On Appropriate Assumptions to Mine Data Streams: Analysis and Practice

Jing Gao† Wei Fan‡ Jiawei Han†

†University of Illinois at Urbana-Champaign
‡IBM T. J. Watson Research Center
jinggao3@uiuc.edu, weifan@us.ibm.com, hanj@cs.uiuc.edu

Abstract

Recent years have witnessed an increasing number of studies in stream mining, which aim at building an accurate model for continuously arriving data. Somehow most existing work makes the implicit assumption that the training data and the yet-to-come testing data are always sampled from the "same distribution", and yet this "same distribution" evolves over time. We demonstrate that this may not be true, and one actually may never know either "how" or "when" the distribution changes. Thus, a model that fits well on the observed distribution can have unsatisfactory accuracy on the incoming data. Practically, one can just assume the bare minimum that learning from observed data is better than both random guessing and always predicting exactly the same class label. Importantly, we formally and experimentally demonstrate the robustness of a model averaging and simple voting-based framework for data streams, particularly when incoming data "continuously follows significantly different" distributions. On a real streaming data, this framework reduces the expected error of baseline models by 60%, and remains the most accurate compared to those baseline models.

1. Problems and Motivations

Classification on stream data has been extensively studied in recent years with many important algorithms developed. Much of the previous work focuses on how to effectively update the classification model when stream data flows in [1, 4, 6]. The old examples can be either thrown away after some period of time or smoothly faded 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 just using the most recent data alone [10, 2, 9, 7, 8]. These algorithms select either old examples or old models with respect to how well they match the current data. Hence, they also implicitly make the assumption that the current training distribution is considerably close to the unknown distribution that produces future data. Among these methods, the weighted ensemble approaches [2, 7, 8, 9] were demonstrated to be highly accurate, when the "stationary distribution assumption" holds true. Formally, we denote the feature vector and class label as $\mathbf{x}$ and y respectively. Data stream could be defined as an infinite sequence of $(\mathbf{x}_i, y_i)$. Training set D and test set T are two sets of sequentially adjacent examples drawn from the data stream. The labels in T are not known during classification process and will only be provided after some period of time. The assumption held by existing algorithms is stated as follows:

Assumption 1 (Shared Distribution - Stationary Distribution) *Training D and test data T are assumed to be generated by the same distribution $P(\mathbf{x}, y) = P(y|\mathbf{x}) \cdot P(\mathbf{x})$ no matter how $P(\mathbf{x}, y)$ evolves as time elapses.*

Given this assumption, one would ask: "what is the difference between stream mining and traditional mining problems?" The most significant difference from traditional "static" learning scenarios is that this shared distribution between training and testing data (abbreviated as "shared distribution" in the rest of paper) evolves from time to time in three different ways [3]: (1) *feature changes*, i.e., the changes of the probability $P(\mathbf{x})$ to encounter an example with feature vector $\mathbf{x}$; (2) *conditional changes*, i.e., the changes of the conditional probability $P(y|\mathbf{x})$ to assign class label y to feature vector $\mathbf{x}$; and (3) *dual changes*, i.e., the changes in both $P(\mathbf{x})$ and $P(y|\mathbf{x})$. An illustration with a real-world intrusion dataset can be found later in this section.

Under the "shared distribution assumption", the fundamental problems that previous works on stream mining focus on are mainly the following areas: 1) How often the shared distribution changes? It could be continuous or periodical, and fast or slow; 2) How much data is collected to mine the "shared distribution"? It could be sufficient, insufficient or "just don't know"; 3) What is this "shared distri-

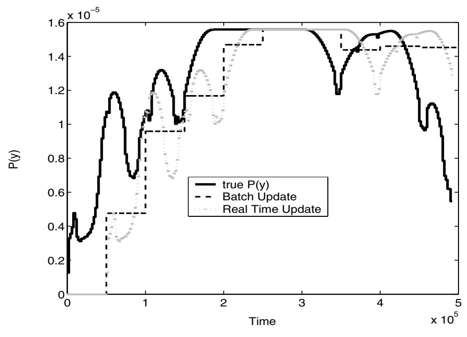


Figure 1. Evolution of $P(y)$

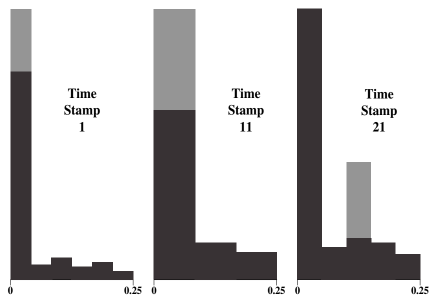


Figure 2. Evolution of $P(\mathbf{x})$ and $P(y|\mathbf{x})$

bution"? It could be balanced or skewed, binary or multi-class, and etc.; 4) How the shared distribution evolves? There could be conditional change, feature change, or dual change; and 5) How to detect the changes in shared distribution? Some methods do not detect them at all and always keep the models up-to-date whereas others only trigger model reconstruction if a change is suspected. Obviously, the validity of some of these problems relies on the "shared distribution assumption", which we challenge below. Interestingly, given "stationary distribution assumption", stream learning would still be effectively the same as traditional learning if the set of training examples collected to mine the "shared distribution" is sufficiently large so that additional examples cannot construct a more accurate model [2].

Realistic Assumption The implicitly held assumption (Assumption 1) may not always be true for data streams. As an example, let us consider the KDDCUP'99 "intrusion detection" dataset that is widely used in the stream mining literature. We plot the evolution on the percentage of intrusions using "averaged shifted histogram (ASH)" in Figure 1. The true probability $P(y)$ to encounter an intrusion is shown in thick solid line. Obviously, $P(y)$ is very volatile. As time elapses, $P(y)$ continues to change and fluctuate. At some period, the change is more significant than others. Except for the flat area between time stamps $2{\times}10^5$ and $3{\times}10^5$, $P(y)$ from the past is always different from that of the future examples. Under "shared distribution" assumption, the training distribution ought to be accurately modeled as the ultimate target. However, it may not precisely match future testing distribution due to continuous change.

