Active Learning From Stream Data Using Optimal Weight Classifier Ensemble

Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi

***Abstract*—In this paper, we propose a new research problem on active learning from data streams, where data volumes grow continuously, and labeling all data is considered expensive and impractical. The objective is to label a small portion of stream data from which a model is derived to predict future instances as accurately as possible. To tackle the technical challenges raised by the dynamic nature of the stream data, i.e., increasing data volumes and evolving decision concepts, we propose a classifier-ensemble-based active learning framework that selectively labels instances from data streams to build a classifier ensemble. We argue that a classifier ensemble's variance directly corresponds to its error rate, and reducing a classifier ensemble's variance is equivalent to improving its prediction accuracy. Because of this, one should label instances toward the minimization of the variance of the underlying classifier ensemble. Accordingly, we introduce a *minimum-variance* (MV) principle to guide the instance labeling process for data streams. In addition, we derive an optimal-weight calculation method to determine the weight values for the classifier ensemble. The MV principle and the optimal weighting module are combined to build an active learning framework for data streams. Experimental results on synthetic and real-world data demonstrate the performance of the proposed work in comparison with other approaches.**

***Index Terms*—Active learning, classifier ensemble, stream data.**

I. Introduction

RECENT developments in storage technology and networking architectures have made it possible for broad areas of applications to rely on stream data for quick response and rapid decision making [1]. One of the recent challenges facing data mining is to digest massive volumes of data collected from data stream environments [1]–[10].

Manuscript received July 23, 2009; revised November 12, 2009; accepted January 7, 2010. Date of publication April 1, 2010; date of current version November 17, 2010. This work was supported in part by the Australia Discovery Grant DP1093762, by the National Science Foundation of China (NSFC) under Grant 60674109, Grant 70621001, and Grant 70531040, and by the 973 Project of the Chinese Ministry of Science and Technology under Grant 2004CB720103. This paper was presented in part at the Proceedings of the 7th IEEE International Conference on Data Mining (ICDM), Omaha, NE, October 2007. This paper was recommended by Associate Editor N. Chawla.

X. Zhu is with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA, and also with the QCIS Center, Faculty of Engineering and Information Technology, University of Technology, Sydney, NSW 2007, Australia (e-mail: xqzhu@cse.fau.edu).

P. Zhang is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100090, China (e-mail: zhangpeng@ict.ac.cn).

X. Lin is with the Department of Management Science and Information Systems, Rutgers Business School, Rutgers, the State University of New Jersey, Newark, NJ 07102 USA (e-mail: lin@business.rutgers.edu).

Y. Shi is with the College of Information Science and Technology, University of Nebraska at Omaha, NE 68118 USA, and also with the Fictitious Economy and Data Science Research Center, Chinese Academy of Sciences, Beijing 100080, China (e-mail: yshi@guca.ac.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2010.2042445

In the domain of classification, providing a set of labeled training examples is essential for generating predictive models. It is well accepted that labeling training examples is a costly procedure [11], which requires comprehensive and intensive investigations on the instances, and incorrectly labeled examples will significantly deteriorate the performance of the model built from the data [12], [23]. A common practice to address the problem is to use active learning techniques to selectively label a number of instances from which an accurate predictive model can be formed [13]–[17], [39]–[44], [54]–[57]. An active learner generally begins with a very small number of randomly labeled examples, carefully selects a few additional examples for which it requests labels, learns from the results of that request, and then by using its newly gained knowledge carefully chooses which examples to label next. The goal of active learning is to maximize the prediction accuracy by labeling only a very limited number of instances, and the main challenge is to identify "important" instances that should be labeled to improve the model training under the fact that one could not afford to label all samples [57]. A general practice for active learning is to employ some heuristics (or rules) in determining the most needed instances. For example, uncertainty sampling [43], query by committee (QBC) [13], [14], or query by margin [41], [43] principles take instances with which the current learners have the highest uncertainty as the mostly needed instances for labeling. The intuition is to label instances on which the current learner(s) has the highest uncertainty, so providing labels to those instances can help improve the model training. A large body of work exists for active learning on static data sets [13]–[17], [39]–[44], all of which aim at building one single optimal model from the labeled data. None of them, however, fits in data stream environments, where the continuous data volumes and the drifting of the concepts raise significant challenges.

A. Concept Drifting in Stream Data

Given a binary classification problem with classes denoted by c_1 and c_2, respectively, the optimal decision [58] in labeling a previously unseen example x is to find the class label c_i, $i \in \{1, 2\}$, that maximizes the joint probability

$$\arg\max_{i\in\{1,2\}} P(c_i, x). \tag{1}$$

Using the probability product rule $P(c_i, x) = P(c_i|x)P(x) = P(x|c_i)P(c_i)$, (1) can be rearranged for the purpose of maximizing the posterior probability as defined by (2), where $P(c_i)$ defines the priori probability (or density) of the class c_i, and

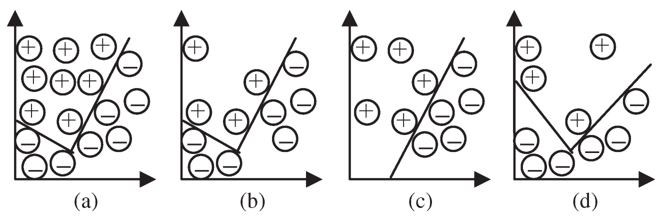

Fig. 1. Conceptual view of concept drifting in data stream. (a) The genuine decision boundary assume all samples were observed. (b) Priori probability drifting scenario (triggered by changes in $P(c_i)$ only). (c) Conditional probability drifting scenario [triggered by changes in $P(x|c_i)$ only, where class priori probability $P(c_i)$ remains the same as (a)]. (d) Conjunct probability drifting [triggered by changes in $P(c_i)$ and $P(x|c_i)$].

$P(x|c_i)$ denotes the class conditional probability (or likelihood) of the sample x given class c_i, i.e.,

$$\arg\max_{i\in\{1,2\}} P(c_i|x) = \arg\max_{i\in\{1,2\}} \frac{P(x|c_i)P(c_i)}{P(x)}. \tag{2}$$

The challenge of the data stream, in comparison with a static data set, lies on the fact that one can only observe a portion of the stream data, so both $P(x|c_i)$ and $P(c_i)$ may constantly change/drift across the stream. As a result, the posterior probability of a class c_i, given a sample x, also constantly changes, which may result in different predictions for two identical instances, depending on the actual time they appear in the stream. Formally, the concept drifting in the data stream refers to the variation of the priori probability $P(c_i)$ and the class conditional probability $P(x|c_i)$ across the stream data. The drifting of the concept can further be decomposed into the following three categories: 1) priori probability drifting: the concept drifting is mainly triggered by the class priori probability $P(c_i)$; 2) conditional probability drifting: the concept drifting is mainly trigged by the class conditional probability; and 3) conjunct probability drifting: both $P(c_i)$ and $P(x|c_i)$ constantly change across the data stream. A conceptual view of the above categorizations is illustrated in Fig. 1.

B. Active Learning for Data Streams

For data streams with continuous volumes, the needs of active learning are compelling, simply because manually investigating and labeling all instances are out of the question for many applications. The objective of employing active learning for data streams is to label "important" samples, based on the data observed so far, such that the prediction accuracy on future examples can be maximized.

For static data sets whose whole candidate pools can be observed and their genuine decision boundaries are invariant, active learning is supposed to find "which samples should be labeled?" For data streams with continuously increasing volumes and varying decision boundaries, active learning needs to answer "when and which examples should be selected for labeling?" Intuitively, if there is no concept drifting involved in the incoming data, then it makes sense to save the labeling cost for future samples that may have different distributions from the current data [9]. Unfortunately, implementing such a *when-and-which* labeling paradigm is difficult for data streams; this is mainly because concept drifting in data streams is mostly triggered by complicated factors (as discussed in the previous section), so we may not be able to accurately capture the best time for labeling. In addition, without seeing the future samples, it is difficult to determine the efforts (costs) one should spend on the current samples such that the overall prediction accuracy can be maximized. To simplify the problem and deliver an applicable active learning framework for data streams, we assume in this paper that the concepts in the data are constantly evolving, so active learning is carried out on a regular basis, and the objective is to find important samples out of a certain number of newly arrived instances for labeling. As we will shortly demonstrate in Section VI, such an approach will produce better results than relying on the detection of concept drifting for active learning.

C. Challenges of Active Learning From Stream Data

Presumably, the challenge of active learning from stream data is threefold [46]: 1) in data stream environments, the candidate pool is dynamically changing, whereas existing active learning algorithms are mainly designed for static data sets only; 2) the concepts, such as the decision logics and class distributions, of the data streams are continuously evolving [2]–[10], [46], whereas existing active learning algorithms only deal with static concepts; and 3) because of the increasing data volumes, building one single model from all the labeled data is computationally expensive for data streams, even if memory is not an issue, whereas most existing active learning algorithms rely on a model built from the whole collection of data for instance labeling [13]–[17], [39]–[44]. In data stream environments, it is impractical to build one single model from all previously labeled examples. On the other hand, as the concepts of the data streams evolve, aggregating all labeled instances may reduce the learner performance instead of improving it [3]. Therefore, we will have to rely on a set of classifiers, instead of one, to fulfill the objective of active learning from data streams. More specifically, a solution to the problem must explicitly address the following three concerns:

1) What is the objective of active learning for stream data? Or what is the final goal of carrying out active learning for stream data?
2) What are the objective function and criteria of instance labeling for stream data?
3) How to tackle a data stream with drifting concepts and massive data volumes for effective active learning?

We present in this paper our recent research efforts in resolving the above concerns. In short, we propose a weighted classifier ensemble [36] framework to address the challenges raised from data streams. Our objective is to maximize the prediction accuracy of the classifier ensemble built from the labeled stream data (first concern). For this purpose, we argue that the objective function of the instance selection is to minimize the variance of the classifier ensemble built from the data (second concern). The employment of the classifier ensemble ensures that our method cannot only handle data with massive volumes but is also able to adapt to the drifting concepts though the adjustment of the weight values of the ensemble members (the third concern).

The remainder of this paper is structured as follows: Section II briefly reviews the related work. Section III presents a motivating example and simple solutions. Section IV introduces

TABLE I
KEY SYMBOLS USED IN THIS PAPER

Symbol	***Description***
c_i	The label for the i^{th} class
l	The total number of classes in the data
S_n	The n^{th} data chunk of the data stream
L_n, U_n	The subsets of labeled and unlabeled instances in S_n
C_n	The classifier built from the labeled subset, L_n, of S_n
$\eta^n_{c_i}$	A random variable accounts for the variance of the classifier C_n *w.r.t.* class c_i
$\sigma^2_{\eta^n_{c_i}}$	The variance of $\eta^n_{c_i}$
β_{c_i}	The bias of the classifier with respect to class c_i
E	An abbreviation of a classifier ensemble which consists of a number of classifiers C_1, C_2, …
k	The number of classifiers forms a classifier ensemble, where each component classifier is called a base classifier
w^n	The weight value of the classifier C_n of the classifier ensemble
$\eta^E_{c_i}$	A random variable accounts for the variance of the classifier ensemble E *w.r.t.* class c_i,
$\sigma^2_{\eta^E_{c_i}}$	The variance of $\eta^E_{c_i}$

classifier variance and an optimal weight calculation method to minimize classifier ensemble error rate. Following the conclusions derived from Section IV, Section V describes the active learning algorithm in detail. Experimental results are reported in Sections VI, and we conclude in Section VII. For ease of presentation, the key symbols used in this paper are listed in Table I.

