Active Learning from Stream Data Using Optimal Weight Classifier Ensemble

Xingquan Zhu[1], Peng Zhang[2], Xiaodong Lin[3], and Yong Shi[2]

[1]Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA

[2]FEDS Center, Graduate University, Chinese Academy of Sciences, Beijing, 100080, China

[3]Department of Management Science & Information Systems, Rutgers, the State University of New Jersey, NJ 07102, USA

xqzhu@cse.fau.edu; {pengzhang,yshi}@guca.ac.cn; lin@business.rutgers.edu

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. In order to tackle the challenges raised by data streams' dynamic nature: increasing data volumes and evolving decision concepts, we propose a classifier-ensemble based active learning framework which selectively labels instances from data streams to build an ensemble classifier. We argue that a classifier ensemble's variance directly corresponds to its error rates and the efforts of reducing a classifier ensemble's variance is equivalent to improving its prediction accuracy. Because of this, an active learning algorithm should label instances towards the minimization of the variance of the underlying classifier ensemble. Therefore, we introduce a *Minimum-Variance* principle to guide instance labeling process for data streams. In addition, we also derive an optimal-weight calculation method to ensure a minimum error rate of the classifier ensemble built from labeled stream data. The proposed minimum-variance principle and the optimal weighting module are then combined to build an active learning framework for data streams. Experimental results on synthetic and real-world data will 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 data streams for quick response and accurate decision making [1]. One of the recent challenges facing data mining is to digest massive volumes of data produced from such data streams [1-10].

In the domain of classification, in order to generate a predictive model it is essential to label a set of examples for training purposes. 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]. 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 the active learning is to achieve a high prediction accuracy classifier 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 [13-14], or query-by-margin [41, 43] principles takes instances, with which the current learners have the highest uncertainty, as the most 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 improve the

model training. A large body of work exists on active learning for static datasets [13-17, 39-44], and all these endeavors aim at building one single optimal model from the labeled data. None of them, however, fits in the setting of data streams, where the continuous data volumes and the drifting of the concepts raise significant challenges and complications.

A. Concept Drifting in Stream Data

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

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

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

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

The complication of the data stream, in comparison with a static dataset, 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 variance 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 be further decomposed into the following three categories: (1) priori probability drifting: the concept drifting is mainly triggered by the class priori probability $P(c_i)$ only, (2) conditional probability drifting: the concept drifting is mainly trigged by the class conditional only, 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 Figure 1.

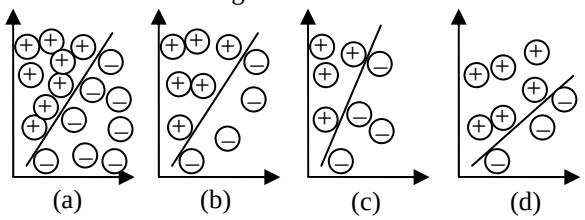


Figure 1: A conceptual view of the concept drifting in data stream. (a) denotes the genuine decision boundary assume all samples of the stream were observed, (b) priori probability drifting scenario (triggered by changes in $P(c_i)$ only), (c) conditional probability drifting scenario (triggered by changes in $P(x|c_i)$ only, where class priori probability $P(c_i)$ remains the same as (a)), and (d) conjunct probability drifting (triggered by changes in $P(c_i)$ and $P(x|c_i)$).

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 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 unseen examples can be maximized.

For static data sets whose whole candidate pools can be observed and their genuine decision boundaries are invariant, the active learning is supposed to answer "which samples should be labeled"? For data streams with continuously increasing volumes and variant decision boundaries, the 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, it makes sense to save the labeling cost for future samples which 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 the concept drifting in data streams is mostly triggered by complicated factors (as discussed in the above subsection) so we may not be able to accurately estimate 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 concepts in the data are

constantly evolving, so the 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 samples, for labeling. As we will shortly demonstrate in Section VI, such an approach will produce better results than relying on the detecting of the concept drifting for active learning.

C. Challenges of Active Learning from Stream Data

Presumably the challenges 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 dealing with static datasets 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 its increasing data volumes, building one single model from all 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 labeled 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 adding help [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 from stream data? Or what is the final goal of carrying out active learning for stream data?
2. What are the objective function and criteria for instance labeling from stream data?
3. How to tackle a concept drifting data stream with massive data volumes for effective active learning?

We present here, in this paper, our recent research efforts in resolving the above concerns. In short, we propose a classifier ensemble [36] based framework, where a set of base classifiers and associated weight values help 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 (1st 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 (2nd concern). The employment of the classifier ensemble ensures that our method can not only handle data with massive volumes through chunk-based learning, but is also able to adapt to the drifting concepts in the streams though the adjustment of the ensemble weight values (the 3rd concern).

The remainder of the paper is structured as follows. Section II briefly reviews the related work. Section III presents a motivating example and simple solutions. Section IV introduces 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, key symbols used in this paper are listed in Table 1.

TABLE 1: Key symbols used in the 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
$\eta_{c_i}^n$	A random variable accounts for the variance of the classifier C_n *w.r.t.* class c_i
$\sigma^2_{\eta_{c_i}^n}$	The variance of $\eta_{c_i}^n$
β_{c_i}	The bias of the classifier with respect to class c_i
E	A shorthand 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 constituting classifier is called a base classifier
w^n	The weight value of the classifier C_n of the classifier ensemble
$\eta_{c_i}^E$	A random variable accounts for the variance of the classifier ensemble E *w.r.t.* class c_i,
$\sigma^2_{\eta_{c_i}^E}$	The variance of $\eta_{c_i}^E$

II. RELATED WORK

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

Classifier ensemble is an established research area. Numerous methods [34-37, 52-53] exist for improving the ensemble accuracy. 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 a number of researches [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 further improve classifier ensemble learning, such as multi-class boosting classification [49] or active ensemble learning [50]. Although all these methods have been using ensemble framework for active learning or vice versa, they are not primarily designed for stream data environments.

For data streams with continuous volumes, classifier ensemble has shown to be an effective solution to tackle the data volumes and concept drifting challenges [3-4, 7-8]. Street [7] proposed a SEA algorithm, which combines all the decision tree models using majority-voting. Kolter [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], we have employed classifier ensemble for stream data cleansing [10] and combining labeled and unlabeled training samples for stream data mining [4]. In summary, although classifier ensemble has been popularly used in 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 active mining framework [9], which labels samples only if it is necessary. In short, active mining uses a decision tree (trained from the currently labeled data) to compare the distributions of the incoming samples and the data collected at hand on the tree/branch (without observing the class labels). If two sets of samples are subject to different distributions, a labeling process is triggered to randomly select some incoming samples for labeling. Although our research shares the same goal as the active mining: minimizing the labeling cost for data streams, the differences between them are, however, fundamental. (1) active mining 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) active mining 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; and (2) active mining is only applicable for decision trees, because the detection of the distributional changes relies on the given decision trees. In comparison, our work can be applied to any learners including decision trees. Our experimental comparisons in Section VI will demonstrate that active mining only marginally outperforms random labeling based approaches in solving our problem.