The fluctuation in $P(y)$ comes from changes in $P(y|\mathbf{x})$ or $P(\mathbf{x})$. Let + denote intrusions. By definition, $P(y = +) = \frac{\sum P(\mathbf{x},y=+)}{\sum P(\mathbf{x})}$ and $\sum P(\mathbf{x})$ is fixed for a given period, then $P(y) \propto P(\mathbf{x}, y) = P(y|\mathbf{x}) \cdot P(\mathbf{x})$. Thus, the change in $P(y)$ has to come from $P(y|\mathbf{x})$, or $P(\mathbf{x})$, or possibly both $P(y|\mathbf{x})$ and $P(\mathbf{x})$. Unless the dataset is synthesized, one normally does not know which of these three cases is true, either before or after mining. Because of this, a model constructed from the training data may not be highly accurate on the incoming data. This can particularly be an issue if the changes are attributed to conditional probability $P(y|\mathbf{x})$. As follows, we illustrate how $P(\mathbf{x})$ and $P(y|\mathbf{x})$ change using the same intrusion detection example.

Figure 2 shows the histograms of the percentage of intrusions and normal connections given the feature 'srv_diff_host_rate' in three different time periods, where gray represents intrusions and black indicates normal connections. The range of this feature, or the percentage of connections to different hosts, remains within [0,1]. Due to the space limit, we only show the histograms between 0 and 0.25. Most bars between 0.25 and 1 have heights close to 0 and do not reveal much useful information. It is obvious that the distribution of this feature, or visually the relative height of each bar in the histogram representing the percentage of connections, is different among these three time periods. This obviously indicates the change in $P(\mathbf{x})$ as data flows in. In addition, the probability distribution to observe intrusions given this feature is quite different among these three periods. For example, in the first time period, $P(y = +|x \in [0.095, 0.105]) = 0$ but it later jumps to around 0.7 at the last time stamp. In the following, we will discuss how the "shared distribution" assumption affects learning when the actual data evolves in the manner described above. It is worth noting that some stream mining algorithms [10, 6, 2, 9, 7, 12, 8] discuss about the concept drifts in streams and recognize the changes in the distribution that generates the data. However, they still make some assumptions about the forms of concept drifts. For example, most of them assume that the most recent training data is drawn from the distribution which is considerably close to that generates the test data [6, 2, 9, 7, 8].

Depending on when labeled training data becomes available, existing stream classification algorithms belong to two main categories. The first group [1, 4] updates the training distribution as soon as labeled example becomes available and flows in, and at the same time, obsolete examples are either discarded or "weighted" out. Under the "shared distribution" assumption, such method obviously assumes that the distribution of the next moment is the same as those observed data in memory. Visually, it assumes a "shifted" or "delayed" $P(y)$ as the distribution of the future, as shown by the "Real Time Update" curve in Figure 1. To be precise, when either the number of examples kept in memory is not sufficiently large or the fading weights are not set properly, $P(y)$ may not only be shifted but also carry a "different shape" from the plot constructed by average shifted histogram. The second family of stream classification algorithms [2, 9, 7, 8] normally receives labeled data in "chunks", and assumes that the most recent chunk is the closest to the future distribution. Thus, they concentrate on learning from the most recent data accurately as well as some old examples that are similar to the current distribution. Due to the changing $P(y)$, we observe both "shift" and "flattening" of the assumed future distribution, shown in the "Batch Update" curve in Figure 1. "Flattening" is due to chunking and is hard to avoid since labeled data may arrive in chunks. As a summary, for both families of methods, "shifting" is not desirable and ought to be resolved.

In fact, "shift" or "delay" is inevitable under the "shared distribution assumption", since the culprit is the assumption itself: the future data is not known and can change in different ways from the current data, but they are implicitly assumed to be the same. In order to overcome the "delaying" problem, the main question is how one should judiciously use what is known in order to optimally match the unknown future, with the least surprise and disappointment. Existing algorithms have obviously taken the road to accurately match the training distribution with the hope that it will perform well on the future data. However, from the above example as well as detailed experiments on this example in Section 3, they could perform poorly when the future is quite different from the current. By this token, we could see that the commonly held "shared distribution assumption" may not be appropriate, and stream classification algorithms ought to consider situations where training and testing distributions are different. Thus, we take this difference into consideration and suggest a relaxed and realistic assumption as follows:

Assumption 2 (Learnable Assumption) *The training and testing distributions are similar to the degree that the model trained from the training set D has higher accuracy on the test set T than both random guessing and predicting the same class label.*

The core of this new assumption is that it does not assume to know any exact relationship between current training and future test distribution, but simply assume that they are similar in the sense that learning is still useful. As commonly understood, this is the bare minimum for learning. It should be noted that this assumption is made concerning the inductive learning problem. Mining data streams from other perspectives, such as clustering, association mining, may require other appropriate assumptions. All the notations and assumptions we made in the paper are summarized in Figure 3. With the relaxed assumption, we first elaborate the idea that one should only match the training distribution to a certain degree, then we shall provide a straightforward framework that can maximize the chance for models to succeed on future data with different distributions.

- $\mathbf{x}$ is feature vector from feature space X and $P(\mathbf{x})$ is the probability distribution of feature vectors.
- y is the class label from space Y and $P(y)$ is the prior class probability.
- $P(\mathbf{x}, y)$ is the joint probability of having feature vector $\mathbf{x}$ and class label y, and $P(y|\mathbf{x})$ is the conditional probability for $\mathbf{x}$ to have label y.
- Stream data is an infinite sequence of $X - Y$ pairs, $\{(\mathbf{x}_i, y_i)\}$ where the value of y_i is known after a certain time period.
- Since $P(\mathbf{x}, y)$ is evolving in streams, we use $P_t(\mathbf{x}, y)$ to represent the joint distribution over $X - Y$ space at time t.
- Training set D and test set T contain sequentially adjacent examples drawn from the stream data. The true values of y_i in T is not known at the time of learning.
- Training set D is drawn from distribution $P_a(\mathbf{x}, y)$, and test set T is drawn from $P_e(\mathbf{x}, y)$. $a < e$, and $P_a(\mathbf{x}, y)$ and $P_e(\mathbf{x}, y)$ are different.
- $P_a(\mathbf{x}, y)$ and $P_e(\mathbf{x}, y)$ are similar in the sense that the model trained on D and evaluated on T is more accurate than random guessing and fixed prediction.