II. Related Work

In addition to active learning, our research is closely related to the existing work on classifier ensemble and active mining (AM).

Classifier ensemble is an established research area. Numerous methods [34]–[37], [52], [53] exist for improving the ensemble learning accuracies. Examples include bagging [52], boosting [53], weighted voting [35], or diversity customization for ensemble construction [48]. Combining classifier ensemble and active learning has been reported in much research [44], [48]–[50], [57]. Traditional query-by-bagging-based active learning approaches [44], [48], for example, employ bootstrap sampling to train a number of classifiers to estimate the uncertainty of each unlabeled example. Other methods [49], [50] employ the active learning principle to improve ensemble learning, such as multiclass boosting classification [49] or active ensemble learning [50]. Although all these methods have been using the ensemble framework to benefit active learning or vice versa, they are not primarily designed for data stream environments.

For data streams with continuous volumes, the classifier ensemble has shown to be effective for tackling data volume and concept drifting challenges [3], [4], [7], [8]. Street and Kim [7] proposed a streaming ensemble algorithm that combines decision tree models using majority voting. Kolter and Maloof [18] proposed an AddExp ensemble method by using weighted online learners to handle drifting concepts. In [3], Wang *et al.* proposed a weighted ensemble framework for concept drifting data streams and proved that the error rate of a classifier ensemble is less than a single classifier trained from the aggregated data of all consecutive k chunks. In [8], Gao *et al.* proposed to employ sampling and ensemble techniques for data streams with skewed distributions. In some recent works [4], [10], [59], we have employed the classifier ensemble for stream data cleansing [10] and for combining labeled and unlabeled samples for multiclass [4] and uniclass [59] stream data mining [4]. In summary, although the ensemble has been popularly used for stream data mining, no theoretical analysis is currently available for calculating the optimal weight values for ensemble learning.

Realizing that labeling all stream data is expensive and heavily time consuming, Fan *et al.* proposed an AM framework [9] that labels samples only if it is necessary. In short, AM uses a decision tree (trained from the currently labeled data) to compare the distributions of the incoming samples and the data collected at hand using the tree branches (without observing the class labels). If two sets of samples are subject to different distributions, then a labeling process is triggered to randomly select a few incoming samples for labeling. Although our research shares the same goal as AM, i.e., minimizing the labeling cost for data streams, the differences between them are, however, fundamental: 1) AM only answers "when to select data for labeling," and once it decides to label the incoming data, it randomly selects samples for labeling. In comparison, our work explicitly answers "which samples should be labeled." 2) AM does not allow users to allocate labeling costs because labeling is triggered only if the distributional changes emerge. In comparison, our work allows users to flexibly control the labeling cost at any point of the stream. 3) AM is only applicable for decision trees because the detection of the distributional changes relies on comparisons on the tree branches. In comparison, our work can be applied to any learners including decision trees. Our experimental comparisons in Section VI will demonstrate that AM is inferior to or marginally better than random-sampling-based approaches in solving our problem.

III. Problem Definition and Simple Solutions

A. Motivating Example and Problem Definition

Consider an online intrusion detection center that monitors the incoming traffic flow of some network servers to identify suspicious users based on their actions and IP package payload. Here, the daily (or hourly) traffic flow constitutes a data stream. Assume that the number of network connections arrives at an average rate of 10 000 connections per hour, out of which the network security experts can only investigate 5%. Accordingly, the problem becomes which 5% of the connections should be inspected to improve the existing intrusion detection model and identify future intrusions as accurately as possible (in this paper, we assume that all network connections are subject to the same cost, so the objective is to minimize the misclassification error instead of minimizing the total loss [45]). To handle massive volumes of stream data, a common solution is to partition the data into chunks, as shown in Fig. 2. We can label 5% of data in each chunk and build one classifier from the labeled data. As a result, a set of base classifiers are trained and used to form a classifier ensemble [34]–[36] to identify intrusions from newly arrived data. The framework in Fig. 2 can be applied to a variety of stream applications as long as instance labeling is of concern. The employment of a classifier ensemble ensures

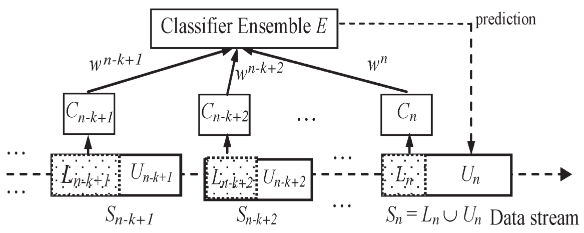

Fig. 2. general framework for active learning from stream data.

that our framework can effectively handle massive volumes of stream data without relying on any complex incremental learning procedures [2], [19].

Based on the framework in Fig. 2, we assume that stream data are partitioned into chunks according to the user-specified chunk size. We also assume that once the algorithm moves to chunk S_n, all instances in previous chunks, $\ldots, S_{n-3}$, S_{n-2}, S_{n-1}, are inaccessible except classifiers built from them (i.e., $\ldots, C_{n-3}, C_{n-2}, C_{n-1}$). Based on the above assumptions, the objective becomes labeling instances in data chunk S_n such that a classifier C_n built from the labeled instances in S_n, along with the most recent $k-1$ classifiers $C_{n-k+1}, \ldots, C_{n-1}$, can form a classifier ensemble with maximum prediction accuracy on unlabeled instances in S_n.

B. Simple Solutions

1) (RS): Arguably, random sampling (RS), where instances in S_n are randomly sampled and labeled, is the simplest approach to solve our problem. Although simple, it turns out that RS works surprisingly well in practice (as we will shortly see in Section VI). The niche of RS stems from the fact that in data stream the class distributions may vary significantly across data chunks. While general active learning algorithms seek to label "important" instances, they may significantly change class distributions by favoring one class of instances. Accordingly, the labeled instances no longer reveal genuine class distributions in the data chunk. This problem is less severe for a static data set where the candidate pool is fixed and active learning algorithms are able to survey all instances. RS avoids this problem by randomly selecting samples for labeling. As a result, it can produce a training set with the most similar class distributions to the current data chunk, although the instances it labeled might not be as "informative" as those carefully selected.

2) LU Sampling: Another way of solving the problem is to disregard the dynamic nature of data streams and treat each data chunk S_n as a static data set. One can then apply existing active learning algorithms to the chunk S_n without considering any other data chunks. Because instance labeling is carried out independently in each data chunk, the weakness of local uncertainty (LU) is obvious: although each data chunk might be able to label the most important instances from their own perspectives, the locally labeled instances have very limited value for training the global classifier ensemble.

3) GU Sampling: Global uncertainty (GU) sampling-based active learning will use historical classifiers, along with the one from S_n, to form a committee for instance labeling. Upon the receiving of a data chunk S_n, GU randomly labels a tiny set of instances from S_n and builds a classifier C_n. This classifier, along with $k-1$ historical classifiers, form a committee to assess instances in S_n and label the ones with the largest uncertainty. The whole process repeats until a certain number of instances in S_n are labeled. At any stage, the user may choose to rebuild C_n by using labeled examples in S_n to update the base classifiers for labeling.

GU essentially labels instances on which the classifier committee has the highest uncertainty. This appears to be a promising design as the same concept has been validated by QBC-based approaches [13], [14]. Unfortunately, our experimental results in Section VI indicate that GU's performance is still unsatisfactory and often inferior to RS. One possible reason is that different from the traditional QBC, where committee members are learned from samples drawn from the same distributions, the committee classifiers in data streams are learned from different data chunks. Because of this, the classifiers may vary significantly in classifying each single instance, and the average uncertainty over all committee classifiers (like QBC does) may not reveal an ensemble's genuine uncertainty on the instance.

4) AM-Based Sampling: AM sampling is motivated by the AM framework proposed by Fan *et al.* [9], and the intuition is to select incoming samples, which have the most significant distributional changes from the currently observed data, for labeling. More specifically, for each data chunk S_k, a decision tree dt_k is trained from the labeled samples in S_k and is used to calculate the distribution of all samples in S_k with respect to the tree dt_k (please refer to [9] for technical details). Assume that in the arrival of a data chunk S_n for labeling, AM randomly labels a tiny set of instances from S_n and builds a decision tree dt_n. After that, AM calculates the distributions of unlabeled samples in S_n with respect to each of the k trees $(dt_{n-k+1}, \ldots, dt_{n-1}, dt_n)$. The distributions are used to compare with the retained distribution of each tree, and the differences are used to assign a weight value to each unlabeled instance in S_n such that samples with the most significant distributional changes are selected for labeling. The above process repeats until a certain number of instances in S_n are labeled.

IV. Classifier Ensemble Variance Reduction for Error Minimization

In this section, we first study the classifier variance for a single learner and then extend our analysis to classifier ensemble. We argue that minimizing the classifier ensemble variance is equivalent to minimizing its error rate. Following this conclusion, we derive an optimal-weight calculation method to assign weight values to the classifiers such that they can form an ensemble with minimum error rate. The minimization of the classifier ensemble error rate through variance reduction acts as a principle to actively select mostly needed instances for labeling.

A. Bias & Variance Decomposition for a Single Classifier

A Bayes optimal decision rule [58] assigns input x to a class c_i if the *a posterior probability* $p(c_i|x)$ is the largest among a set of classes c_i, $i \in \{1, \ldots, l\}$. Although we expect that a classifier's probability estimation in classifying x is equal to $p(c_i|x)$, the actual probability $f_{c_i}(x)$, is, however, subject to an added error $\varepsilon_i(x)$, as given in

$$f_{c_i}(x) = p(c_i|x) + \varepsilon_i(x). \tag{3}$$

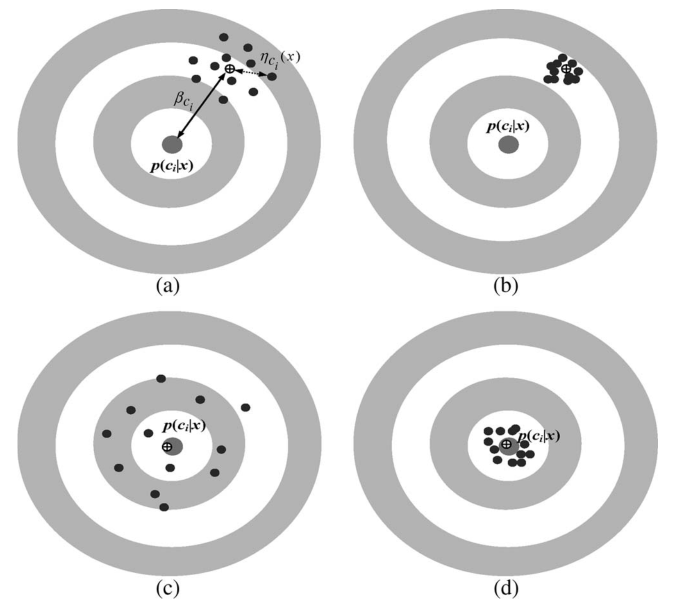

Fig. 3. Classifier bias and variance illustration in shooting arrows at a target. The center denotes the posterior probability $p(c_i|x)$, and the probability produced from a classifier C_n in classifying an instance x into class c_i, $f_{c_i}(x)$, is decomposed into three components: $p(c_i|x)$, β_{c_i}, and $\eta_{c_i}(x)$. Each black dot denotes a prediction of a classifier C_n on an instance x. Assume a set of classifiers $C_1, C_2, \ldots, C_n$ trained by using the same learning algorithm but different versions of the training data are used to predict an instance x, the bias β_{c_i} is the average offset of all classifiers' prediction on x, and the variance is the dispersion of one specific prediction with respect to the averaged prediction center (picture revised from [32]). (a) Bias and variance decomposition. (b) Large bias small variance. (c) Small bias large variance. (d) Small bias small variance.