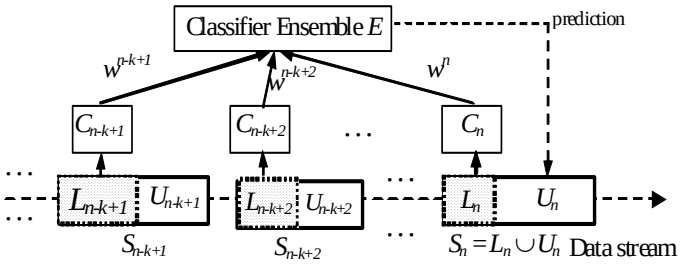


Figure 2: A general framework for active learning from stream data

III. PROBLEM DEFINITION AND SIMPLE SOLUTIONS

A. A Motivating Example and Problem Definition

Consider an online intrusion detection center, which 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, and the network security experts can only investigate 5% of them. Now the problem becomes which 5% of 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]). In order to handle massive volumes of stream data, a common solution is to partition the data into chunks, as shown in Figure 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 built and used to form a classifier ensemble [34-36] to identify intrusions from newly arrived data. The framework in Figure 2 can be applied to a variety of applications as long as instance labeling for data streams is of concern. The utilization of classifier ensemble ensures that our framework can effectively handle massive volumes of stream data, without relying on any complex incremental learning techniques [2, 19].

Based on the framework in Figure 2, we assume that stream data are partitioned chunk by chunk 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

B.1 Random Sampling (RS)

Arguably, the simplest approach to solve our problem is random sampling, where instances in S_n are randomly sampled and labeled. Although simple, it turns out that RS works surprisingly well in practice (as we will shortly see in Section VI). The niche of random sampling 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, consequently, the labeled instances no longer reveal genuine class distributions in the data chunk. This problem is less severe for a static dataset where candidate pool is fixed and active learning algorithms are able to survey all candidates. Random sampling avoids this problem by randomly labeling instances. As a result, it can produce a training set with the most similar class distributions to the current data chunk, although instances it labeled might not be as "informative" as those carefully selected.

B.2 Local Uncertainty Sampling (LU)

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 dataset. Existing active learning algorithm can then be applied to S_n without considering any other data chunks. Because instance labeling is carried out independently in each data chunk, the weakness of LU is obvious: contributions of the labeled instances with respect to the global classifier ensemble are not clear, although each data chunk might indeed be able to label the most important instances locally.

B.3 Global Uncertainty Sampling (GU)

Global uncertainty sampling based active learning will use historical classifiers, along with the one from S_n, to form a classifier 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 forms a committee to assess instances in S_n and label the ones with the largest uncertainty. The labeling 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 improve classifier committee's capability.

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 Query by Committee (QBC) [13-14] based approaches. Unfortunately, our experimental results in Section VI indicate that GU's performance is still unsatisfactory and often loses to random sampling. One possible reason is that different from the traditional QBC where committee members are learnt from samples drawn from the same distributions, the committee classifiers in data streams are learnt 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.

B.4 Active Mining Based Sampling (AM)

AM sampling is motivated by the active mining framework proposed by Fan et al. [9], and the intuition is to select 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 the arrival of a data chunk S_n for labeling, AM randomly labels a tiny set of instances from S_n and build a decision tree dt_n. After that, AM calculates distributions of unlabeled samples in S_n with respect to each of the k trees (dt_{n-k+1},…, 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 instances in S_n such that the 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 classifier variance for a single learner and then extend our analysis to classifier ensemble. We argue that minimizing classifier ensemble variance is equivalent to minimizing its error rate. Following this conclusion, we derive an optimal-weight calculation method to assign optimal 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 the 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 Eq. (3).

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

If we consider that the added error of the classifier is mainly derived from two sources: classifier bias and variance [30-33], the added error $\varepsilon_i(x)$ in Eq. (3) can be decomposed into two terms: 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 accounts for the variance of the classifier (with respect to class c_i). This gives Eq. (4) [3, 20-22].

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

In Eq. (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, …, C_n) from T, then all classifiers will suffer from 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 Figure 3. Given a learning set T and a specific learning algorithm, say C4.5 [26] or Naïve Bayes [28], one can randomly sample T (with replacement) and generate a number of training set T_1, T_2, … T_n, each of which has the same number of instance as T. After that, one classifier is trained from each single dataset, which results in n classifiers C_1, C_2, …, C_n in total. At the last step, the n classifiers are used to predict a specific instance x, with the class probability value predicted from each classifier (with respect to class c_i) denoted by a dot in Figure 3 (for ease of understanding, we assume that each prediction is a two-dimensional point). In Figure 3(a), the center denotes the Bayes posterior probability $p(c_i|x)$, whereas each prediction from classifier C_1, C_2, …, C_n is an offset value from the center. Because C_1, C_2, …, C_n are trained by using the same learning algorithm but different versions of the same dataset, 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.

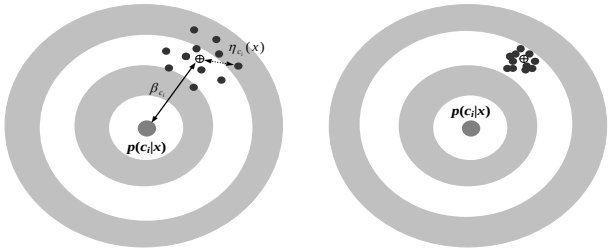
Depending on the learning algorithms and the training set used to train the classifiers, one can expect that the overall predictions of C_1, C_2, …, C_n may vary from large bias small variance (Figure 3(b)), small bias large variance

(Figure 3(c)), to small bias small variance (Figure 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.

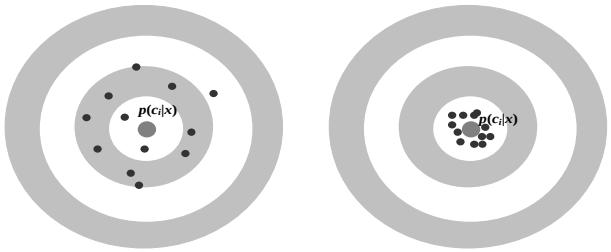
Assume we are using the same learning algorithm in our analysis, without loss of generality, we can ignore the bias term [3, 20]. Consequently, the learner's probability in classifying x into class c_i becomes

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

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 Eq.(5), which approximates $p(c_i|x)$, is also shifted from the optimum Bayes decision boundary, as shown in Figure 4.