Figure 3. Notations and Assumptions

2. A Robust and Extensible Framework

In Section 1, we illustrate that when learning from stream data, it is unlikely that training and testing data always come from the same distribution. This phenomenon hurts existing algorithms that are based upon such an assumption. Some stream mining work has investigated the change detection problem [5] or utilized the concept change in model construction [12]. However, since there are only unlabeled examples available in the test data set, the "change detection" could at most detect feature change. It is rather difficult to detect the change in $P(y|\mathbf{x})$ before class labels are given. The moral of the relaxed assumption (Assumption 2) ought to be understood in the way that "strong assumptions are no good for stream mining". To carry this understanding one step further, any single learning method on data streams also makes assumptions one way or the other on how to match the training distribution effectively and still perform well on testing distribution, and these assumptions can also fail for a continuously changing data stream. Instead, we use a naive model averaging based approach that does not depend specifically on any single technique but combines multiple techniques wherever and whenever available. Formally, suppose k models $\{M_1, M_2, \ldots, M_k\}$ are trained (e.g. using different learning algorithms) and each of them outputs an estimated posterior probability $P(y|\mathbf{x}, M_i)$ for each test example $\mathbf{x}$. We use simple averaging to combine the probability outputs, thus $f^A(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} P(y|\mathbf{x}, M_i)$, and its optimality is discussed below.

Performance Guarantee As described above, we generate k models and each model M_i outputs an estimated probability $P(y|\mathbf{x}, M_i)$ for $\mathbf{x}$. For the sake of simplicity, we use M to denote any of the k models M_i and use Θ_M to represent the collection of the k models. Then any base model M's expected mean squared error is the difference between its predicted probability and the true probability integrated over all test examples:

$$Err^M = \sum_{(\mathbf{x},y)\in T} P(\mathbf{x}, y)(P(y|\mathbf{x}) - P(y|\mathbf{x}, M))^2$$
$$= E_{P(\mathbf{x},y)}[P(y|\mathbf{x})^2 - 2P(y|\mathbf{x})P(y|\mathbf{x}, M) + P(y|\mathbf{x}, M)^2]$$

Suppose each model M has probability $P(M)$ on the test set, then the expected error incurred by randomly choosing a base streaming model to do prediction is the above error Err^M integrated over all models:

$$Err^S = \sum_{M\in\Theta_M} \sum_{(\mathbf{x},y)\in T} P(\mathbf{x}, y)(P(y|\mathbf{x}) - P(y|\mathbf{x}, M))^2$$
$$= E_{P(M),P(\mathbf{x},y)}[P(y|\mathbf{x})^2 - 2P(y|\mathbf{x})P(y|\mathbf{x}, M) + P(y|\mathbf{x}, M)^2]$$

It should be noted that the above equation only evaluates the general performances of base streaming models, but the predictions of test examples are not averaged. Now, we come to the analysis of ensemble where the predictions are averaged. As introduced before, we make the following "model averaging" prediction: $f^A(\mathbf{x}) = E_{P(M)}[P(y|\mathbf{x}, M)]$. Then the expected error of this ensemble should be the error integrated over the universe of test examples:

$$\begin{aligned}Err^A =& \sum_{(\mathbf{x},y)\in T} P(\mathbf{x}, y)(P(y|\mathbf{x}) - E_{P(M)}[P(y|\mathbf{x}, M)])^2\\
=& E_{P(\mathbf{x},y)}[P(y|\mathbf{x})^2 - 2P(y|\mathbf{x})E_{P(M)}[P(y|\mathbf{x}, M)]\\
&+E_{P(M)}[P(y|\mathbf{x}, M)]^2]\\
\le& E_{P(\mathbf{x},y)}[P(y|\mathbf{x})^2 - 2P(y|\mathbf{x})E_{P(M)}[P(y|\mathbf{x}, M)]\\
&+E_{P(M)}[P(y|\mathbf{x}, M)^2]]\end{aligned}$$

The inequality holds since $E[f(x))]^2 \le E(f(x)^2]$ (in this case, $E_{P(M)}[P(y|\mathbf{x}, M)]^2 \le E_{P(M)}[P(y|\mathbf{x}, M)^2]$. Therefore, $Err^A \le Err^M$, i.e., probability averaging of multiple models is superior to any base streaming model chosen at random with respect to reduction in expected errors on all possible examples.

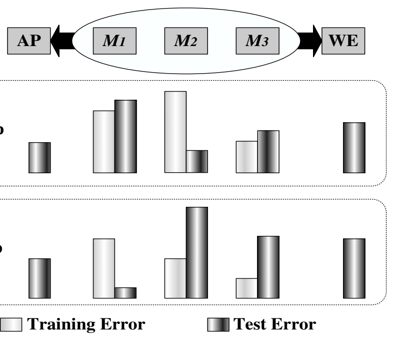


Figure 4. Error Comparison

We are not claiming that model averaging is more accurate than any single model at any given time. As a simple illustration, Figure 4 shows the errors of three models at time A and time B. At a specific time stamp, a single model M that fits current distribution well could have much better performances on test data than other models, e.g., M_2 at time A and M_1 at time B. At this same time stamp, the probability averaging of three models (shown as AP) may not necessarily be more accurate than using a specific model. However, in stream learning problems, it is hard to find a single model that works well on all possible training-test pairs drawn independently from continuously changing distributions. Since it is unknown which single model could be optimal at each and every time stamp, the current practice is to select a method and hope it will perform the best at any time stamp. However, this could be risky. In the above example, the most accurate model M_2 at time stamp A turns out to be the least accurate at time stamp B. On the other hand, the model averaging approach could reduce the probability of surprises and guarantee the most reliable performance. The above analysis formally proves the expected error incurred by randomly choosing a single model is greater than model averaging. Therefore, unless we know exactly which model is always the best, unrealistic in a constantly changing stream environment, we could expect model averaging to have the best expected performance.