If we consider that the added error of the classifier mainly comes from two sources, i.e., classifier bias and variance [30]–[33], then the added error $\varepsilon_i(x)$ in (3) can be decomposed into two terms, i.e., β_{c_i} and $\eta_{c_i}(x)$, where β_{c_i} represents the bias of the current learning algorithm, and $\eta_{c_i}(x)$ is a random variable that accounts for the variance of the classifier (with respect to class c_i), which gives [3], [20]–[22]

$$f_{c_i}(x) = p(c_i|x) + \beta_{c_i} + \eta_{c_i}(x). \tag{4}$$

In (4), β_{c_i} and $\eta_{c_i}(x)$ are essentially determined by the underlying learning algorithm and the examples used to train the classifiers. Existing analysis on bias and variance decomposition [29]–[31] has concluded that, given a specific learning algorithm and a training set T, if we build a set of classifiers (denoted by $C_1, C_2, \ldots, C_n$) from T, then all classifiers will share the same level of bias (which is the bias of the learning algorithm) but different levels of variances in predicting a test instance x.

To clearly state the bias and variance decomposition, we follow Moore and McCabe's illustration [32] and demonstrate the value of $f_{c_i}(x)$ in Fig. 3. Given a training set T and a specific learning algorithm, for example, C4.5 [26] or Naive Bayes [28], one can randomly sample T (with replacement) and generate a number of training set $T_1, T_2, \ldots, T_n$, each of which has the same number of instance as T. After that, one classifier is trained from each single data set, which results in n classifiers $C_1, C_2, \ldots, C_n$ in total. At the last step, the n classifiers are used to predict a specific instance x, with the class probabilities of each classifier (with respect to class c_i) denoted by a dot in Fig. 3 (for ease of understanding, we assume that each prediction corresponds to a 2-D point). In Fig. 3(a), the center denotes the Bayes posterior probability $p(c_i|x)$, whereas each prediction from classifier $C_1, C_2, \ldots, C_n$ is an offset value from the center. Because $C_1, C_2, \ldots, C_n$ are trained by using the same learning algorithm but different versions of the same data set, their predictions on a specific instance center around a specific value $\oplus$, which is the bias of the learning algorithm. The distance between each dot and $\oplus$ denotes the variance of the classifier's prediction on x.

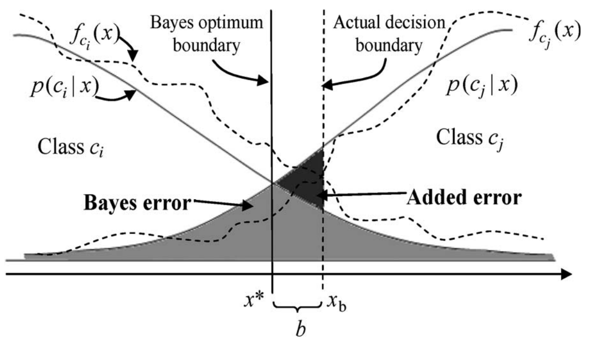

Fig. 4. Decision boundaries and error regions associated with approximating the *a posteriori* probabilities (picture revised from [21]).

Depending on the learning algorithms and the training set used to train the classifiers, one can expect that the overall predictions of $C_1, C_2, \ldots, C_n$ may vary among large bias small variance [Fig. 3(b)], small bias large variance [Fig. 3(c)], small bias small variance [Fig. 3(d)], and others. In summary, the above analysis indicates that classifiers trained by using the same learning algorithm but different versions of the training data suffer from the same level of bias but different variance values.

Assuming that we are using the same learning algorithm in our analysis, without loss of generality, we can ignore the bias term [3], [20]. Consequently, the learner's probability in classifying x into class c_i becomes

$$f_{c_i}(x) = p(c_i|x) + \eta_{c_i}(x). \tag{5}$$

Because the Bayes optimum decision boundary is the loci of all points x^* such that $p(c_i|x^*) = p(c_j|x^*)$, where $p(c_j|x^*) = \max_{k \neq i} p(c_k|x)$, the decision boundary of a classifier denoted by (5), which approximates $p(c_i|x)$, is also shifted from the optimum Bayes decision boundary, as shown in Fig. 4.

In Fig. 4, the actual decision boundary is denoted by x_b, the Bayes optimum boundary is denoted by x^*, and $b = x_b - x^*$ denotes the amount by which the boundary of the classifier differs from the optimum boundary. The darkly shaded region represents the area that is erroneously classified by the classifier f. Under mild regularity conditions, we can perform a linear approximation of $p(c_k|x)$ around x^* and express the density function $f_b(b)$ of b in terms of $\eta_{c_i}(x)$ [21]. Consequently, the expected added error of classifier f is given by

$$\mathrm{Err}_{\mathrm{add}} = \int_{-\infty}^{\infty} A(b) f_b(b)\, db \tag{6}$$

where $A(b)$ is the area of the darkly shaded region, and f_b is the density function for b.

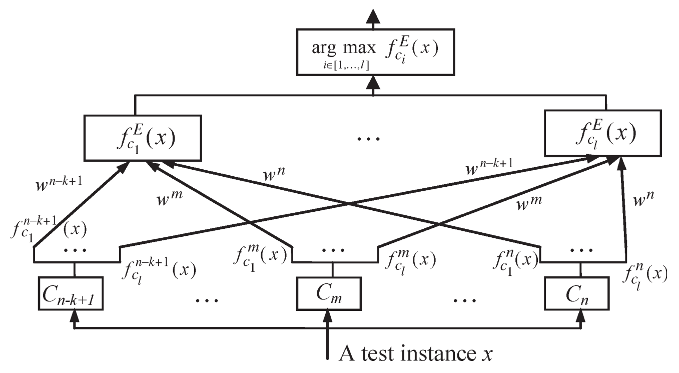

Fig. 5. classifier ensemble model consisting of k base classifiers $C_{n-k+1}, C_{n-k+2}, \dots, C_m, \dots, C_n$. Each base classifier C_m has an associated weight value w^m. The class probability of a test instance on class c_i is the sum of the weighted class probabilities over all base classifiers. The final prediction of x is the class with the largest class probability.

Tumer and Ghosh [20] showed that this quantity can be expressed as

$$\mathrm{Err}_{\mathrm{add}} = \frac{\sigma^2_{\eta_{c_i}} + \sigma^2_{\eta_{c_j}}}{s} = \frac{\sigma^2_{\eta_c}}{s} \tag{7}$$

where $p'(\cdot)$ denotes the derivative of $p(\cdot)$, $s = p'(c_j|x^*) - p'(c_i|x^*)$, which is independent of the trained model, and $\sigma^2_{\eta_{c_i}}$ denotes the variance of $\eta_{c_i}(x)$.

Equation (7) states that the expected added error of a classifier is proportional to its variance $\sigma^2_{\eta_c}$; thus, reducing this quantity reduces the classifier's expected error rate.

B. Classifier Ensemble Variance

Given a classifier ensemble E with k base classifiers (in this paper, each member of the ensemble is called a base classifier), the probability of the ensemble E in classifying an instance x is given by a linear combination of the probabilities produced by all of its base classifiers. Here, we employ a weighted ensemble framework, as shown in Fig. 5, where each classifier C_n has a weight value w^m. Under the framework in Fig. 5, the probability of E in classifying x into class c_i is given by (8), where $f^m_{c_i}(x)$ denotes the probability of base classifier C_m in classifying x into class c_i, i.e.,

$$\begin{aligned} f^E_{c_i}(x) &= \sum_{m=n-k+1}^{n} w^m f^m_{c_i}(x) \Bigg/ \sum_{m=n-k+1}^{n} w^m \\ &= p(c_i|x) + \sum_{m=n-k+1}^{n} w^m \eta^m_{c_i} \Bigg/ \sum_{m=n-k+1}^{n} w^m. \end{aligned} \tag{8}$$

This probability can be expressed as

$$f^E_{c_i}(x) = p(c_i|x) + \eta^E_{c_i}(x) \tag{9}$$

where $\eta^E_{c_i}(x)$ is a random variable accounting for the variance of the classifier ensemble E with respect to class c_i, and

$$\eta^E_{c_i} = \sum_{m=n-k+1}^{n} w^m \eta^m_{c_i} \Bigg/ \sum_{m=n-k+1}^{n} w^m. \tag{10}$$

Therefore, the variance of $\eta^E_{c_i}(x)$ is given by

$$\sigma^2_{\eta^E_{c_i}} = \frac{\sum\limits_{m=n-k+1}^{n} \sum\limits_{g=n-k+1}^{n} w^m w^g \mathrm{cov}\left(\eta^m_{c_i}, \eta^g_{c_i}\right)}{\sum\limits_{m=n-k+1}^{n} w^m \cdot \sum\limits_{g=n-k+1}^{n} w^g} \tag{11}$$

which can be rewritten as

$$\begin{aligned} \sigma^2_{\eta^E_{c_i}} = &\frac{\sum\limits_{m=n-k+1}^{n} (w^m)^2 \sigma^2_{\eta^m_{c_i}}}{\left(\sum\limits_{m=n-k+1}^{n} w^m\right)^2} \\ &+ \frac{\sum\limits_{m=n-k+1}^{n} \sum\limits_{g=n-k+1 g\neq m}^{n} w^m w^g \mathrm{cov}\left(\eta^m_{c_i}, \eta^g_{c_i}\right)}{\left(\sum\limits_{m=n-k+1}^{n} w^m\right)^2}. \end{aligned} \tag{12}$$

Assuming that $\eta^m_{c_i}$ and $\eta^g_{c_i}$ are independent for any classifier pairs C_g and C_m $(g \neq m)$, the second term in (12) is equal to zero, and the variance of $\eta^E_{c_i}(x)$ becomes

$$\sigma^2_{\eta^E_{c_i}} = \sum_{m=n-k+1}^{n} (w^m)^2 \sigma^2_{\eta^m_{c_i}} \Bigg/ \left(\sum_{m=n-k+1}^{n} w^m\right)^2 \tag{13}$$

where $\sigma^2_{\eta^m_{c_i}}$ denotes the variance of the random variable $\eta^m_{c_i}$. In our system, $\sigma^2_{\eta^m_{c_i}}$ is calculated by

$$\sigma^2_{\eta^m_{c_i}} = \frac{1}{|\Lambda_x|} \sum_{(x,c)\in\Lambda_x} \left(y^x_{c_i} - f^m_{c_i}(x)\right)^2 \tag{14}$$

where Λ_x is an evaluation set used to calculate the classifier variance, $|\Lambda_x|$ denotes the number of instances in Λ_x, and $y^x_{c_i}$ is the genuine class probability of instance x. If x is labeled as class c_i, then $y^x_{c_i}$ is equal to 1; otherwise, it is equal to 0. Consequently, the variance of the classifier C_m over all class $c_1, c_2, \dots, c_l$ is given by

$$\sigma^2_{\eta^m_c} = \sum_{i=1}^{l} \sigma^2_{\eta^m_{c_i}}. \tag{15}$$

The total variance of the ensemble E is then given by

$$\sigma^2_{\eta^E} = \sum_{i=1}^{l} \sigma^2_{\eta^E_{c_i}} = \sum_{m=n-k+1}^{n} (w^m)^2 \sigma^2_{\eta^m_c} \Bigg/ \left(\sum_{m=n-k+1}^{n} w^m\right)^2. \tag{16}$$

which is called the *classifier ensemble variance* in this paper. According to Tumer and Ghosh's conclusion in (7), a classifier's expected added error is proportional to its variance. Consequently, a classifier ensemble's expected error can be written as

$$\mathrm{Err}^E_{\mathrm{add}} = \frac{\sigma^2_{\eta^E}}{S}. \tag{17}$$

Equation (17) states that, to minimize the error rate of a classifier ensemble, we can minimize its variance instead. This objective can be achieved through the adjustment of the weight value associated with each of E's base classifier C_m.