(a) Bias and variance decomposition (b) Large bias small variance



(c) Small bias large variance (c) Small bias small variance

Figure 3: Classifier bias and variance illustration in shooting arrows at a target. The center denote the posterior probability $p(c_i|x)$, 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, …, 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]).

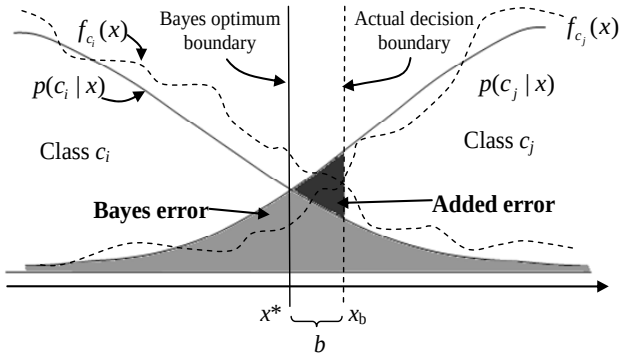


Figure 4: Decision boundaries and error regions associated with approximating the a posteriori probabilities (picture revised from [21])

In Figure 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 Eq. (6), where $A(b)$ is the area of the darkly shaded region, and f_b is the density function for b.

$$Err_{add} = \int_{-\infty}^{\infty} A(b)\, f_b(b)\, db \ . \tag{6}$$

Tumer et al [20] showed that this quantity can be expressed as

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

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

Eq. (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.

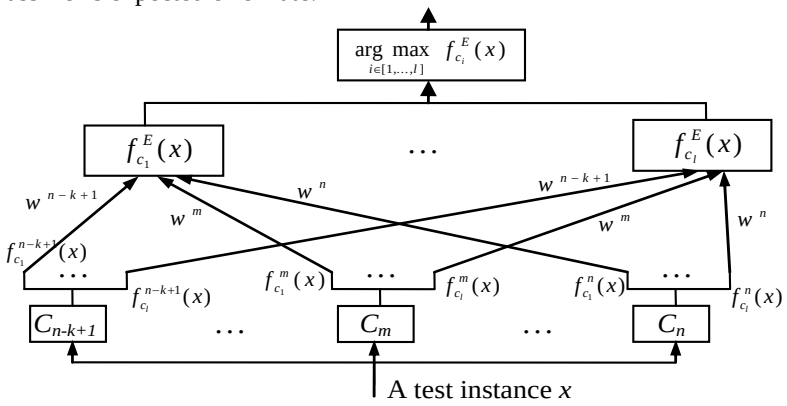


Figure 5: A classifier ensemble model consisting of k base classifiers C_{n-k+1}, C_{n-k+2},.., C_m, .., C_n. Each base classifier C_m has an associated weight value w^m. The class probability of a test instance on class c_i is the sum of the weighted class probabilities over all base classifiers. The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

B. Classifier Ensemble Variance

Assume the given of a classifier ensemble E with k base classifiers (in this paper, each member of a classifier 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 its base classifiers. Here we employ a weighted ensemble framework, as shown in Figure 5, where each classifier C_n has a weight values w^m. Under the framework in Figure 5, the probability of E in classifying x into class c_i is given by Eq. (8), where $f^m_{c_i}(x)$ denotes the probability of base classifier C_m in classifying x into class c_i.

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

This probability can be expressed as

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

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

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

Therefore, the variance of $\eta^E_{c_i}(x)$ is given by Eq. (11)

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

which can be rewritten as Eq. (12)

$$\sigma^2_{\eta^E_{c_i}} = \frac{\sum_{m=n-k+1}^{n} (w^m)^2 \sigma^2_{\eta^m_{c_i}}}{\left(\sum_{m=n-k+1}^{n} w^m\right)^2} + \frac{\sum_{m=n-k+1}^{n}\sum_{g=n-k+1;\, g\neq m}^{n} w^m w^g \operatorname{cov}(\eta^m_{c_i}, \eta^g_{c_i})}{\left(\sum_{m=n-k+1}^{n} w^m\right)^2} \tag{12}$$

Assume 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 Eq. (12) equals zero, and the variance of $\eta^E_{c_i}(x)$ becomes Eq. (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 Eq. (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, $y^x_{c_i}$ is equal to 1, otherwise, it is equal to 0. Consequently, the variance of the classifier C_m over all class $c_1, c_2,\ldots,c_l$ is given by Eq. (15).

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

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

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

The total variance of the ensemble E is then given by Eq. (16), which is called the *classifier ensemble variance* in this paper.

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

According to Tumer et al's conclusion in Eq. (7), a classifier's expected added error is proportional to its variance. Consequently, a classifier ensemble's expected error can be denoted by Eq. (17)

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

Eq. (17) states that in order to minimize a classifier ensemble error rate, we can minimize its variance instead, this can be achieved through the adjustment of the weight value w^m, m=n-k+1,.., n, associated to each of E's base classifier C_m.

C. Optimal-Weight Classifier Ensemble

Section III.B concludes that in order to build a classifier ensemble E with minimum error rate, we need to find the weight values w^m, m=n-k+1,…,n such that the classifier ensemble variance defined by Eq. (14) can reach the minimum. This is equivalent to the problem given by Eq. (18).

$$\arg\min_{w^m} \left(\sigma^2_{\eta^E_c}\right) \tag{18}$$

To find solution for the problem given in Eq. (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

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

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

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

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

$$\text{Min} \quad \sigma^2_{\eta^E} = \sum_{m=n-k+1}^{n} \{(p^m)^2 \cdot \sigma^2_{\eta_c^m}\} \qquad (21)$$

$$\text{s.t.}: \quad \sum_{m=n-k+1}^{n} p^m = 1$$

This mathematical programming model can be easily solved explicitly by Lagrange multiplier, where the Lagrange function is given in Eq. (22)

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

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

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

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

$$p^i h^i = p^j h^j, \quad \forall i \neq j \qquad (24)$$

According to Eq. (22) and the constraint $\sum_{m=n-k+1}^{n} p^m = 1$ in Eq. (21), we can easily find solution to p^m, as shown in Eq. (25).

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

Because the value of w^m is proportional to p^m, so we can directly use the p^m values calculated from Eq. (25) as ensemble weight value w^m, $m=n-k+1, ..,n$, 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: MINIMUM-VARIANCE ACTIVE LEARNING FROM STREAM DATA

In stream data environments, because labeling all instances in each data chunk is out of 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 help build a classifier ensemble with maximum accuracy in predicting future unseen samples*?