Optimality of Uniform Weights The next question is how to decide $P(M)$, or the probability of model M being optimal. The simplest way is to set $P(M^*) = 1$ where M^* is the most accurate model and set other model's probability as 0. This is one of the common practice adopted by some stream mining algorithms where the model itself is fixed but its parameters are re-estimated as labeled data flows in. As discussed above, the expected performance of a single model could be low, when the distribution is continuously evolving. Another more sophisticated approach is introduced in [9], where each model is assigned a weight that is reversely proportional to its error estimated using training data. That is to say, $P(M)$ is higher if the model M incurs less errors when cross-validated using the training data. This weighting scheme is problematic because: 1) the training examples may be insufficient to reflect the true accuracy of model M, thus the weights may not represent the true $P(M)$; and 2) more importantly, the training and testing distributions may not be the same as previous methods have assumed, thus the weights derived from the training data would be essentially inappropriate for the test data. As illustrated in Figure 4, when training and test data have different distributions, $P(M)$ calculated using training data may be off from its true value, thus leading to the unsatisfactory performance of weighted ensemble (denoted as WE) as compared with the simple model averaging (AP). As follows, we formally illustrate why simple averaging with uniform weights beats other non-uniform weighting schemes.

Suppose the weights of k models are $\{w_1, w_2, \ldots, w_k\}$, each of which is from [0,1] and satisfies the constraint $\sum_{i=1}^{k} w_i = 1$. Ideally, the weight of model $M_i (1 \leq i \leq k)$ ought approximate its true probability $P(M_i)$ as well as possible. We use the following measure to evaluate the difference between the assigned weights and the true weights:

$$D(\mathbf{w}) = \sum_{i=1}^{k} (P(M_i) - w_i)^2 \qquad (1)$$

Let Θ_i be the hypothesis space where M_i is drawn, which has a uniform distribution with a constant density C^i. In other words, we don't have any prior knowledge about the optimality of a model for a constantly changing stream. This is a valid assumption since the choice of optimal model is changing with the evolving distribution. The test distribution is somewhat revealed by the training distribution, however, which model fits the distribution the best remains unknown. Another clarification is that $P(M_i) \neq P(M_j) (i \neq j)$ on a specific pair of training and test sets given in time. This means that we cannot have preference for some model over others, since the preference needs to change continuously considering all possible training and test sets in time. The constraint $\sum_{i=1}^{k} P(M_i) = 1$ should also be satisfied. As an example, suppose there are two models, M_1 and M_2. Then $P(M_1)$ and $P(M_2)$ are both uniformly distributed within [0,1]. At one evaluation point, $P(M_1) = 0.8$ and $P(M_2) = 0.2$, but at another time, $P(M_1) = 0.3$ and $P(M_2) = 0.7$. It is clear that both M_1 and M_2 would be preferred at some time but it is unknown when and how this preference is changing. As another example, look at Figure 4 again, it is clearly shown that M_2 and M_1 are the best models with lowest test errors at time A and B respectively. However, since the labels of test examples are not known in advance, we could never know this changing preference before mining.

Integrating the distance measure in Eq. 1 over all possible M_i, we could obtain the expected distance as:

$$\begin{aligned} E[D(\mathbf{w})] &= \sum_{i=1}^{k} \int_{\Theta_i} C^i (P(M_i) - w_i)^2 \mathrm{d}M_i \\ &= \sum_{i=1}^{k} \int_{\Theta_i} C^i (P(M_i)^2 - 2P(M_i) w_i + w_i^2) \mathrm{d}M_i \end{aligned}$$

The aim is to minimize $E[D(\mathbf{w})]$ w.r.t $\mathbf{w}$. Eliminating irrelevant items, the above could be simplified to:

$$E[D(\mathbf{w})] = C_1 - C_2 \sum_{i=1}^{k} w_i + C_3 \sum_{i=1}^{k} w_i^2 \qquad (2)$$

where $\{C_1, C_2, C_3\}$ are positive constants. Since $\sum_{i=1}^{k} w_i = 1$, the problem is transformed to:

$$\text{Minimize} \sum_{i=1}^{k} w_i^2 \text{ Subject to } \sum_{i=1}^{k} w_i = 1 \text{ and } 0 \leq w_i \leq 1$$

The closed form solution to this constrained optimization problem is: $w_i = \frac{1}{k} (1 \leq i \leq k)$. Therefore, when we have no prior knowledge about each model, equal weights are expected to be the closest to true model probabilities on the test data over some period of time, thus giving the best performances on average. This is particularly true in the stream environment where the distribution is continuously changing. As shown in the following experiments, the best model on current data may have bad performances on future data, in other words, $P(M)$ is changing and we could never estimate the true $P(M)$ and when and how it would change. Hence non-uniform weights could easily incur over-fitting, and relying on a particular model should be avoided. Under such circumstances, uniform weights for the models are the best approximate of the true $P(M)$.

3. Experiments

We conduct an extensive performance study using both synthetic and real data sets, where training and testing distributions are explicitly generated differently, to demonstrate the effectiveness of the averaging ensemble against change. As discussed below in detail, this empirical study validates the following claims: 1) ensemble based on model averaging would reduce expected errors compared with single models, thus is more accurate and stable; and 2) previous weighted ensemble approach is less effective than ensemble based on simple voting or probability averaging.