C. Optimal-Weight Classifier Ensemble

Section III-B concludes that, to build a classifier ensemble E with minimum error rate, we need to find the weight values w^m, $m = n-k+1, \ldots, n$, such that the classifier ensemble variance defined by (16) can reach the minimum. This is equivalent to the problem given by

$$\arg\min_{w^m} \left(\sigma^2_{\eta_c^E}\right). \tag{18}$$

To find the solution for the problem given in (18), we define that the weight value of the base C_m, w^m, is inversely proportional to the sum of C_m's variance $\sigma^2_{\eta^m_{c_i}}$ over all of the l classes as

$$w^m = \mu^m \Big/ \left(\sigma^2_{\eta_c^m}\right) \tag{19}$$

where μ^m is a coefficient that grants one more degree of freedom in addition to $\sigma^2_{\eta_c^m}$ defining w^m. Combining (19) and (16), the ensemble variance defined by (16) can be rewritten as

$$\sigma^2_{\eta^E} = \sum_{m=n-k+1}^{n} \left\{(\mu^m)^2 \cdot \sigma^2_{\eta_c^m}\right\} \Big/ \left(\sum_{m=n-k+1}^{n} \mu^m\right)^2. \tag{20}$$

Now, letting $p^m = \mu^m/\Sigma\mu^m$, the optimization problem in (18) can be formulated as finding an optimal solution $\hat{p}^m$ such that

$$\begin{aligned} \text{Min} \quad & \sigma^2_{\eta^E} = \sum_{m=n-k+1}^{n} \left\{(p^m)^2 \cdot \sigma^2_{\eta_c^m}\right\} \\ \text{s.t.}: \quad & \sum_{m=n-k+1}^{n} p^m = 1. \end{aligned} \tag{21}$$

This mathematical programming model can easily be solved explicitly by using a Lagrange multiplier, where the Lagrange function is given as

$$L(p^m, \lambda) = \sum_{m=n-k+1}^{n} \left\{(p^m)^2 \cdot \sigma^2_{\eta_c^m}\right\} + \lambda\left(\sum_{m=n-k+1}^{n} p^m - 1\right). \tag{22}$$

Taking partial derivative on p^m, $m = n-k+1, \ldots, n$, we have

$$\begin{cases} \frac{\partial L(p^{n-k+1},\lambda)}{\partial p^{n-k+1}} = 2p^{n-k+1} \cdot \sigma^2_{\eta_c^{n-k+1}} + \lambda = 0 \\ \cdots \\ \frac{\partial L(p^n,\lambda)}{\partial p^n} = 2p^n \cdot \sigma^2_{\eta_c^n} + \lambda = 0 \end{cases}. \tag{23}$$

Letting $h^m = \sigma^2_{\eta_c^m}$, the above equations are equivalent to

$$p^i h^i = p^j h^j \qquad \forall\, i \neq j. \tag{24}$$

According to (22) and the constraint $\sum_{m=n-k+1}^{n} p^m = 1$ in (21), we have

$$p^m = \frac{1}{1 + \sigma^2_{\eta_c^m}\left(\sum_{t=n-k+1, t\neq m}^{n} 1 \Big/ \sigma^2_{\eta_c^t}\right)}. \tag{25}$$

Because the value of w^m is proportional to p^m, we can directly use the p^m values calculated from (25) as the ensemble weight value w^m, $m = n-k+1, \ldots, n$, for each individual base classifier of the classifier ensemble E. Because the whole process ensures that the weight values are determined such that the variance of the classifier ensemble can reach the minimum, the ensemble classifier is guaranteed to have the lowest error rate.

V. MV Active Learning From Stream Data

In stream data environments, because labeling all instances in each data chunk is out of the question, alternative solutions must determine that given the number of instances for labeling, which instances should be labeled for each chunk S_n, such that the labeled instances can build a classifier ensemble with maximum accuracy in predicting future samples?

To explicitly answer the above question, we shall recall that Tumer and Ghosh [20] have concluded that a classifier's expected added error (i.e., the error in addition to the Bayesian error) is proportional to its variance, so reducing the variance directly reduces the classifier's expected error rate. Following this conclusion, we can assert that the variance of an ensemble classifier is also proportional to its added error. Assuming that the weight values of the base classifiers are properly selected, the variance corresponding to each individual instance can then be used to assess whether the instance is "uncertain" w.r.t. the current ensemble classifier. More specifically, if an instance contributes a small variance value, then it means that the base classifiers are consistent in classifying the instance, or in other words, the knowledge of the instance is already hard coded in the classifiers. Consequently, labeling this instance will most likely not provide much "fresh" information to enhance the ensemble classifier. On the other hand, if an instance has a large variance value, then it means that the base classifiers are inconsistent in classifying the instance, possibly because classifiers do not have sufficient knowledge for prediction (e.g., due to the concept drifting). Therefore, labeling such samples will most likely provide valuable information to help improve the ensemble classifier.

Based on the above analysis, we propose a *minimum-variance* (MV) principle for active learning in the data stream, where the intuition is to label instances that are responsible for the large ensemble variance values. We expect that labeling those instances can reduce the variance of the ensemble classifier and eventually minimize its error rate. Fig. 6 shows the proposed framework, where the whole process consists of three major steps: 1) initialization; 2) instance labeling; and 3) calculating optimal weight values for ensemble learning. A loop between steps 2) and 3) continuously repeats with one step building on the results of the other step. More specifically, the instance labeling and the ensemble weight updating are mutual-beneficial procedures with expectation maximization logics as follows:

1) E-Step (Steps 7 to 9 in Fig. 6): Given a set of weight values for a classifier ensemble, we use them to find instances from the data stream for labeling (this is the interest of active learning).
2) M-Step (Step 10 in Fig. 6): Given a number of labeled samples, we use them to determine the optimal weight values for the ensemble classifier, such that the overall

```
Procedure: Active Learning from Data Streams
Given: (1) current data chunk $S_n$; and (2) k-1 classifiers $C_{n-k+1}$, .., $C_m$, .., $C_{n-1}$ built from the most recent data chunks;
Parameters: (1) $\alpha$, the percentage of instances should be labeled from $S_n$; (2) e, # of epochs in labeling $\alpha$ percentage of instances from $S_n$;
Objective: Label $\alpha$ of instances in $S_n$, such that the classifier built from $L_n$, denoted by $C_n$, along with the previous k-1 classifier can form a classifier
           ensemble as accurate as possible (in terms of its accuracy on unlabeled instances in $S_n$).
   1.  $L_n \leftarrow \emptyset$;  $U_n \leftarrow S_n$
   2.  $L_n \leftarrow$ Randomly label a tiny portion, e.g. 1~2.5%, of instances from $U_n$.
   3.  $\kappa \leftarrow 0$                                        // recording the total number of labeled instances
   4.  Build a classifier $C_n$ from $L_n$
   5.  $K \leftarrow 0$                                        // recording the number of instances labeled in the current epoch
   6.  Initialize weight value $w^m$ for each classifier $C_m$, m=n-k+1,.., n, where $w^m$ is equal to $C_m$'s prediction accuracy on $L_n$.
   7.  Use $C_{n-k+1}$,..,$C_m$, ..., $C_n$ to form a classifier ensemble E as shown in Figure 5.
   8.  For each instance $I_x$ in $U_n$
           a.  Use ensemble E to predict a class label for $I_x$.
           b.  Build an evaluation set $\Lambda_x = L_n \cup \hat{I}_x$, where $\hat{I}_x$ denotes $I_x$ with a predicted class label.
           c.  Calculate each classifier $C_m$'s variance on $\Lambda_x$ (Eqs. (14) and (15)), and feed the value into Eq. (16) to calculate ensemble variance and
               use this value as instance $I_x$'s expected ensemble variance on E.
       EndFor
   9.  Choose instance $I_x$ in $U_n$ with the largest ensemble variance, label $I_x$, and put labeled $I_x$ into $L_n$, i.e., $L_n = L_n \cup I_x$; $U_n = U_n \setminus I_x$
   10. Recalculate the ensemble variance of each base classifier $C_m$ on $L_n$ and find optimum weight value $w^m$ for all base classifiers
           a.  Calculate each classifier $C_m$'s variance on $\Lambda_x = L_n$ (Eqs. (14) and (15))
           b.  Use Eq. (25) to find $p^m$ values, m=n-k+1, ..., n
           c.  Use $p^m$ as optimal weight values $w^m$ for each base classifiers
           d.  Update classifier ensemble E by using the new weight values
   11. $\kappa \leftarrow \kappa+1$;  $K \leftarrow K+1$
   12. If $\kappa \geq |S_n| \cdot \alpha$
           a.  Exit                                        // exist if $\alpha$ percentage of instance are labeled
   13. Else if $K \geq (|S_n| \cdot \alpha / e)$
           a.  Repeat step 4                               // rebuild model $C_n$ if one epoch ends
   14. Else
           a.  Repeat step 8                               //continue instance labeling without rebuilding $C_n$
```

Fig. 6. MV active learning from stream data.

prediction accuracy can be maximized (i.e., the interest of weight updating for ensemble learning).

Assuming that the most recent $k-1$ classifiers are denoted by $C_{n-k+1}, C_{n-k}, \ldots, C_{n-1}$, upon the arrival of a data chunk S_n, our algorithm initiates an active learning process on S_n to selectively label α percent of instances from S_n such that the classifier built from the labeled instances in S_n along with the previous $k-1$ classifiers can form an accurate classifier ensemble. Denoting L_n and U_n as the labeled and unlabeled instance subsets of S_n, respectively, since none of the data in S_n have labels in the beginning, we set $L_n \leftarrow \emptyset$ and $U_n \leftarrow S_n$ on step 1) in Fig. 6. After that, we randomly label a tiny portion of instances in S_n and put them into L_n; this process is followed by a learning procedure that builds a classifier C_n from L_n. The k classifiers $C_{n-k+1}, C_{n-k}, \ldots, C_{n-1}, C_n$ form an ensemble E, where the initial weight value w^m of each base classifier C_m, $m = n-k+1, \ldots, n$, is set as the prediction accuracy of the classifier C_m on L_n (obviously, E is not optimized at this stage). After that, our algorithm must determine which instances in U_n should be labeled such that upon the accomplishment of the labeling process the classifier ensemble E has the minimal error rate on the remaining instances in S_n. Accordingly, the instance labeling process uses the MV principle to check each unlabeled instances in U_n and selects those with the largest ensemble variance value for labeling.