To explicitly answer the above question, we shall recall that Tumer et. al. [20] has 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. Assume the weight values of the base models 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 or not. More specifically, if an instance contributes a small variance value, 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, it means that the base classifiers are inconsistent in classifying the instance, possibly because that the classifiers do not have sufficient knowledge for prediction (e.g., due to the

shifting of the concepts). So 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* principle for active learning in data stream where the intuition is to label instances which 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. Figure 6 shows the proposed framework following this principle, 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 (EM) like logics. On one hand, given a set of weight values for a classifier ensemble, our algorithm intends to find instances worth labeling by using the ensemble framework and its weight values (this is the interest of the active learning). On the other hand, given a number of labeled samples, our algorithm uses them to help determine the optimal weight values for the ensemble classifier, such that the overall prediction accuracy can be maximized (*i.e.*, the interest of weight updating for ensemble learning).

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) α, the percentage of instances should be labeled from S_n; (2) e, # of epochs in labeling α percentage of instances from S_n;
Objective: Label α 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. $L_n \leftarrow \varnothing$; $U_n \leftarrow S_n$
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 4.
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 $\hat{I}_x$ means I_x with a predicted class label.
   c. Calculate each classifier C_m's variance on Λ_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 α 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

Figure 6: MV: Minimum variance active learning from stream data

Assume that the algorithm has just finished data chunk S_{n-1} with the most recent k-1 classifiers denoted by C_{n-k+1}, C_{n-k}, .., 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 denoted by C_{n-k+1}, C_{n-k}, .., C_{n-1} can form an accurate classifier ensemble. Denoting L_n and U_n the labeled and unlabeled instance subsets of S_n, since all data in S_n have no labels in the beginning, we set $L_n \leftarrow \varnothing$ and $U_n \leftarrow S_n$ on step (1) in Figure 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 which builds a classifier C_n from L_n. The k classifiers C_{n-k+1}, C_{n-k}, .., $C_{n-1,}$ C_n form an ensemble E, where the initial weight value w^m of each base classifier C_m, $m=n-k+1$, .., 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 after the labeling process is done the classifier ensemble E has the minimal error rate on the remaining instances in S_n. In our algorithm, the instance selection process is explicitly guided by the Minimum-Variance principle which checks each unlabeled instances in U_n and selects the ones with the largest ensemble variance value for labeling.

According to Eqs. (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 $\sigma^2_{\eta_c^m}$ 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 Eq. (16) to calculate ensemble variance on Λ_x, this value is taken 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. The labeled instance I_x is 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 on 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 Eq.(25) and calculate the new weight values w^m for each base classifier.

Following the weight optimization process, the algorithm checks the following three conditions consecutively to ensure an 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 Figure 6 can effectively address concept drifting challenges in data streams. This can be justified from both local and global perspectives. Locally, since classifier C_n is learnt from the most recent data chunk S_n, and C_n has smaller variance on L_n compared to other k-1 classifiers built from the previous data chunks. The solutions to Eq. (25) will show that a classifier with a smaller variance on L_n will receive a larger weight value and thus play a more important role in the classifier ensemble. If concepts in data chunk S_n are significantly different from other data chunks, the large weight value associated to 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 address concept drifting in S_n adaptively. From the global point of view, a classifier ensemble consists of a set of base classifiers, each of which containing a weight value, as concepts evolve over time, we only need to adjust the weight value of each base classifiers. According to Eq. (25), the weight value of the base model is inversely proportional to its variance on chunk S_n. Assume the concept in chunks S_n is similar to the chunk S_{n-k+1} and distinct from the rest of chunks in the ensemble (*i.e.*, S_{n-k+2},…,S_{n-1}), according to Eq. (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 Figure 6 updates weight values once for each single labeled instance. To speed up the process, one can label multiple, say τ, instances in one time (select the ones with the largest variance values), and use all 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 the number of instances in S_n is 10,000, and the user has specified α=0.1 and $e=5$. It means that we should label 10% of instances in five epochs, and classifier C_n is retrained from L_n after the labeling of every 200 instances (one epoch). If the user specifies $\tau=20$, the algorithm will update weight values of the classifier ensemble once in every 20 instances. In practice, the value of τ can be selected such that we only update weights for a fixed number of times in each epoch to speed up the process.

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, give 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 is of 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 containing N instances. The active learning process is supposed to label α percent of instances from a data chunk S_n in e epochs, and (3) The active learning process labels one instance each time and the weight values w^m, $m=n-k+1,..., n$, 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 assumptions listed above, 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). Meanwhile, the number of examples for each training is given by 0[1], $N\cdot\alpha/e$, $2\times N\cdot\alpha/e$, …, $(e-1)\times N\cdot\alpha/e$, respectively. So the total time complexity for model training is given by Eq. (26).

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

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, 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, the active learning process needs to calculate the variance of all base classifiers on the labeled subset (Eqs. (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 is continuously growing, according to Eq. (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, assume the size of Λ_x continuously grows from 0 to $N\cdot\alpha$, the total time complexity for active learning is $2\times O(l\cdot k\cdot N)= O(l\cdot k\cdot N)$, which includes the variance calculation $O(N)$ for each classifier (k classifiers in total) and each class (l classes in total), and 2 means that we have to calculate classifier variance twice (before and after the labeling).

For weight updating, the calculation of p^m, $m=n-k+1,\ldots, n$, in Eq.(25) takes $2\times O(k)$ for all k classifiers (it takes $O(k)$ to calculate the summation in Eq. (25) and $O(k)$ to calculate the p^m values of all classifiers). So in total, the time complexity in each repetition for weight updating is $2\times O(k)$. Because the whole weight updating needs to repeat $N\cdot\alpha$ 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 Eq. (27).

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

The above analysis is based on one single data chunk S_n, assume a data stream has M data chunks in total, the total time complexity $T(n)$ is given by Eq. (28), where e is the number of times retraining the classifier, and l and k denote the number of classes and number of base classifiers of the classifier ensemble.

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

To further simplify Eq. (28), we can consider practical parameter settings, where M and N are usually quite large, whereas e, l, k are usually small (α is a numerical value between 0 and 1). Consequently, we can we can safely assume that $l\cdot(k+\alpha)\le N$ and $\alpha^2\cdot e\le 1$, which gives Eq. (29).

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

Eq. (29) shows that the total time complexity is bounded by two important factors: (1) the number of instances in

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

each data chunk N; and (2) the number of data chunks M. Give one data stream, because model training in each data chunk is nonlinear *w.r.t.* the data chunk size, we may prefer a relative small chunk size from the computational efficiency perspective.

VI. EXPERIMENTAL COMPARISONS

A. Experimental Settings

A.1 General Setting

In order to compare different active learning methods and assess the quality of the instances labeled by them, we train a classifier ensemble E for each of them by using the same framework in Figure 2. Therefore, if one classifier ensemble outperforms its other peers, we can conclude that this is because 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 comparison purposes, we have to make sure that all methods are compared based on the same test sets. Therefore, our comparisons are made on two types of test sets: (1) the average prediction accuracies on all instances in data chunk S_n over the whole data stream; and (2) the average prediction accuracies on a separate test set T, which remains unchanged for each data stream.