3.1. Experiment Setup

Synthetic Data Generation We describe how to generate synthetic data and simulate its concept changes, rephrased from [3]. **Form of $P(\mathbf{x})$**. $\mathbf{x}$ follows a Gaussian distribution, i.e., $P(\mathbf{x}) \sim N(\mu, \mathbf{\Sigma})$, where μ is the mean vector and $\mathbf{\Sigma}$ is the covariance matrix. The feature change is simulated through the change of the mean vector where μ_i is changed to $\mu_i s_i(1+t)$ for each data chunk. t is between 0 to 1, representing the magnitude of changes, and $s_i \in \{-1, 1\}$ specifies the direction of changes and could be reversed with a probability of 10%. **Form of $P(y|\mathbf{x})$ in deterministic problems**. In binary problems, the boundary between two classes is defined using function $g(\mathbf{x}) = \sum_{i=1}^{d} a_i x_i x_{d-i+1} - a_0$ where $\mathbf{a}$ is the weight vector. Then the examples satisfying $g(\mathbf{x}) < 0$ are labeled positive, whereas other examples are labeled negative. a_i is initialized by a random value in the range of [0,1] and the value of a_0 is set based on the values of r and $\{a_1, \ldots, a_d\}$. In multi-class problems, suppose there are l classes and the count of examples in each class is $\{C_1, C_2, \ldots, C_l\}$. We calculate the value of $g(\mathbf{x})$ for each $\mathbf{x}$ using the definition given in binary problems. All the examples are ranked in ascending order of $g(\mathbf{x})$. Then the top C_1 examples are given class label 1, examples with ranks C_1+1 to C_1+C_2 are assigned to class 2, and so on. In both problems, the concept change is represented by the change in weight a_i, which is changed to $a_i s_i(1+t)$ for every data chunk. The parameters t and s_i are defined in the same way as in the feature change. **Form of $P(y|\mathbf{x})$ in stochastic problems**. We use a sigmoid function to model the posterior distribution of positive class: $P(+|\mathbf{x}) = 1/(1+\exp(g(\mathbf{x})))$. The concept changes are also realized by the changes of weights as illustrated in the deterministic scenario.

The distribution within a data chunk is unchanged whereas between data chunks, the following changes may occur: 1) each data chunk could either be deterministic or stochastic (in binary problem); 2) in each chunk, the Gaussian distribution of the feature values may either have diagonal variance matrix or non-diagonal one; 3) either one of the three concept changes (feature change, conditional change and dual change) may occur; 4) the number of dimensions involved in the concept change is a random number from 2 to 6; and 5) the magnitude of change in each dimension is randomly sampled from $\{10\%, 20\%, \ldots, 50\%\}$. Since lots of random factors are incorporated into the simulated concept change, it is guaranteed that training and testing distributions are different and evolving quickly.

Real-World Data Sets We test several models on KDD Cup'99 intrusion detection data set, which forms a real data stream. This data set consists of a series of TCP connection records, each of which can either correspond to a normal connection or an intrusion. We construct three data streams from the 10% subset of this data set: **Shuffling**. Randomly shuffle the data and partition it into 50 chunks with varying chunk size from 5000 to 10000. **Stratified Sampling**. Put the data into class buckets: One for normal connections and one for intrusions. Generate 50 chunks as follows: 1) choose an initial $P(y)$, 2) sample without replacement from each bucket to form a chunk that satisfies $P(y)$, 3) evolve $P(y)$ and sample from the remaining data in the buckets as the next chunk, and finally, 4) put data sampled in step 2 back to the buckets and repeat steps 2 and 3. The chunk size is also varied from 5000 to 10000. **Merging**. Partition the data into chunks and maintain its original order. Both normal connections and intrusions should account for at least 5% in each chunk, if this is not satisfied, merge the chunk with the next chunk until the percentage is above the threshold. In the experiments, we discard one large data chunk with only normal examples and there are altogether 22 chunks with chunk size varying from 1069 to 32122. We construct these three data sets because the original data set does not have explicit time stamps and there are chunks of intrusions followed by chunks of normal examples. Each chunk may only contain examples of one class and this is a non-learnable problem. On the other hand, the three data streams we constructed have the following properties: 1) each chunk is ensured to have examples from both classes; and 2) the distributions of two consecutive data chunks are different and evolve in different ways in three streams. Upon such datasets, we could validate our claims of stream mining upon the relaxed assumption.

Measures and Baseline Methods For a data stream with chunks $T_1, T_2, \ldots, T_N$, we use T_i as the training set to classify T_{i+1} and the distribution of the test set T_{i+1} is not necessarily the same as that of T_i. We evaluate the accuracy of each model. For the classifier having posterior probability as the output, the predicted class label is the class with the highest posterior probability under zero-one loss function. Another measure is mean squared error (MSE), defined as the averaged difference between estimated probability and true posterior probability $P(y|\mathbf{x})$. In problems where we are only exposed to the class labels but do not know the true probability, we set $P(y|\mathbf{x}) = 1$ if y is $\mathbf{x}$'s true class label, otherwise $P(y|\mathbf{x}) = 0$. We are comparing the following algorithms: single models built using Decision Tree (DT), SVM, Logistic Regression (LR) and ensemble approaches including Weighted Ensemble (WE), Simple Voting (SV) and Averaging Probability (AP). Different from averaging ensemble framework, the weighted ensemble approach assigns a weight to each base model which reflects its predictive accuracy on the training data (obtained by tenfold cross validation) and the final prediction outputs are combined through weighted averaging. In previous work,

Table 1. Performance Comparison on Synthetic Stream Data

Binary Stream Data

Measure	Chunk Size 100						Chunk Size 2000					
	DT	SVM	LR	WE	SV	AP	DT	SVM	LR	WE	SV	AP
Aacc	0.7243	0.7591	0.7346	0.7461	0.7595	**0.7690**	**0.8424**	0.8318	0.8366	0.8339	0.8370	0.8369
Amse	0.2731	0.2387	0.2625	0.1889	0.2379	**0.1752**	0.1540	0.1649	0.1601	0.1262	0.1597	**0.1242**
AR	2.2323	1.6465	2.1111	1.8889	1.5152	**1.4848**	2.1313	1.8485	1.6869	1.7980	**1.5455**	**1.5455**
SR	0.8902	0.6898	0.8193	0.7544	**0.5414**	**0.5414**	0.9757	0.8732	0.8765	0.8687	**0.7460**	**0.7460**
#W	30	47	28	34	50	**53**	41	46	58	49	**60**	**60**
#L	53	12	39	23	**2**	**2**	54	31	27	29	**15**	**15**