According to (14) and (15), the variance of a classifier ensemble is based on its base classifiers' variance on a specific evaluation set Λ_x. For each unlabeled instance I_x in U_n, its evaluation set consists of all labeled instances in L_n as well as I_x itself, e.g., $\Lambda_x = L_n \cup I_x$. Because the calculation of $\sigma^2_{\eta^m_c}$ requires that each instance in Λ_x should have a class label, and I_x's label is yet to be found, we will use E to assign a class label for I_x (which might be incorrect). We then use (16) to calculate the ensemble variance on Λ_x, which is treated as the ensemble variance of I_x, as shown on Steps 8(a) to 8(c).

After the calculation of the ensemble variance for all instances in U_n, we are now able to screen all unlabeled instances in S_n and label the ones with the largest ensemble variance value, with the labeled instance I_x moved from U_n to L_n. After that, we recalculate the optimal weight value for each base classifier by using the updated evaluation set L_n to ensure the minimum error rate of the classifier ensemble E. The weight-updating process can also be beneficial for instance labeling in the next round. For this purpose, we recalculate each base classifier C_m's variance on L_n (the one calculated in Step 8(c) is not accurate in the sense that the class label of I_x is not labeled but predicted from E). The variance values of the base classifiers are used to solve (25) and calculate new weight values w^m for each base classifier.

Following the weight updating, the algorithm checks the following three conditions to adjust the iterative labeling process: 1) whether the user-specified number of instances have been labeled (Step 12); 2) whether a new classifier C_n should be rebuilt after a certain number of instances are labeled (Step 13); and 3) whether the algorithm should repeat and label the next instance (Step 14).

A. Concept Drifting

The MV active learning framework in Fig. 6 can effectively address concept drifting challenges in data streams. This can be justified from both local and global perspectives. Locally, since the classifier C_n is learned from the most recent data chunk S_n, C_n has a smaller variance on L_n compared with the other $k-1$ classifiers built from previous data chunks. The solutions to (25) will show that a classifier with a smaller variance on L_n will receive a larger weight value and thus plays a more important role in the classifier ensemble. If the concepts in chunk S_n are significantly different from the other chunks, then the large weight value associated with C_n will force our algorithm to favor instances with large variance values w.r.t. C_n. Because the most recent classifier is believed to be mostly relevant to the current concept, assigning a large weight value to C_n helps our algorithm locally adapt to concept drifting in S_n. From the global point of view, a classifier ensemble consists of a set of weighted base classifiers; as concepts evolve over time, we only need to adjust the weight value of each base classifier. According to (25), the weight value of the base classifier is inversely proportional to its variance on chunk S_n. Assuming that the concept in chunks S_n is similar to the chunk S_{n-k+1} but distinct from the rest of the chunks of the ensemble (i.e., $S_{n-k+2}, \ldots, S_{n-1}$), according to (25), the weight value of C_{n-k+1} is going to be significantly higher than the weights of the rest of chunks. By doing so, the whole active learning framework can quickly adapt to the drifting concepts in data streams.

B. Speed Enhancement

The algorithm in Fig. 6 updates weight values once for each single labeled instance. To speed up the process, one can label multiple, for example, τ, instances in one time (select the ones with the largest variance) and use all the τ-labeled instances (along with instances in L_n) to find optimal weight values for the classifier ensemble (i.e., the weight values are updated once every τ instances). For example, assume that the number of instances in S_n is 10 000, and the user has specified $\alpha = 0.1$ and $e = 5$. It means that we should label 10% of instances (1000) in five epochs, and the classifier C_n is retrained from L_n after the labeling of every 200 instances (one epoch). If the user specifies $\tau = 20$, then the algorithm will update the ensemble weight values once in every 20 instances.

C. Time Complexity

We carry out the time complexity analysis under following assumptions: 1) A quadratic time complexity learning algorithm is employed in the system. In other words, given a training set with N (labeled) instances, it takes $O(N^2)$ computational time to train a classifier from the data. In addition, we also assume that the classifier has linear time complexity for prediction, so it takes $O(N)$ for a classifier to classify N instances. 2) The classifier ensemble E consists of k base classifiers built from k consecutive chunks, each of which contains N instances. The active learning process is supposed to label α percent of instances from a data chunk S_n in e epochs. 3) The active learning process labels one instance each time, and the weight values w^m, $m = n-k+1, \ldots, n$, update after the labeling of each single instance (notice that this assumption gives the upper bound of the proposed algorithm).

Under the above assumptions, the time complexity of the algorithm on a specific data chunk S_n can be decomposed into two major parts: $B(n)$ and $U(n)$, where $B(n)$ denotes the time complexity for model training, and $U(n)$ denotes the time complexity for active learning and ensemble weight updating.

The value of $B(n)$ is determined by the number of times retraining the classifier C_n and the number of instances used to train the classifier. Following the above assumptions, we know C_n is retrained at the end of each epoch, so we need to train C_n for e times (including the first time), and the number of training examples for each time is given by 0,[1] i.e., $N \cdot \alpha/e, 2 \times N \cdot \alpha/e, \ldots, (e-1) \times N \cdot \alpha/e$, respectively. Therefore, the total time complexity for model training is given by

$$\begin{aligned} B(n) &= O\left(\frac{N^2 \cdot \alpha^2 \cdot 1^2}{e^2}\right) + \cdots + O\left(\frac{N^2 \cdot \alpha^2 \cdot (e-1)^2}{e^2}\right) \\ &= O\left(\frac{N^2 \cdot \alpha^2}{e^2} \cdot \sum_{t=1}^{e-1} t^2\right) = O(N^2 \cdot \alpha^2 \cdot e). \end{aligned} \tag{26}$$

The value of $U(n)$ is determined by the time complexity of instance selection and weight updating, as well as the number of times the whole process repeats. If one instance is labeled each time, then the labeling and weight updating process need to repeat $N \cdot \alpha$ times (again, we assume that the number of initially randomly labeled instances is 0). In each repetition, active learning needs to calculate the variance of all base classifiers on the labeled subset [(14) and (15)], and the weight updating process needs to recalculate the variance based on newly labeled instances. Because the size of the labeled instance subset grows continuously, according to (14), a classifier's variance on the evaluation set Λ_x can be calculated incrementally by adding the value of the new instance to the previous results. Accordingly, assuming that the size of Λ_x grows from 0 to $N \cdot \alpha$, the total time complexity is $2 \times O(l \cdot k \cdot N) = O(l \cdot k \cdot N)$.

For weight updating, the calculation of p^m, $m = n-k+1, \ldots, n$, in (25) takes $2 \times O(k)$ for all k classifiers (it takes $O(k)$ to calculate the summation in (25) and $O(k)$ to calculate the p^m values). In total, the time complexity in each repetition for weight updating is $2 \times O(k)$. Because the whole weight updating process needs to be repeated $N \cdot \alpha$ times, the total time complexity for weight calculation is $2 \times O(N \cdot \alpha \cdot k)$. As a result, the time complexity for active learning and weight updating is given by

$$U(n) = 2 \times O(l \cdot k \cdot N + \alpha \cdot l \cdot N). \tag{27}$$

Assuming that a data stream has M data chunks, the total time complexity $T(n)$ is given by

$$\begin{aligned} T(n) &= M \times (B(n) + U(n)) \\ &= O(N^2 \cdot M \cdot \alpha^2 \cdot e) + O\left(N \cdot M \cdot l \cdot (k + \alpha)\right). \end{aligned} \tag{28}$$

To further simplify (28), we can consider practical parameter settings, where M and N are relatively large, whereas e, l, and k are usually small (α is a numerical value between 0 and 1).

[1]For simplicity, we assume that the number of initially randomly labeled instances on Step (2) in Fig. 6 is very small, which is close to zero.

Consequently, we can safely assume that $l \cdot (k+\alpha) \leq N$ and $\alpha^2 \cdot e \leq 1$, which gives

$$\begin{aligned} T(n) &= O(N^2 \cdot M \cdot \alpha^2 \cdot e) + O\left(N \cdot M \cdot l \cdot (k+\alpha)\right) \\ &\leq 2 \times O(N^2 \cdot M). \end{aligned} \tag{29}$$

Equation (29) shows that the total time complexity is bounded by two important factors: 1) the number of instances in each chunk N and 2) the number of data chunks M. Because model training in each chunk is nonlinear w.r.t. the chunk size, we may prefer a relatively small chunk size to save the computational cost.

VI. Experimental Comparisons

A. Experimental Settings

1) General Setting: To compare different active learning methods and assess the quality of the labeled instances, we train a classifier ensemble E for each method by using the framework in Fig. 2. Therefore, if one classifier ensemble outperforms its peers, then we can safely conclude that this is because the instances used to train the classifier ensemble are of better quality. Notice that different active learning methods may select different portions of instances from S_n, which may lead to different test sets for validation. For valid comparisons, we have to ensure that all methods are compared on the same test set. Therefore, our comparisons are based on the average prediction accuracies on all instances in chunk S_n over the whole data stream.

2) Data Streams:

Synthetic data: To simulate data streams on which we can fully control the concept drifting speed and magnitude, we employ a hyperplane-based synthetic data stream generator that is popularly used in stream data mining research [3], [7]–[9]. The hyperplane of the data generation is controlled by a nonlinear function defined by (30) (we intentionally use a nonlinear hyperplane to challenge active learning methods). Given an instance x, its class label is determined by the $f(x)$ value given in (30). Assuming that an $f(x)$ value larger than a threshold a_0 indicates that x belongs to class A, otherwise x belongs to class B, then changing the values of $a_i, i = 1, \ldots, d$, and threshold a_0 may lead to different posteriori probabilities $p(c_i|x)$ for instance x. In an extreme situation, it may result in different class labels for two identical instances. By doing so, we are introducing dynamic and variant decision boundaries into the data to simulate concept drifting in data streams, e.g.,

$$f(x) = \sum_{i=1}^{d} \frac{a_i}{x_i + (x_i)^2}. \tag{30}$$

In (30), d is the total dimensions of the input data x. Each dimension $x_i, i = 1, \ldots, d$, is a value randomly generated in the range of [0, 1]. A weight value $a_i, i = 1, \ldots, d$, is associated with each input dimension, and the value of a_i is initialized randomly in the range of [0, 1] at the beginning. In the data generation process, we gradually change the value of a_i to simulate concept drifting by using the following three parameters [2], [7], [8]: 1) t, controlling the magnitude of concept drifting (in every N instances); 2) p, controlling the number of attributes involved in the change; 3) h and $n_i \in \{-1, 1\}$, controlling the weight adjustment direction for attributes involved in the change. After the generation of each instance x, a_i is adjusted continuously by $n_i \cdot t/N$ (as long as a_i is involved in the concept drifting), and the value a_0 is recalculated to change the decision boundaries (concept drifting). Meanwhile, after the generation of N instances, there is an h percentage of chances that weight change will invert its direction, i.e., $n_i = -n_i$ for all attributes a_i involved in the change. In summary, c2–I100k–d10–p5–N1000–t0.1–h0.2 denotes a two-class data stream with 100k instances, each containing ten dimensions. Concept drifting involves five attributes, and their weights change with a magnitude of 0.1 in every 1000 instances, and weight inverts the direction with 20% of chance.