A.2 Data Streams

Synthetic data: In order to simulate data streams on which we can fully control the concept drifting magnitude and speed, we employ a hyper-plane based synthetic data stream generator which is popularly used in stream data mining research [2, 7-9]. The hyper-plane of the data generation is controlled by a non-linear function defined by Eq.(30) (we intentionally use a non-linear hyper-plane to challenge the active learning methods). Given an instance x, its $f(x)$ value defined by Eq.(30) determines its class label. Assume an $f(x)$ value larger than a threshold θ indicates that x belongs to class A, otherwise, x belongs to class B, then changing the values of a_i, i=1,..,d, and threshold θ may make an instance x have different posteriori probability $p(c_i|x)$ with respect to a particular class c_i. In an extreme situation, it may make two identical instances have different class labels. By doing so, we are introducing dynamic and variant decision boundaries into the data to simulate concept drifting in data streams.

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

In Eq. (30), d is the total dimensions of the input data x. Each dimension x_i, i=1,..,d, is a value randomly generated in the range of [0, 1]. A weight value a_i, i=1, .., d, is associated to each input dimension, and the value of a_i is initialized randomly in the range of [0, 1] at the beginning. In data generation process, we will gradually change the value of a_i to simulate concept drifting. The general concept drifting is controlled through the following three parameters [2, 7-8]: (1) t, controlling the magnitude of the concept drifting (in every N instances): (2) p, controlling the number of attributes whose weights are 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 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-I100*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 denotes a two class data stream with 100k instances, each containing 10 dimensions. The concept drifting involves 5 attributes, and their weights change with a magnitude of 0.1 in every 1000 instances and weight inverts the direction with 20% of chance.

Real-world Data: we select three relatively large datasets from UCI data repository [27] and treat them as data streams for active learning. The datasets we selected are Adult, Covtype, and Letter (as listed in Table 2). To help interested readers capture the learning complexity of these datasets, Table 2 also lists the 10-fold cross-validation accuracies of the classifiers built from all instances (based on C4.5 and Naïve Bayes classifiers).

Adult: U.S. Census data (1994) which predicts whether a person's income exceeds $50K/year or not. The 15 attributes include age, workclass, education, martial-status, occupation, native-country, race and other features.

Covtype: Predicting forest cover type (7 types) by using cartographic information, such as elevation, slope, horizontal/vertical distance to hydrology, and soil type. Each observation is a 30×30 meter cell, and its actual forest cover type was determined from US Forest Services (USFS) Region 2 Resource Information System (RIS) data.

Letter: Identifying each of an input image (a box) as one of the 26 capital letters in the English alphabet. Each imager is described by 16 features such as width/height of box and horizontal/vertical position of box, and one class label (A to Z)

Among these three datasets, Adult and Covtype are considered dense datasets, which means that a small portion of examples can learn genuine concepts quite well. The class distribution in the Covtype dataset is severely biased, where two classes (*i.e.*, two cover types) dominant the whole dataset and cover over 85% of the instances), whereas the minority class only covers 0.5% of the samples. Letter is a sparse dataset and 26 classes of examples are evenly distributed, and a small portion of examples are insufficient to learn genuine concepts underlying the data.

All methods are implemented in Java platform. The learning algorithms employed include C4.5 [26] and Naïve Bayes [28] which are directly imported from the WEKA data mining tool [25].

A.3 Benchmark Methods

For comparison purposes, we implemented four methods introduced in Section III, including random sampling (RS), local uncertainty (LU), global uncertainty (GU) sampling, and active mining (AM). In our implementation, AM exactly follows the GU framework except that the instance selection procedure is replaced by using the active mining [9] module to select instances with the most significant distributional differences for labeling. The proposed minimum-variance based active learning method is denoted by MV. For fairness of the comparisons, all classifier ensembles use exactly the same architecture as shown in Figure 1. 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 setting 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 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.

Table 2: 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

B. Experimental Results

B.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 3.1) three-class (Table 3.2) and four-class (Table 3.3), and report the results in Table 3. The accuracies in the tables denote an ensemble classifier's average accuracy in predicting instances in the current data chunk S_n, where the chunk size varies from 250 to 2000. We fix α value to 0.1 and set value k to 10, which means that 10% of instances are labeled for each data chunk, and only the most recent 10 classifiers are used to form a classifier ensemble.

The results from Table 3 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 show in Eq. (29), the time complexity of our algorithm nonlinearly increases *w.r.t.* the number of instances in each chunk.

For any particular method, Table 3 indicates that its results on multi-class data streams are significantly worse than a binary class data stream, although all data streams are generated from the same type of hyper-planes (defined by Eq. (30)). This shows that active learning from a multi-class 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 local uncertainty based method (LU) is not an option for active learning from data streams, and its performance is constantly worse than RS regardless of whether the classification problems are binary or multiple classes. Although GU outperforms random sampling (RS) quite often, for multi-class data streams (e.g. four classes) its performance is 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 3.2). As we have analyzed in Section III, random sampling 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 data chunk.

For active mining (AM) based sampling method, we observed that its performance is worse or marginally better than random selection and worse than GU most of times. 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 is two-fold. (1) active mining relies on sample distributions to select "important" instances for labeling. Notice that sample distributions do not necessarily have a direct connection to indicate whether a sample is "important" for labeling or not, even if the decision tree does accurately capture the data distributions. Consider a two classes sample set with genuine decision boundary denoted by x=0 vertical line (*i.e.*, instances with $x \leq 0$ are classifiers 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), and 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 S2={x_5=-1, x_6=1}, their distributions on the tree are (c_1:1.0, c_2:0.0) and (c_1:0.5 c_2:0.5) respectively. Accordingly to the active mining's 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 information to enhance the current decision boundary (x'=-0.5) towards 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 differences are worth labeling, using decision tree may not be able to capture the genuine distributions of the underlying data. In all results we observed in our experiments, AM does not show any significant performance gain than RD. In addition, when the number of training samples is small, the decision tree tends to be a single node tree (with no leaf). As a result, the AM algorithm is deteriorated as the RD algorithm.

The results in Table 3 only show different methods' average accuracy over all data chunks. In order to compare different methods at individual data chunk level, we record each method's accuracy on each single data chunk and report the results (10 times average) in Figures 7 (a) to 7(c), where the x-axis represents data chunk ID in its temporal order and the 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 Figures 8(a) to 8(c). The significant accuracy increase from data chunks 1 to 11 in Figures 8(a) to 7(c) is caused by the increasing number of base classifiers at the beginning of the data streams. The results in Figures 7(a) to 7(c) indicate that the accuracies of the data chunks fluctuate frequently and the fluctuations are quite significant for some data streams (e.g., Figure 7(b)). 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 the classification accuracy. Of course, the actual accuracy not only relies on the class distributions, but also on the complexity of the decision surfaces. The results here indicate that changing class distributions is a challenge for active learning from data streams, especially for a multi-class streams.