Multi-Class Stream Data

Measure	Chunk Size 100						Chunk Size 2000					
	DT	SVM	LR	WE	SV	AP	DT	SVM	LR	WE	SV	AP
Aacc	0.5111	0.5295	0.5298	0.5301	**0.5320**	0.5314	0.4991	0.4939	0.4920	0.5130	0.4950	**0.5139**
Amse	0.1745	0.1413	0.1272	0.1210	0.1872	**0.1208**	0.1764	0.1461	0.1322	0.1246	0.2020	**0.1244**
AR	2.3636	1.9293	1.9798	1.8283	1.8788	**1.7273**	2.0202	2.2626	2.2424	1.6667	2.1111	**1.4040**
SR	0.8263	0.7986	0.7822	**0.5159**	0.6589	0.6197	0.8919	0.8758	0.9045	**0.4949**	0.9023	0.5330
#W	22	35	31	23	28	**36**	38	28	31	34	35	**61**
#L	58	28	29	**6**	16	9	40	54	55	**1**	46	2

such weighted ensembles are shown to be effective when the "shared distribution" assumption holds true. In our experiments, we evaluate its performances upon the relaxed assumption. For all the base learning algorithms, we use the implementation in Weka package [11] with parameters set to be the default values. In the averaging ensemble framework, either SV or AP, the base streaming models could be chosen arbitrarily. We test the framework where base models are constructed from either different learning algorithms or different samples of the training sets.

For a learning algorithm A_h, we build a model based on T_i and evaluate it on T_{i+1} to obtain its accuracy p_{ih} and MSE e_{ih}. There are altogether $N-1$ models and we report its average accuracy (Aacc) and average MSE (Amse). Furthermore, in each of the $N-1$ runs, we compare the performances of all algorithms and decide the winner and loser in the following way: if p_{ih} is within $m\%$ of $\max_h p_{ih}$, algorithm A_h is a winner in that run, similarly, if p_{ih} is within $m\%$ of $\min_h p_{ih}$, it is a loser. In other words, we tolerate some small difference between two algorithms, if their accuracies are the same with respect to the "margin tolerance rate" m, we regard their performances as the same. We report the number of wins and loses for each algorithm (#W and #L). With winners ranking the first, losers ranking the third and all other algorithms occupying the second position, we give $N-1$ ranks to each algorithm and obtain the mean and standard deviation of the ranks (AR and SR). A good algorithm will have a higher accuracy, a lower MSE and average rank closer to 1. If it has a lower standard deviation in the ranks, the learning algorithm is more stable.

3.2. Empirical Results

We report the experimental results comparing the two ensemble approaches (SV, AP) with single model algorithms (DT, SVM, LR) as well as weighted ensemble method (WE). As discussed below in detail, the results clearly demonstrate that on the stream data where training and testing distributions are different and fast evolving, the two ensemble approaches have the best performances on average with higher accuracy and lower variations. Therefore, when facing unknown future, the ensemble framework is the best choice to minimize the number of bad predictions.

Test on Concept-Drifting Stream Data We generate four synthetic data streams, each of which is either binary or multi-class and has chunk size 100 or 2000. Each data set has 10 dimensions and 100 data chunks. The margin tolerance rate is set to be 0.01. From Table 1, it is clear that the two ensemble approaches (SV and AP) have better performances (best are highlighted in bold font) regardless of the measures we are using, the problem type (binary or multi-class) and the chunk size. Take the binary problem with chunk size 100 as an example. AP proves to be the most accurate and stable classifier with the highest accuracy (0.7690), lowest MSE (0.1752), 53 wins and only 2 loses. SV is quite comparable to AP with 50 wins and 2 loses. The best single classifier SVM wins 47 times and loses 12 times and WE approach seems to suffer from its training set-based weights with only 34 wins but 23 loses. These results suggest the following: when the "same distribution" between training and testing data does not exist: 1) there are no uniformly best single classifiers, even decision tree, which has the worst average performance, still wins 30 times among all 99 competitions. The large variabilities of single models result in their high expected errors; 2) on average, ensemble approaches, simple voting or probability averaging, are the most capable of predicting on future data with unknown distributions; 3) assigning a weight to each

Table 2. Data Steam with Shared Distribution

	DT	SVM	LR	WE	SV	AP
Aacc	0.7366	0.7756	0.7604	0.7704	0.7772	**0.7797**
Amse	0.2369	0.1980	0.2052	0.1427	0.1939	**0.1254**
AR	2.3030	1.7071	2.1010	1.7879	1.6768	**1.6364**
SR	0.8506	0.6739	0.8143	0.7460	**0.5312**	0.5431
#W	25	**41**	28	40	35	39
#L	55	12	38	19	**3**	**3**

base learner even hurts the predictive performances on testing data since the distribution it tries to match is different from the true one.

For binary streams, we also record the results on the first 40 chunks to see how the concept evolution affects the classification performances. The results indicate that even within the same data stream, the best single classifier for the first 40 chunks is different from the best one on the whole data set. Take the stream data with chunk size 100 as an example. At first, LR has 18 wins, compared with DT (4 wins) and SVM (14 wins), it appears to be the best on average. However, later, SVM takes the first place with 47 wins (DT 30 and LR 28). This clearly indicates that in a stream whose distribution evolves, a model which performs well on current data may have poor performances on future data. Since we never know when and how the distribution changes, depending on one single classifier is rather risky. On the other hand, ensemble based on averaged probability is more robust and accurate, which is the winner for classifying data streams with regard to the average performance (ranks around 1.5 while others rank more than 2 on average). Ensemble based on simple voting (SV) produces results similar to that of AP in binary stream problems, but is not that competitive in multi-class problems. The reason may be that two class problems are easier for prediction tasks, so the probability outputs of a classifier may be rather skewed, greater than 0.9 or less than 0.1. So there isn't much difference between simple voting and averaging probability in this case. However, when the number of classes grows large, it is quite unlikely that the predicted probability is skewed. The strengths of probability averaging over simple voting is therefore demonstrated on multi-class problems. As for the weighted ensemble approach, it sometimes increases the predictive accuracy, but sometimes gives even worse predictions compared with single models. Whether it performs good or not is dependent on how the training and testing distributions match. In this sense, the other two simple ensemble methods are more robust since they are not based on the assumption that training and testing data come from the same distribution.