TABLE II
Data Characteristics of the Real-World Data Used for Evaluation

Name	# of instances	# of attributes	# of classes	Majority:Minority class ratio	Accuracy (%)	
					C4.5	NB
Adult	48,842	15	2	0.761:0.239	85.17	82.38
Covtype	581,012	55	7	0.488:0.005	89.56	66.24
Letter	20,000	17	26	0.041:0.037	87.98	64.12

Real-world data: We select three relatively large data sets from the UCI data repository [27] and treat them as data streams for active learning. The data sets we selected are Adult, Covtype, and Letter (as listed in Table II). The domain information about the three data sets is introduced in the technical report [60]. To help interested readers capture the learning complexity of these data sets, Table II also lists the tenfold cross-validation accuracies of the classifiers built from all instances (based on C4.5 and naive Bayes classifiers).

3) Benchmark Methods: For comparison purposes, we implemented the four methods introduced in Section III, including RS, LU, GU sampling, and AM. The proposed MV-based active learning method is denoted by MV. In our implementation, AM follows exactly the GU framework except that its instance selection procedure is replaced by using AM [9] to select instances with the most significant distributional changes for labeling. For fairness of the comparisons, all classifier ensembles use exactly the same architecture as shown in Fig. 2. The number of base classifiers is the same for all methods, and the weight of each base classifier is determined by its prediction accuracy on subset L_n. The parameter settings for all methods, e.g., chunk size α and epoch value e, are the same except for RS, which labels all instances in one epoch. Classifier C_n is retrained at the end of each epoch for LU, GU, AM, and MV. Each time MV updates its weights, we also update the weights for LU, GU, and AM by using instances in L_n, so we can avoid MV taking advantage of doing more updating than LU, GU, and AM.

All methods are implemented in Java platform. The learning algorithms employed include C4.5 [26] and Naive Bayes [28], which are directly imported from the WEKA data mining tool [25]. Due to page limitations, majority of the results reported in this paper are based on the C4.5; interested readers can refer to the technical report [60] for detailed results and project source codes.

TABLE III
AVERAGE CLASSIFICATION ACCURACY ON DIFFERENT DATA STREAMS (C4.5). (a) ACCURACY ON c2–I50k–d10–p5–N1000–t0.1–h0.2 ($\alpha = 0.1$). (b) ACCURACY ON c3–I50k–d10–p5–N1000–t0.1–h0.2 ($\alpha = 0.1$). (c) ACCURACY ON c4–I50k–d10–p5–N1000–t0.1–h0.2 ($\alpha = 0.1$)

(a)

Chunk Size	***RS***	***LU***	***GU***	***AM***	***MV***
250	$74.54_{\pm 2.89}$	$73.70_{\pm 2.75}$	$75.51_{\pm 3.35}$	$73.93_{\pm 3.02}$	$\mathbf{81.84_{\pm 2.23}}$
500	$83.07_{\pm 2.42}$	$82.56_{\pm 2.36}$	$85.16_{\pm 2.95}$	$82.89_{\pm 2.87}$	$\mathbf{87.62_{\pm 2.18}}$
750	$86.10_{\pm 2.75}$	$85.69_{\pm 3.13}$	$88.14_{\pm 3.04}$	$85.84_{\pm 2.93}$	$\mathbf{89.48_{\pm 2.12}}$
1000	$86.43_{\pm 3.52}$	$86.11_{\pm 4.12}$	$88.79_{\pm 3.47}$	$86.79_{\pm 3.42}$	$\mathbf{90.12_{\pm 3.02}}$
2000	$86.47_{\pm 5.01}$	$85.94_{\pm 4.82}$	$88.69_{\pm 4.27}$	$86.62_{\pm 4.49}$	$\mathbf{89.16_{\pm 3.28}}$

(b)

Chunk Size	***RS***	***LU***	***GU***	***AM***	***MV***
250	$56.36_{\pm 2.71}$	$55.46_{\pm 2.64}$	$56.57_{\pm 2.71}$	$57.93_{\pm 3.12}$	$\mathbf{66.38_{\pm 2.02}}$
500	$66.31_{\pm 4.30}$	$66.11_{\pm 4.23}$	$66.99_{\pm 4.98}$	$65.84_{\pm 4.94}$	$\mathbf{71.95_{\pm 3.29}}$
750	$71.79_{\pm 2.86}$	$70.49_{\pm 3.61}$	$72.59_{\pm 3.77}$	$69.40_{\pm 3.51}$	$\mathbf{75.00_{\pm 3.33}}$
1000	$75.29_{\pm 2.88}$	$74.23_{\pm 3.27}$	$76.45_{\pm 3.72}$	$74.14_{\pm 3.62}$	$\mathbf{79.01_{\pm 2.32}}$
2000	$73.92_{\pm 6.11}$	$72.97_{\pm 6.01}$	$76.37_{\pm 4.64}$	$74.07_{\pm 5.23}$	$\mathbf{76.76_{\pm 4.79}}$

(c)

Chunk Size	***RS***	***LU***	***GU***	***AM***	***MV***
250	$42.27_{\pm 2.01}$	$41.38_{\pm 1.80}$	$41.34_{\pm 2.09}$	$41.67_{\pm 2..11}$	$\mathbf{47.21_{\pm 1.56}}$
500	$50.47_{\pm 2.34}$	$49.81_{\pm 2.47}$	$49.91_{\pm 2.45}$	$49.29_{\pm 2.52}$	$\mathbf{54.51_{\pm 2.17}}$
750	$58.11_{\pm 3.65}$	$56.75_{\pm 3.26}$	$56.48_{\pm 3.09}$	$57.91_{\pm 3.24}$	$\mathbf{60.12_{\pm 4.02}}$
1000	$63.08_{\pm 3.67}$	$62.72_{\pm 4.07}$	$61.92_{\pm 3.65}$	$64.25_{\pm 3.71}$	$\mathbf{65.14_{\pm 3.06}}$
2000	$71.87_{\pm 4.47}$	$70.67_{\pm 4.98}$	$70.98_{\pm 5.09}$	$72.13_{\pm 4.96}$	$\mathbf{73.09_{\pm 3.77}}$

B. Experimental Results

1) Active Learning With a Fixed α Value: We use C4.5 as the base learner and apply active learning to three types of synthetic data streams (two-class [Table III(a)], three-class [Table III(b)], and four-class [Table III(c)]) and report the results in Table III. The accuracies in the tables denote an ensemble classifier's average accuracy in predicting instances in the current data chunk S_n, with chunk sizes varying from 250 to 2000. We fix the α value to 0.1 and set the k value to 10, which means that 10% of the instances are labeled for each data chunk, and only the most recent ten classifiers are used to form a classifier ensemble.

The results from Table III indicate that the performance of all methods deteriorates as a consequence of the shrinking chunk size. This is because a smaller data chunk contains fewer examples, and sparse training examples usually produce inferior learners in general. The advantage of having a small chunk size is the training efficiency. This is particularly significant for learning algorithms with nonlinear computational complexity. As shown in (29), the time complexity of our algorithm nonlinearly increases w.r.t. the number of instances in each chunk.

For any particular method, Table III indicates that the results in multiclass data streams are significantly worse than a binary class data stream, although all data streams are generated from the same type of hyperplanes [defined by (30)]. This shows that active learning from a multiclass data stream is more challenging than a binary class data stream.

Comparing all five methods, we can easily conclude that MV receives the best performance across all data streams. The LU-based method is not an option for active learning from data streams, and its performance is constantly worse than RS regardless of whether the underlying data are binary or multiple classes. Although GU outperforms RS quite often, for multiclass data streams (e.g., four classes), its performance is

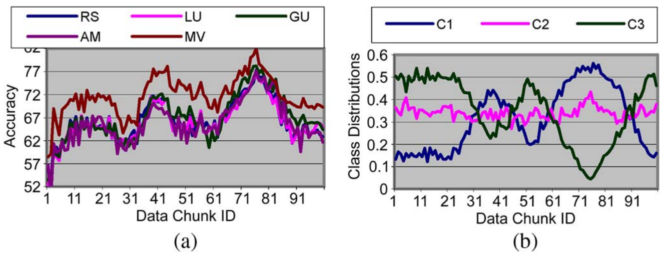

Fig. 7. (a) Classifier ensemble accuracy and (b) class distributions across different data chunks (c3–I50k–d10–p5–N1000–t0.1–h0.2, $\alpha = 0.1$, $e = 3$, data chunk size: 500, C4.5).

unsatisfactory and is almost always inferior to RS. The results from RS are surprisingly good, and it is generally quite difficult to beat RS with a substantial amount of improvement [the largest improvement of MV over RS we observed is 9.81%, which is from a two-class data stream in Table III(b)]. As we have analyzed in Section III, RS naturally addresses the challenges of varying class distributions in data streams by randomly labeling instances in the data chunk. As a result, it produces a subset with the most similar class distributions to the genuine distributions in the original chunk.

For the AM-based sampling method, we observed that its performance is worse or marginally better than random selection and worse than GU most of the time. Since AM strictly follows the GU framework, except that the instance selection procedure is replaced by using a distribution-based measure (instead of the uncertainty measure), this concludes that instance distribution is less effective than uncertainty-based measures for finding the most "important" samples for labeling. We believe that the reason behind this is twofold: 1) AM relies on sample distributions to select "important" instances for labeling. Notice that sample distributions do not necessarily have a direct connection to indicate whether an instance is "important" for labeling or not, even if the decision tree does accurately capture the data distributions. Consider a two-class sample set with genuine decision boundary denoted by $x = 0$, the vertical line (i.e., instances with $x \leq 0$ are classified as c_1, or c_2 otherwise). Given a labeled two-instance sample set $S_1 = \{c_1 : x_1 = -1, c_2 : x_2 = 0\}$, which outputs a one-node decision tree with two leaves (splitting at -0.5), the distribution of S_1 on the tree is $(c_1 : 0.5, c_2 : 0.5)$. Given another two unlabeled sets $S_2 = \{x_3 = -3, x_4 = -4\}$ and $S_3 = \{x_5 = -1, x_6 = 1\}$, their distributions on the tree are $(c_1 : 1.0, c_2 : 0.0)$ and $(c_1 : 0.5c_2 : 0.5)$, respectively. According to the AM principle, one should label S_2 instead of S_3 because the former has a larger distributional difference from S_1. It is, however, very clear that labeling samples in S_2 does not provide any additional information to enhance the current decision boundary $(x' = -0.5)$ toward the genuine decision boundary $(x = 0)$, whereas labeling S_3 can indeed offer help. 2) It is well known that decision trees are sensitive to the training set, and changing one or multiple training samples can produce trees with radically different structures. Consequently, even if instances with significant distributional changes are worth labeling, using a decision tree may not be able to capture the genuine distributions of the underlying data.