The results from Figure 7 show that MV consistently outperforms other methods across all data chunks. This observation asserts that for concept drifting data streams with constant changing prior class distributions and continuous evolving decision surfaces, MV can adaptively label instances from data chunks to build a superior classifier ensemble. The advantage of MV can be observed across different types of data streams (binary-class and multi-class), as well as data chunks across the whole data stream.

In Table 4, we compare different methods by using Naïve Bayes (NB) as our learning algorithm (using the same parameter setting as what we did with C4.5). The results, again, confirm that MV receives the best performance over all data streams. This is very encouraging, as it has been reported that class probabilities estimated by NB are not accurate, although it can often produce good results *w.r.t.* 0/1 loss (*i.e.*, classification accuracy) [28]. Our results show that even when base classifiers are incapable of estimating each instance's class probabilities accurately, our approach may still achieve good performance. This is because MV relies on the variance of the base classifiers' probability estimation, but not the absolute probability values.

Table 3: Average classification accuracy on different data streams (C4.5)

Table 3.1: Accuracy on c2-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (α=0.1)

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

Table 3.2: Accuracy on c3-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (α=0.1)

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

Table 3.3: Accuracy on c4-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (α=0.1)

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

Table 4: Average accuracies on different data streams (NaiveBayes)

Table 4.1: Accuracy on c2-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (α=0.1, *e*=3)

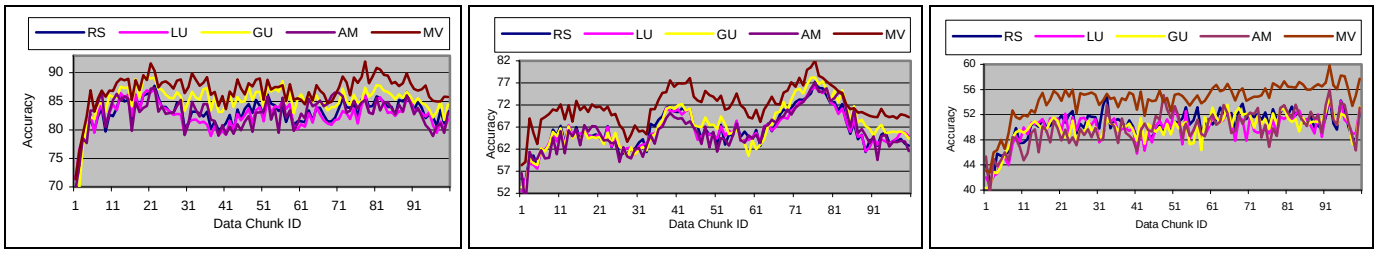
Chunk Size	***RS***	***LU***	***GU***	***MV***
250	78.22±2.59	75.66±2.41	78.87±2.41	**82.21±2.10**
500	78.71±1.82	78.02±1.69	82.31±1.85	**83.47±1.67**
750	79.05±1.71	78.26±1.76	82.84±1.75	**83.89±1.68**
1000	78.94±1.91	78.02±2.01	82.96±1.64	**84.77±1.19**
2000	78.71±2.87	76.98±2.76	83.32±1.87	**84.12±1.44**

Table 4.2: Accuracy on c3-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (α=0.1, *e*=3)

Chunk Size	***RS***	***LU***	***GU***	***MV***
250	59.88±6.52	58.33±6.43	59.20±7.40	**65.52±5.21**
500	63.40±5.68	61.92±6.11	61.76±8.18	**67.11±5.26**
750	63.37±5.91	62.31±5.90	62.03±8.22	**65.75±5.08**
1000	63.10±6.54	61.33±6.61	62.01±7.91	**65.23±5.12**
2000	63.25±8.16	60.77±8.04	63.60±6.96	**65.07±6.87**

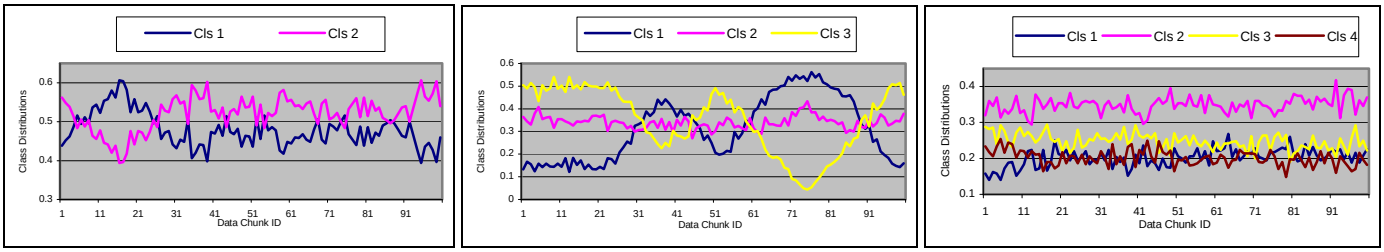
Table 4.3: Accuracy on c4-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (α=0.1, *e*=3)

Chunk Size	***RS***	***LU***	***GU***	***MV***
250	44.80±2.84	43.27±2.85	47.7±3.04	**51.24±1.49**
500	49.71±2.58	48.32±2.44	50.90±2.43	**53.45±2.14**
750	51.47±2.33	49.57±2.45	51.75±2.14	**53.62±2.02**
1000	51.92±2.32	50.57±2.32	50.98±2.07	**54.03±1.58**
2000	**53.32±1.99**	51.33±2.08	51.40±1.78	53.06±1.83



(a) c2-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (b) c3-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (c) c4-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2

Figure 7: Classifier ensemble accuracy across different data chunks: (a) two-class; (b) three-class; (c) four-class data streams (α=0.1, *e*=3, data chunk size: 500)



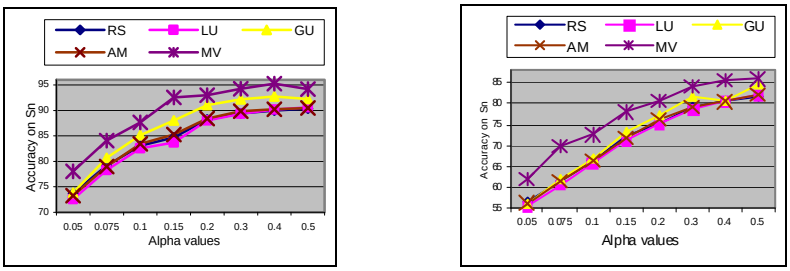
(a) c2-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (b) c3-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (c) c4-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2

Figure 8: Class distributions across different data chunks: (a) two-class; (b) three-class; (c) four-class data streams (α=0.1, *e*=3, data chunk size: 500)

B.2 Active Learning with Different α Values

In Fig. 9, we compare all four methods for different α values. Not surprisingly, when the α value increases, all methods gain better prediction accuracy. This is because the increasing number of labeled instances help build strong base classifiers. Overall, MV and GU achieve the best performance, and LU performs inferior to RS in majority of cases. This, again, asserts that applying traditional uncertainty sampling locally to each single data chunk is inappropriate for data streams, where dynamically changing class distributions and evolving concepts require an active learning algorithm to take these issues into consideration for effective instance labeling. Both MV and GU label instances for the benefits 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 while MV prefers to label instances contradict predicted by base classifiers.