We also compare all the methods on data streams where training and testing distributions are identical, as assumed by previous stream mining algorithms. Two data streams are generated, where the first one is used as the training set and the second one is the testing data. They have "evolving shared distribution" in the sense that the corresponding chunks of training and testing streams are sampled from the same distribution, but this shared distribution is evolving in the way we have described in Section 3.1. Each stream has 100 data chunks with 100 examples in each chunk. Since the training examples may be far from sufficient due to the small training size, it may not obtain an accurate model even if the training and testing distributions are the same. As indicated in Table 2, again, ensemble could help reduce the classification errors on such data sets (from around 0.2 to 0.12). AP has obtained the highest accuracy, the lowest mean squared error and the highest rank on average. Weighted ensemble wins 40 times, which appears to be competitive, but worse than AP in terms of loses (19 versus 2). Therefore, even if the "shared distribution" assumption holds true, simple averaging or simple voting are still more effective than weighted ensemble because the training samples may be insufficient and training errors derived from such data sets are unreliable. We could safely conclude that the averaging ensemble framework could maximize the chance of matching the true distribution, thus return the most satisfactory prediction results in general.

Table 3. Ensemble on Real Data

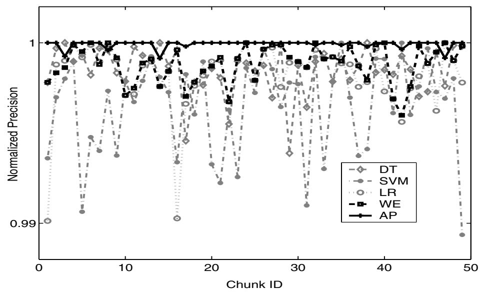
Shuffling

	DT	SVM	LR	WE	SV	AP
Aacc	0.9961	0.9941	0.9957	0.9964	**0.9975**	**0.9975**
Amse	0.0039	0.0059	0.0043	0.0028	0.0025	**0.0024**
AR	1.9592	2.5306	1.9388	1.6939	**1.0000**	**1.0000**
SR	0.8406	0.7665	0.8013	0.7959	**0**	**0**
#W	18	8	17	25	**49**	**49**
#L	16	34	14	10	**0**	**0**

Stratified Sampling

	DT	SVM	LR	WE	SV	AP
Aacc	0.9720	0.9744	0.9699	0.9707	**0.9755**	**0.9755**
Amse	0.0280	0.0256	0.0301	0.0259	0.0245	**0.0232**
AR	1.6531	1.5510	1.6122	1.5306	**1.2245**	**1.2245**
SR	0.9026	0.7654	0.8854	0.8191	**0.4684**	**0.4684**
#W	31	30	32	33	**39**	**39**
#L	14	8	13	10	**1**	**1**

Test on KDD Cup'99 Data In Section 3.1, we describe the three data streams we generate from the KDD Cup'99 intrusion detection data set and how the training and testing distributions are made different explicitly. Also, as illustrated in Section 1, both $P(\mathbf{x})$ and $P(y|\mathbf{x})$ undergo continuous and significant changes in this stream data. Results of various methods on streams generated by "Shuffling" and "Stratified Sampling" are summarized in Table 3 where margin tolerance rate is set to be 0.001. Similar to earlier results on simulated streams, the advantage of the ensemble framework is clearly demonstrated. The two ensemble



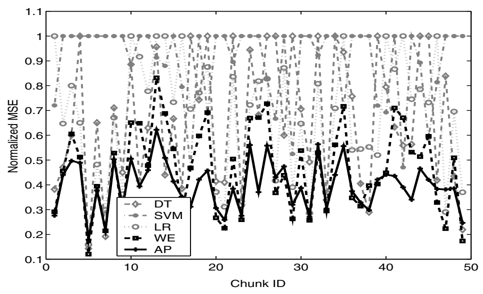


Figure 5. Accuracy and MSE on Real Data

approaches not only increase the accuracy of single models but also occupy the first place in most of the evaluations. The most significant improvements could be observed on the data set generated by shuffling, where accuracy goes up from 0.9961 to 0.9975 and the number of wins increases from 18 to 49 after combining outputs of multiple models. The performances of SV and AP are almost the same for these two data sets. As discussed in the synthetic data experiments, SV and AP are expected to have similar predictions when the estimated probabilities of each class are skewed in binary problems. Another observation is that the weighted ensemble approach could improve over a single model but the improvements are less significant compared with simple averaging. This phenomenon again shows that the weighting scheme cannot survive the relaxed assumption where training and testing distributions could be different since it fits the training data too "tightly".

Figure 5 reveals some detailed information about the evaluation results (Accuracy and MSE w.r.t Chunk ID) on the first data set where data records are randomly shuffled. To exclude the effects of different scales, we normalize the measures by the maximal value. It is obvious that the probability averaging ensemble (AP) is the most accurate classifier in general with normalized accuracy close to 1 and mean squared error below 0.5. Also, as shown in both plots, as measures of single models fluctuate within a wide range, the performances of probability averaging ensemble are much more stable. This clearly shows the benefits of using our ensemble framework when the testing distribution is unknown and departed from the training distribution. On average, the ensemble would approximate the true distribution more accurately than single models, with least number of loses. The weighted ensemble could achieve higher accuracy than single-model classifier but still has larger variance and worse average performance compared with AP. For example, the highest normalized MSE of AP is only around 0.6, but over 0.8 for weighted ensemble approach.