To compare different methods at individual data chunk level, we record each method's accuracy on each single data chunk and report the results (ten times average) in Fig. 7(a), where the x-axis represents data chunk ID in its temporal order, and the

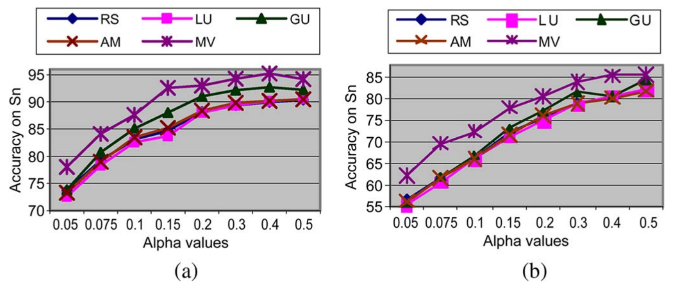

Fig. 8. Classifier ensemble accuracy with respect to different α values: (a) two-class and (b) three-class (chunk size 500, $e = 3$, C4.5). (a) c2–I50k–d10–p5–N1000–t0.1–h0.2. (b) c3–I50k–d10–p5–N1000–t0.1–h0.2.

y-axis shows the classifier ensemble accuracy (chunk size 500). Meanwhile, to demonstrate the impact of the changing priori class distributions on data streams, we record class distributions of each data chunk and report the values in Fig. 7(b). The results in Fig. 7 indicate that the accuracies of the data chunks vary significantly across data streams. Comparing the accuracies to the corresponding class distributions reveals a clear connection between them. If a data stream is experiencing a significant class distribution change, it immediately impacts on the classification accuracy.

The results from Fig. 7(a) show that MV consistently outperforms other methods across all data chunks, which asserts that the advantage of MV can be observed across different types of data streams (binary class and multiclass) and different chunks across the whole data stream.

2) Active Learning With Different α Value: In Fig. 8, we compare all four methods w.r.t. different α values. Not surprisingly, when the α value increases, all methods gain better prediction accuracies. This is because the increasing number of labeled instances helps build strong base classifiers. Overall, MV and GU achieve the best performance, and LU is inferior to RS in the majority of the cases. Both MV and GU label instances for the benefit of a global classifier ensemble but from different perspectives. GU labels instances with the largest uncertainty w.r.t. the current classifier ensemble, whereas MV labels instances with the largest ensemble variance. The main difference is that GU intends to label instances that have the largest mean uncertainty over all base classifiers, whereas MV prefers to label instances contradictory to base classifiers.

As discussed in Section III, GU extends QBC to data streams by using classifiers learned from different data chunks as committee members. In the original QBC, the committee members are learned from the data with the same distributions (randomly sampled from the labeled data); therefore, committee members are similar to each other with relatively small variances in their predictions. In data stream environments, the committee classifiers are learned from different portions of stream data, and the concept drifting in data chunks also renders classifiers significantly different from each other. As a result, the weighted average uncertainty over all committee members may not reveal the genuine uncertainty of the ensemble formed by them. The results in Fig. 8 support our hypothesis very well. As we can see, MV constantly outperforms GU across all α values. For multiclass data streams, the performance of GU is unsatisfactory and is largely inferior to the RS. This becomes extremely clear in the next section, where GU's performance on a real-world 26-class data stream is significantly worse than RS and MV, particularly when a small portion of instances are labeled.

3) Active Learning From Real-World Data: In Fig. 9, we report the algorithm performance on three real-world data. Different from synthetic data streams, where the decision concepts in data chunks gradually change and follow the formula given in (30), the real-word data do not share such concept drifting property among data chunks, and in fact, we do not even know the genuine concepts underneath the real-world data.

The results in Fig. 9 again assert that MV consistently outperforms other methods. Although GU is able to perform well for both Adult and Covtype, its performance on Letter is significantly worse than all the other methods, where its accuracy can be as much as 20% lower than MV. Considering that Letter is a sparse data set with 26 evenly distributed classes, this observation supports our analysis in Sections III and VI-B, where for sparse data streams with a large number of classes, the classifiers built from different data chunks vary significantly. Simply calculating the averaged uncertainty over all committee classifiers (like QBC does [13], [14]) is not a good solution and can produce much worse results than RS.

4) Active Learning From Noisy Stream Data: The adoption of the chunk-based ensemble learning framework for data streams raises a possible concern that the whole framework may be sensitive to attribute noise, particularly when the size of the data chunk is small. This is because of the following: 1) a classifier learned from a small chunk may be overfit to the noisy data, and 2) the instance selection process may focus on "noisy" samples since they are likely the ones responsible for a high variance value. To assess the system performance on a noisy stream data, we employ a random noise injection process [23] to corrupt the stream data. Given an instance x and an attribute noise level, for example $p = 0.1$, the noise is injected such that each attribute value of x has a probability of p to be changed to another randomly selected value. Therefore, if x has ten attributes and $p = 0.1$, it means that, on average, at least one attribute of x is noise corrupted.

In Fig. 10, we report the results on two data streams, with the attribute noise varying from 0 to 0.25. Overall, attribute noise brings significant impact to the underlying methods, where for as little as 5% attribute noise, the prediction accuracy can deteriorate up to 6%. When comparing RS to MV, we can find that MV constantly outperforms RS across all noise levels, and their performances both deteriorate, as the noise levels increase, at almost the same rate. This asserts that compared to random selection, which has no special treatment for noise, the proposed MV framework does not seem to be any more vulnerable to noise. Indeed, as long as noise is randomly distributed in some or across all attributes, the chance for an "important" instance to be corrupted is the same as an "unimportant" instance. Therefore, the proposed MV method is practically not sensitive to noisy data and small chunk sizes.

5) Runtime Performance Study: In Fig. 11, we report the system runtime performance, where the x-axis denotes the chunk size, and the y-axis denotes the average system runtime w.r.t. a single data chunk. Because AM strictly follows the GU framework except that its instance selection module is replaced by using a distributional measure, the runtimes of AM and GU are very close to each other with AM slightly more efficient than GU. Not surprisingly, RS has demonstrated itself to be the

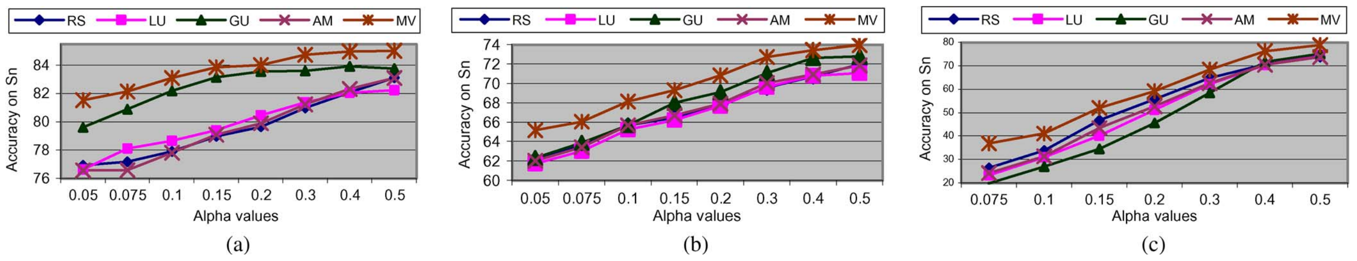

Fig. 9. Classifier ensemble accuracy on data chunk S_n (chunk size 500, $e = 3$, C4.5). (a) Adult. (b) Covtype. (c) Letter.

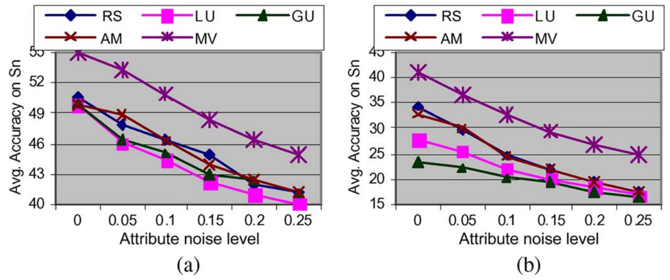

Fig. 10. Experimental result comparisons on noisy stream data. The x-axis denotes the attribute noise level (the probability that an attribute value is changed to another random value) and the y-axis denotes the average active learning accuracy across all chunks (the chunk size is 500 and $\alpha = 0.1$, C4.5). (a) c4–I50k–d10–p5–N1000–t0.1–h0.2. (b) Letter.

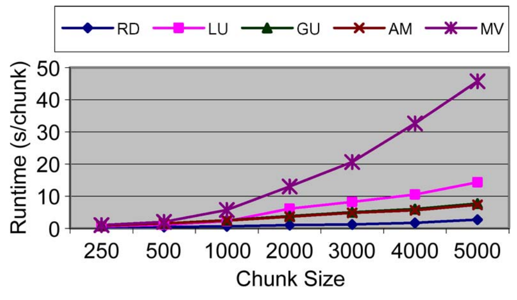

Fig. 11. System runtime with respect to different chunk sizes (c4–I50k–d10–p5–N1000–t0.1–h0.2 stream and $\alpha = 0.5$).

most efficient method due to its simple random selection nature. GU and AM are at the second tier because instance labeling in each chunk involves a recursive labeling and retraining process. Similar to GU and AM, LU also requires a recursive instance selection process plus a number of local training to build classifiers from each chunk. Consequently, LU is less efficient than GU and AM. The proposed MV method is the most time-consuming approach mainly because the calculation of the ensemble variance and the weight updating require additional scanning in each chunk. On average, when the chunk size is 5000 or less, the runtime of MV is about two to four times more expensive than its other peers. The larger the chunk size, the more expensive the MV can be, because the weight updating and instance labeling requires more iterations. Such observations are consistent with the analysis in Section V-C and suggest that MV prefers smaller chunk sizes from the computational efficiency perspective.

VII. Conclusion

In this paper, we have proposed a new research topic on active learning from stream data, where data volumes continuously increase and data concepts dynamically evolve, and the objective is to label a small portion of the data to form a classifier ensemble with minimum error rate in predicting future samples. To address the problem, we studied the connection between a classifier ensemble's variance and its prediction error rates and showed that minimizing a classifier ensemble's variance is equivalent to minimizing its error rates. Based on this conclusion, we derived an optimal weighting method to assign weight values for base classifiers such that they can form an ensemble with minimum error rate. Following the above derivations, we proposed an MV principle for active learning from stream data, where the key is to label instances responsible for a large variance value from the classifier ensemble. Our intuition was that providing class labels for such instances can significantly reduce the variance of the classifier ensemble and therefore minimize its error rates. Experimental results on synthetic and real-world data showed that the dynamic nature of data streams imposes significant challenges to existing active learning algorithms, particularly when dealing with multiclass problems. Simply applying uncertainty sampling globally or locally to the data may not receive good performance in practice. The proposed MV principle and active learning framework address these challenges using a variance measure to guide the instance selection process, followed by the weight optimization to ensure that the instance labeling process can quickly adapt to the drifting concepts in the stream data.

Acknowledgment

The authors would like to thank the anonymous reviewers whose constructive comments and advice have helped improve this paper.

References

[1] C. Aggarwal, *Data Streams: Models and Algorithms*. New York: Springer-Verlag, 2007.

[2] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. KDD*, 2000, pp. 71–80.

[3] H. Wang, W. Fan, P. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. KDD*, 2003, pp. 226–235.

[4] P. Zhang, X. Zhu, and L. Guo, "Mining data streams with labeled and unlabeled training examples," in *Proc. ICDM*, 2009, pp. 627–636.