As we discussed in Section III, GU extends Query by Committee to data streams by taking classifiers learnt from different data chunks as committee members. In the original QBC, the committee members are learnt 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 learnt from different portion of stream data, and the concept drifting in the data chunks also render classifiers significantly different from each other. As a result, weighted average uncertainty over all committee members no longer reveals the genuine uncertainty of the classifier ensemble formed by them. The results in Figure 8 support our hypothesis very well. As we can see, MV constantly outperforms GU across all α values. For multi-class data streams, the performance of GU is unsatisfactory and is inferior to random sampling sometime. This becomes extremely clear in the following section, where GU's performance on a real-world 26-class data stream is significantly worse than RS and MV, especially when a small portion of instances are labeled



(a) c2-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (b) c3-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2

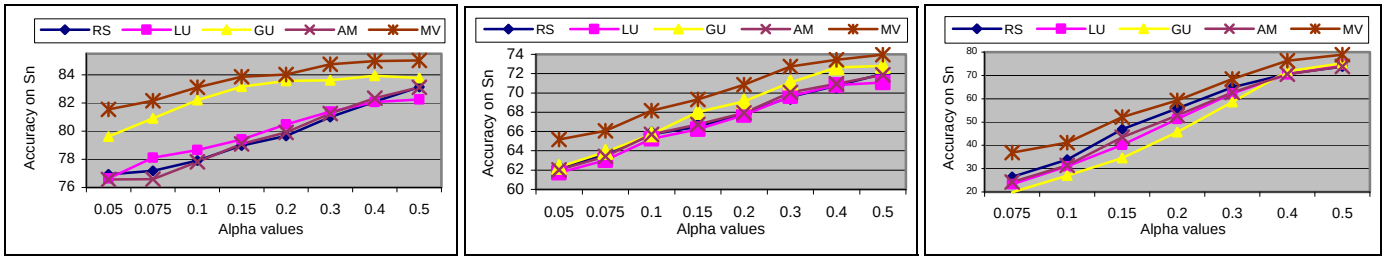
Figure 9: Classifier ensemble accuracy with respect to different α values: (a) two-class; (b) three-class (chunk size 500, *e*=3, C4.5)

B.3 Active Learning from Real-World Data

In Figures 10 and 11 we report the performance of different algorithms on three real-world data. Different from synthetic data streams where the decision concepts in data chunks gradually change following the formula given in Eq.(30), the data chunks of the real-word data do not share such property, and the concept drifting among data chunks are not clear to us (in fact, we do not even know the genuine concepts of the data). Because of this, we compare algorithms on two types of test sets. In Figure 10, the algorithms are tested on all instances in data chunk S_n, this is exactly like what we did with synthetic data streams. In Figure 11, the algorithms are tested on a separate test set generated from 10-fold cross validation. In this case the training set was treated as a data stream and the models built from each data chunk are evaluated on the test set. Our objective is to validate that if a test set is isolated from the data stream and has weak correlations with the data chunks, whether our algorithm is still able to outperform other methods and provide good results.

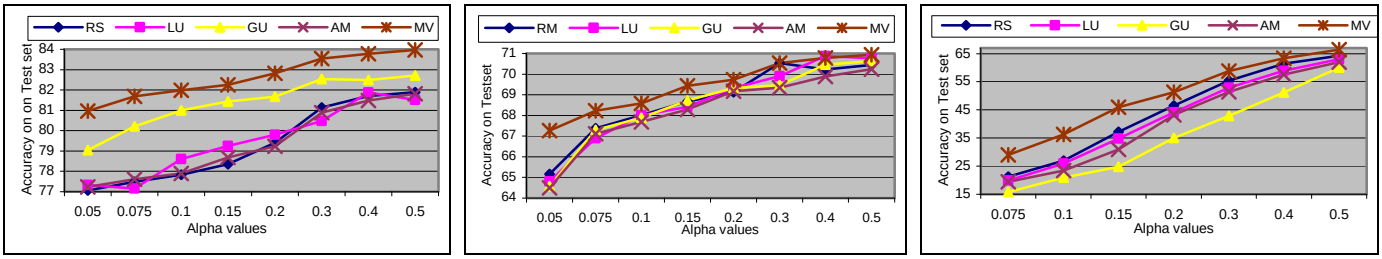
The results in Figures 10 and 11 indicate that, overall, the accuracies evaluated on individual data chunks are

slightly better than the accuracies acquired from the isolated test set. This confirms that in stream data environments, utilizing correlations between test set and data chunks can help deliver a better prediction model than those ignoring such correlation (or if the test set has no correlation with the data stream at all). But overall, an algorithm's relative performance on each individual data chunk or on a separate test set does not make big difference. If method A's performance is worse than B in Figure 10, the same observations can be made in Figure 11. The results in Figures 10 and 11, 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 the accuracy of GU can be as much as 20% lower than MV. Considering Letter is a sparse dataset with 26 evenly distributed classes, it shows that for data streams with a large number of classes, traditional uncertainty sampling would perform poorly. This asserts our analysis in Section III and Section VI.B, that for sparse data streams with a large number of classes, the classifiers built from different data chunk 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 random sampling.