Table 4. Ensemble on Data Set with Merging

	DT	SVM	LR	WE	SV	AP
Aacc	**0.8508**	0.8193	0.7869	0.8182	0.8218	0.8218
Amse	**0.1492**	0.1807	0.2131	0.1801	0.1782	0.1559
AR	1.95	**1.65**	2.55	**1.65**	**1.65**	**1.65**
SR	0.8256	0.7452	0.6048	**0.5871**	0.6708	0.6708
#W	7	**10**	1	8	9	9
#L	6	3	12	**1**	2	2

Table 5. Ensemble of Multiple Samples

	DT	SVM	WE	AP	APS
Aacc	0.8508	0.8193	0.8182	0.8218	**0.8579**
Amse	0.1492	0.1807	0.1801	0.1559	**0.1301**
AR	2.25	2.05	1.9	1.85	**1.65**
SR	0.9665	0.8256	**0.7182**	0.7452	0.7452
#W	7	6	6	7	**10**
#L	12	7	4	4	**3**

In Table 4, the results on the third data stream where the original order is maintained are reported. It seems that logistic regression is consistently worse than the other two base learners, which wins only once, whereas DT and SVM win 7 and 10 times respectively. In general, the ensemble approaches (SV and AP) still predict well, win 9 times and lose twice. Unlike the synthetic stream where distribution evolves quickly and no uniformly best classifier could be derived, this data set may have certain characteristics that favor one classifier over another. This property could be observed and analyzed from the data set and regarded as prior knowledge. We could incorporate such knowledge into each base learner and further improve a single model. In this case, if we know that logistic regression is not suitable for this data and expected to perform bad, we could construct an ensemble based on multiple samples from the training data using decision tree and SVM as the base learners. In this experiment, we build 10 base models from 10 samples of the original data and combine their outputs by averaging the probabilities. As shown in Table 5, the ensemble based on multiple samples (denoted as APS), boosts the accuracy from 0.8508 (accuracy of the best single classifier) to 0.8579 and reduces the MSE from 0.1492 down to 0.1301. When counting the number of wins and loses, APS is the best compared with single models and other ensemble approaches. It should be noted that we are not against the ensemble based on multiple learning algorithms. Logistic regression is not working on this data set but may perform good on others, when we know nothing about the data set, ensemble based on multiple learning algorithms is the safest way for prediction. The key point here is that ensemble demonstrates its strengths no matter we know nothing or something about the stream data. Regardless of the baseline models, the averaging ensemble framework is expected to generate the best estimate of the true target function from the limited information conveyed by training examples.

4. Related Work

Sample selection bias [13] investigates the effect on learning accuracy when the training data is a "biased" sample of the true distribution. Although the true target function to be modeled, $P(y|\mathbf{x})$, does not "explicitly" change, its value can be wrong in various ways in the biased training data. Previously, decision tree based model averaging has been shown to be helpful to correct feature bias or the bias where the chance to sample an example into the training set is independent on y given $\mathbf{x}$ [13]. The most important difference of our work from these previous studies is: (1) $P(y|\mathbf{x})$ in our problem is allowed to explicitly change and can change significantly, (2) changes in $P(y|\mathbf{x})$ are combined with changes in $P(\mathbf{x})$. To consider the significance of our work under sample selection bias formulation, our comprehensive results significantly extend the previous work and demonstrate that model averaging can reliably correct sample selection bias where biased conditional probability is quite different from unbiased testing data.

5. Conclusions

We demonstrate that assuming training and testing data follow the same distribution, as commonly held by much existing work, is inappropriate for practical streaming systems. On the contrary, the distributions on both feature vector and class label given feature vector can evolve in some unknown manner, and models matching training distribution well may perform poorly in continuously changing distributions. As a result, the difference between training and testing distributions needs to be taken into account. We also argue that, contrary to common practice, in order to design robust and effective stream mining algorithms against changes, an appropriate methodology is not to overly match the training distribution, such as by weighted voting or weighed averaging where the weights are assigned according to training distribution. On these basis, we use both model averaging of conditional probability estimators and simple voting of class labels as a robust framework "against change" and argue that weighted averaging/voting are inappropriate. We demonstrate both formally and empirically such a framework can reduce expected errors and give the best performance on average when the test data does not follow the same distribution as the training data. Among many experiments, in a test on KDDCup'99 intrusion detection dataset, the framework's predictions are the most accurate in 49 out of 49 competitions, whereas the best baseline model is the most accurate only 18 times. The baseline models are not limited to those used in the empirical evaluations and other more sophisticated methods can be plugged in. Since the property of expected error reduction is proved formally, the framework is expected to have robust and better performance regardless of chosen baseline models. Nonetheless, for stream mining research in general, practitioners and researchers ought to clearly specify the training and testing assumptions made in the algorithms, and evaluation benchmarks ought to be designed with these assumptions clearly considered.

References

[1] C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On demand classification of data streams. In *Proc. KDD'04*, pages 503–508.

[2] W. Fan. Systematic data selection to mine concept-drifting data streams. In *Proc. KDD'04*, pages 128–137.

[3] J. Gao, W. Fan, J. Han, and P. Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *Proc. SDM'07*.

[4] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. KDD'01*, pages 97–106.

[5] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. VLDB'04*, pages 180–191.

[6] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proc. ICML'00*, pages 487–494.

[7] J. Kolter and M. Maloof. Using additive expert ensembles to cope with concept drift. In *Proc. ICML'05*, pages 449–456.

[8] M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In *Proc. ECML/PKDD'05 Workshop on Knowledge Discovery in Data Streams*.

[9] H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. KDD'03*, pages 226–235.

[10] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

[11] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[12] Y. Yang, X. Wu, and X. Zhu. Combining proactive and reactive predictions for data streams. In *Proc. KDD'05*, pages 710–715.

[13] B. Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proc. ICML'04*.