[5] H. Wang, J. Yin, J. Pei, P. Yu, and J. Yu, "Suppressing model overfitting in mining concept-drifting data streams," in *Proc. KDD*, 2006, pp. 736–741.

[6] Y. Yang, X. Wu, and X. Zhu, "Combining proactive and reactive predictions of data streams," in *Proc. KDD*, 2005, pp. 710–715.

[7] W. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. KDD*, 2001, pp. 377–382.

[8] J. Gao, W. Fan, J. Han, and P. Yu, "A general framework for mining concept-drifting data streams with skewed distributions," in *Proc. SIAM Int. Conf. Data Mining*, 2007, pp. 3–14.

[9] W. Fan, Y. Huang, H. Wang, and P. Yu, "Active mining of data streams," in *Proc. SIAM Int. Conf. Data Mining*, 2004, pp. 457–461.

[10] X. Zhu, P. Zhang, X. Wu, D. He, C. Zhang, and Y. Shi, "Cleansing noisy data streams," in *Proc. 8th IEEE Int. Conf. Data Mining*, 2008, pp. 1139–1144.

[11] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Mach. Learn.*, vol. 15, no. 2, pp. 201–221, May 1994.

[12] X. Zhu, X. Wu, and Q. Chen, "Eliminating class noise in large datasets," in *Proc. ICML*, 2003, pp. 920–927.

[13] H. Seung, M. Opper, and H. Sompolinsky, "Query by committee," in *Proc. COLT*, 1992, pp. 287–294.

[14] Y. Freund, H. Seung, and E. Tishby, "Selective sampling using the query by committee algorithm," *Mach. Learn.*, vol. 28, no. 2/3, pp. 133–168, Aug./Sep. 1997.

[15] S. Hoi, R. Jin, J. Zhu, and M. Lyu, "Batch mode active learning and its application to medical image classification," in *Proc. ICML*, 2006, pp. 417–424.

[16] H. Nguyen and A. Smeulders, "Active learning using pre-clustering," in *Proc. ICML*, 2004, p. 79.

[17] M. Culver, D. Kun, and S. Scott, "Active learning to maximize area under the ROC curve," in *Proc. ICDM*, 2006, pp. 149–158.

[18] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift," in *Proc. ICML*, 2005, pp. 449–456.

[19] P. Utgoff, "Incremental induction of decision trees," *Mach. Learn.*, vol. 4, no. 2, pp. 161–186, Nov. 1989.

[20] K. Tumer and J. Ghosh, "Error correlation and error reduction in ensemble classifier," *Connection Sci.*, vol. 8, no. 3/4, pp. 385–404, Dec. 1996.

[21] K. Tumer and J. Ghosh, "Analysis of decision boundaries in linearly combined neural classifiers," *Pattern Recognit.*, vol. 29, no. 2, pp. 341–348, Feb. 1996.

[22] R. Kohavi and D. Wolpert, "Bias plus variance decomposition for zero-one loss function," in *Proc. ICML*, 1996, pp. 275–283.

[23] X. Zhu and X. Wu, "Class noise vs attribute noise: A quantitative study of their impacts," *Artif. Intell. Rev.*, vol. 22, no. 3/4, pp. 177–210, Nov. 2004.

[24] J. Snyman, *Practical Mathematical Optimization*. New York: Springer-Verlag, 2005.

[25] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. San Mateo, CA: Morgan Kaufmann, 2005.

[26] J. Quinlan, *C4.5: Programs for Machine learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[27] D. Newman, S. Hettich, C. Blake, and C. Merz, UCI Repository of Machine Learning, 1998.

[28] P. Domingos and M. Pazzani, "On the optimality of the simple Bayesian classifier under zero–one loss," *Mach. Learn.*, vol. 29, no. 2/3, pp. 103–130, Nov./Dec. 1997.

[29] L. Breiman, "Bias, variance, and arching classifiers," Statistics Dept., Univ. California at Berkeley, Berkeley, CA, Tech. Rep. #460, 1996.

[30] J. Friedman, "On bias, variance, 0/1-loss, and the curse-of-dimensionality," *Data Mining Knowl. Discover*, vol. 1, no. 1, pp. 55–77, 1996.

[31] E. Kong and T. Dietterich, "Error-correcting output coding corrects bias and variance," in *Proc. 12th ICML*, 1995, pp. 313–321.

[32] D. Moore and G. McCabe, *Introduction to the Practice of Statistics,* 4th ed. San Francisco, CA: Michelle Julet, 2002.

[33] C. Corinna and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[34] J. Kittler, M. Hatef, R. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 3, pp. 226–239, Mar. 1998.

[35] T. Ho, J. Hull, and S. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 1, pp. 66–75, Jan. 1994.

[36] L. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Hoboken, NJ: Wiley, 2004.

[37] J. Kittler and F. Alkoot, "Sum versus vote fusion in multiple classifier systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 1, pp. 110–116, Jan. 2003.

[38] P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi, "A low-granularity classifier for data streams with concept drifts and biased class distribution," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 9, pp. 1202–1213, Sep. 2007.

[39] P. Mitra, C. Murthy, and S. Pal, "A probabilistic active support vector learning algorithm," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 3, pp. 413–418, Mar. 2004.

[40] D. Cohn, Z. Ghahramani, and M. Jordan, "Active learning with statistical models," *Artif. Intell. Res.*, vol. 4, no. 1, pp. 129–145, Jan. 1996.

[41] C. Campbell, N. Cristianini, and A. Smola, "Query learning with large margin classifiers," in *Proc. ICML*, 2000, pp. 111–118.

[42] S. Ji, B. Krishnapuram, and L. Carin, "Variational Bayes for continuous hidden Markov models and its application to active learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 522–532, Apr. 2006.

[43] D. Lewis and J. Catlett, "Heterogeneous uncertainty sampling for supervised learning," in *Proc. ICML*, 1994, pp. 148–156.

[44] N. Abe and H. Mamitsuka, "Query learning strategies using boosting and bagging," in *Proc. 15th ICML*, Madison, WI, 1998, pp. 1–9.

[45] X. Zhu and X. Wu, "Class noise handling for effective cost-sensitive learning by cost-guided iterative classification filtering," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 10, pp. 1435–1440, Oct. 2006.

[46] X. Zhu, P. Zhang, X. Lin, and Y. Shi, "Active learning from data streams," in *Proc. ICDM*, 2007, pp. 757–762.

[47] J. Rodriguez, L. Kuncheva, and C. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1619–1630, Oct. 2006.

[48] P. Melville and R. Mooney, "Diverse ensembles for active learning," in *Proc. 21st ICML*, 2004, p. 74.

[49] H. Mamitsuka and N. Abe, "Active ensemble learning: Application to data mining and bioinformatics," *Syst. Comput. Jpn.*, vol. 38, no. 11, pp. 100–108, 2007.

[50] J. Huang, S. Ertekin, Y. Song, H. Zha, and C. Giles, "Efficient multi-class boosting classification with active learning," in *Proc. SDM*, 2007, pp. 297–308.

[51] N. Pedrajas, C. Osorio, and C. Fyfe, "Nonlinear boosting projections for ensemble construction," *J. Mach. Learn. Res.*, vol. 8, pp. 1–33, 2007.

[52] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[53] Y. Freud and R. Schapire, "Experiments with a new boosting algorithm," in *Proc. ICML*, 1996, pp. 148–156.

[54] W. Hu, W. Hu, N. Xie, and S. Maybank, "Unsupervised active learning based on hierarchical graph-theoretic clustering," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 5, pp. 1147–1161, Oct. 2009.

[55] G. Stemp-Morlock, "Learning more about active learning," *Commun. ACM*, vol. 52, no. 4, pp. 11–13, Apr. 2009.

[56] C. Monteleoni and M. Kaariainen, "Practical online active learning for classification," in *Proc. CVPR Online Learning Classification Workshop*, 2007, pp. 1–8.

[57] B. Settles, "Active learning literature survey," Univ. Wisconsin-Madison, Madison, WI, Computer Science Tech. Rep. 1648, 2009.

[58] J. Berger, *Statistical Decision Theory and Bayesian Analysis,* 2nd ed. New York: Springer-Verlag, 1985.

[59] X. Zhu, X. Wu, and C. Zhang, "Vague one-class learning for data streams," in *Proc. ICDM*, 2009, pp. 657–666.

[60] X. Zhu, *Active learning for concept drifting data streams: Results and source codes*, Florida Atlantic Univ., Boca Raton, FL. [Online]. Available: http://www.cse.fau.edu/~xqzhu/Stream/ActiveLearning/index.html

Xingquan Zhu received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2001.

He was a Postdoctoral Associate with the Department of Computer Science, Purdue University, West Lafayette, IN, from 2001 to 2002, a Research Assistant Professor with the Department of Computer Science, University of Vermont, Burlington, from 2002 to 2006, and a tenure track Assistant Professor with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, from 2006 to 2009. He is currently an Associate Professor with the Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Sydney, Australia. Since 2000, he has published more than 100 referred journal and conference proceeding papers. His research mainly focuses on data mining, machine learning, and multimedia systems.

Dr. Zhu has been an Associate Editor of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING since 2009 and is a Program Committee Cochair for the 9th International Conference on Machine Learning and Applications (ICMLA 2010).

Peng Zhang received the B.S. degree in computer science from Nanchang University, Nanchang, China, in 2003 and the Ph.D. degree in computer science from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2009.

He is currently an Assistant Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include data stream mining, information filtering, and information security.

Xiaodong Lin received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 1997 and the M.S. degree in statistics, the M.S. degree in computer science, and Ph.D. degree in statistics from Purdue University, West Lafayette, IN, in 2000, 2003, and 2003, respectively.

From 2003 to 2004, he was a Postdoctoral Fellow with the National Institute of Statistical Sciences and the Statistical and Applied Mathematical Sciences Institute. He was an Assistant Professor from 2004 to 2008 and an Associate Professor from 2008 to 2009 with the Department of Mathematical Sciences, University of Cincinnati, Cincinnati, OH. He is currently an Associate Professor with the Department of Management Sciences and Information Systems, Rutgers Business School, Rutgers, the State University of New Jersey, Newark. His research interests include statistical data mining, Bayesian statistics, privacy preserving data mining, and bioinformatics.

Dr. Lin is an elected member of the International Statistics Institute.

Yong Shi received the Ph.D. degree in management science from the University of Kansas, USA.

He is the Executive Deputy Director of the Fictitious Economy and Data Science (FEDS) Research Center, Chinese Academy of Sciences, Beijing, China. Since 1999, he has been the Charles W. and Margre H. Durham Distinguished Professor of information technology with the College of Information Science and Technology, Peter Kiewit Institute, University of Nebraska, Omaha, NE. He has published more than 15 books, over 150 papers in various journals, and numerous conferences/proceedings papers. He is the Editor-in-Chief of the *International Journal of Information Technology and Decision Making* and a member of Editorial Board for a number of academic journals. He has consulted or worked on business projects for a number of international companies in data mining and knowledge management. His research interests include business intelligence, data mining, and multiple criteria decision making.

Dr. Shi has received a number of distinguished awards including the Georg Cantor Award of the International Society on Multiple Criteria Decision Making in 2009, the Outstanding Young Scientist Award, the National Natural Science Foundation of China in 2001, and the Speaker of Distinguished Visitors Program (DVP) for 1997–2000 from the IEEE Computer Society.