M.M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining Data Streams: A Review," *ACM SIGMOD Record*, vol. 34, no. 2, 2005, pp. 18–26.
4. S. Muthukrishnan, "Data Streams: Algorithms and Applications," *Proc. ACM/SIAM Symp. Discrete Algorithms* (SODA 03), Soc. for Industrial and Applied Mathematics, 2003, p. 413.
5. K. Tumer and J. Ghosh, "Analysis of Decision Boundaries in Linearly Combined Neural Classifiers," *Pattern Recognition*, vol. 29, no. 2, 1996, pp. 341–348.
6. P. Domingos, "A Unified Bias-Variance Decomposition and Its Applications," *Proc. Int'l Conf. Machine Learning* (ICML 00), Morgan Kaufmann, 2000, pp. 231–238.
7. H. Wang et al., "Mining Concept-Drifting Data Streams Using Ensemble Classifiers," *Proc. Conf. Knowledge Discovery in Data* (KDD 03), ACM Press, 2003, pp. 226–235.
8. I.H. Witten and E. Frank, *Data Mining: Practical Ma-*

chine Learning Tools and Techniques, 2nd ed., Morgan Kaufmann, 2005.

Jing Gao is a PhD student in the Department of Computer Science at the University of Illinois, Urbana-Champaign. Her research interests include data mining and machine learning, particularly transfer learning, mining data streams, and anomaly detection. Gao has an ME from Harbin Institute of Technology, China. She is a student member of the IEEE. Contact her at jinggao3@uiuc.edu.

Bolin Ding is a PhD student in the Department of Computer Science at the University of Illinois, Urbana-Champaign. His research interests include algorithm design for database and data mining problems and algorithmic theory. Ding has an MPhil from Chinese University of Hong Kong. Contact him at bding3@uiuc.edu.

Wei Fan is a researcher at IBM T.J. Watson Research Center. His research interests include risk analysis, high-performance computing, extremely skewed distribution, cost-sensitive learning, data streams, ensemble methods, and commercial data mining systems. Fan has a PhD in computer science from Columbia University. Contact him at weifan@us.ibm.com.

Jiawei Han is a professor in the Department of Computer Science at the University of Illinois, Urbana-Champaign. His research interests include data mining, data warehousing, stream data mining, spatiotemporal and multimedia data mining, biological data mining, social network analysis, text and Web mining, and software bug mining. Han has a PhD in computer science from the University of Wisconsin-Madison. He is an ACM fellow and an IEEE senior member. Contact him at hanj@cs.uiuc.edu.

Philip S. Yu is a professor in the Department of Computer Science at the University of Illinois, Chicago and holds the Wexler Chair in Information and Technology. His research interests include data mining, privacy preserving publishing and mining, data streams, database systems, Internet applications and technologies, multimedia systems, parallel and distributed processing, and performance modeling. Yu has a PhD in electrical engineering from Stanford University. He is a fellow of the ACM and of the IEEE. Contact him at psyu@cs.uic.edu.