(a) Adult (b) Covtype (c) Letter

Figure 10: Classifier ensemble accuracy on data chunk S_n (chunk size 500, e=3, C4.5)



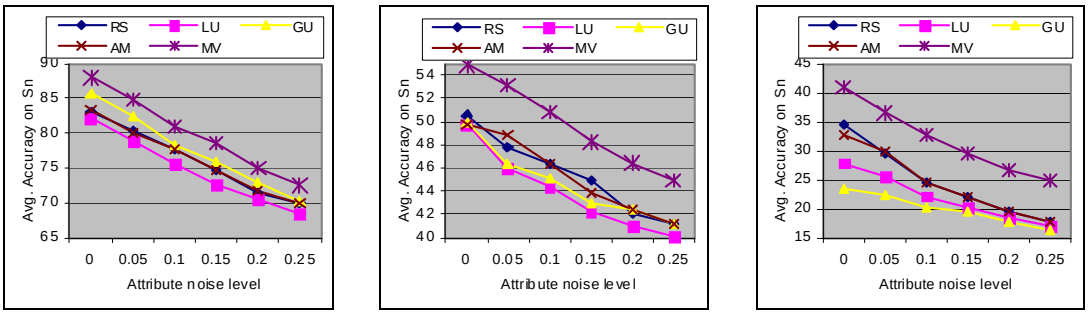
(a) Adult (b) Covtype (c) Letter

Figure 11: Classifier ensemble accuracy on a separate test set (chunk size 500, e=3, C4.5)

B.4. Active Learning on Noisy Data Streams

The adoption of the chunk based ensemble learning framework for data streams raises a possible concern that the whole framework may be sensitive to the attribute noise, especially when the size of the data chunk is small. This is because (1) the classifier learnt from each chunk may over-fit to the noisy data, and (2) the instance selection process may focus on "noisy" samples since they might be the ones responsible for a high variance value. To assess the system performance on 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, say p=0.1, the noise are injected such that each attribute value of x has a probability of p to be changed to another randomly selected value. So if x has 10 attributes and p=0.1, it means that, on average, at least one attribute of x is noise corrupted.

In Figure 12, we report the results on three data streams, with the attribute noise varying from 0 to 0.25. Overall, attribute noise bring significant impact to the underlying methods, where for as little as 5 percent attribute noise the prediction accuracy can deteriorate up to 6%. When comparing RD to MV, we can find that MV constantly outperforms RD across all noise levels, and their performance both deteriorate, as the increase of the noise level, at almost the same rate. This asserts that comparing to random selection, which has no special treatment for noise, the proposed MV framework does not show to be any more vulnerable to noise than random sampling. Indeed, as long as noise are randomly distributed in some or across all attributes, the chance for an "important" instance to be corrupted as an "unimportant" instance, or vice versa, is likely the same. Therefore, the proposed MV method is practically not sensitive to noisy data and small chunk sizes.



(a) c2-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (b) c4-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 (c) Letter

Figure 12: The active learning result comparisons on noisy stream data. The *x*-axis denotes the attribute noise level (the probability that each attribute value is changed to any other random value) and the *y*-axis denote the average active learning accuracy across all chunks of the data stream (the chunk size is 500 and α=0.1)

B.5. Runtime Performance Study

In Figure 13, we report the system runtime performance in response to different chunk sizes. The *x*-axis in Figure 13 denotes the chunk size and the *y*-axis denotes the average system runtime *w.r.t.* a single data chunk. Comparing all five methods, 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, RD has demonstrated itself to be the 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 for each chunk. On average, when the chunk size is 5000 or less, the runtime of MV, is about 2 to 4 times more 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 epochs. Such observations are consistent with the analysis in Section V.C and suggest that MV prefers smaller chunk sizes from the computational perspective.

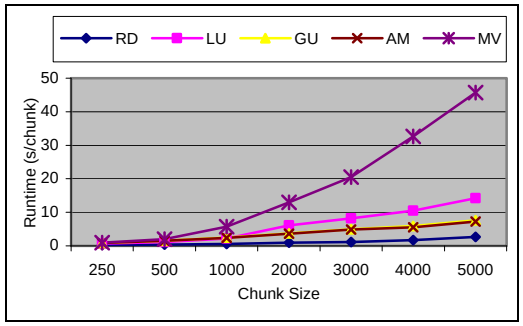


Figure 13: The system runtime with respect to different chunk sizes (c4-I50*k*-*d*10-*p*5-N1000-*t*0.1-*h*0.2 stream and α=0.5).

VII. CONCLUSIONS

In this paper, we 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 data to form a classifier ensemble with minimum error rate in predicting future samples. In order 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 a Minimum-variance (MV) principle for active learning from stream data, where the key is to label instances which are 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 ensemble classifier, 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, especially when dealing with multi-class problems. Simply applying uncertainty sampling globally or locally 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 instance labeling process can quickly adapt to the drifting concepts in stream data.

REFERENCES

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

[2]. P. Domingos & G. Hulten, Mining high-speed data streams, in *Proc. of KDD*, 2000.

[3]. H. Wang, W. Fan, P. Yu, & J. Han, Mining concept-drifting data streams using ensemble classifiers, in *Proc. of KDD*, 2003.

[4]. P. Zhang, X. Zhu, and L. Guo, Mining Data Streams with Labeled and Unlabeled Training Examples, in *Proc. of ICDM*, 2009.

[5]. H. Wang, J. Yin, J. Pei, P. Yu, & J. Yu, Suppressing model overfitting in mining concept-drifting data streams, in *Proc. of KDD*, 2006.

[6]. Y. Yang, X. Wu, & X. Zhu, Combining proactive and reactive predictions of data streams, in *Proc. of KDD*, 2005.

[7]. W. Street & Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in *Proc. of KDD*, 2001.

[8]. J. Gao, W. Fan, J. Han, and P. Yu, A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions, in *Proc. of SIAM International Conference on Data Mining*, 2007.

[9]. W. Fan, Y. Huang, H. Wang, and P. Yu, Active Mining of Data Streams, in *Proc. of SIAM International Conf. on Data Mining*, 2004.

[10]. X. Zhu, P. Zhang, X. Wu, D. He, C. Zhang, and Y. Shi, Cleansing Noisy Data Streams, in *Proc. of the 8th IEEE International Conference on Data Mining*, 2008.

[11]. D. Cohn, L. Atlas, R. Ladner, Improving Generalization with Active Learning, *Machine Learning* 15(2), 1994.

[12]. X. Zhu, X. Wu, and Q. Chen, Eliminating Class Noise in Large Datasets, in *Proc. of ICML*, 2003.

[13]. H. Seung, M. Opper, & H. Sompolinsky, Query by committee, in *Proc. of COLT*, 1992.

[14]. Y. Freund, H. Seung, & E. Tishby, Selective sampling using the query by committee algorithm, *Machine Learning*, 1997.

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

[16]. H. Nguyen & A. Smeulders, Active learning using pre-clustering, in *Proc. of ICML*, 2004.

[17]. M. Culver, D. Kun, & S. Scott, Active Learning to Maximize Area Under the ROC Curve, in *Proc. of ICDM*, 2006.

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

[19]. P. Utgoff, Incremental induction of decision trees, *Mach. Learn.*, 1989.

[20]. K. Tumer & J. Ghosh, Error correlation and error reduction in ensemble classifier, *Connection Science*, 8(3-4):385-404, 1996.

[21]. K. Tumer & J. Ghosh, Analysis of decision boundaries in linearly combined neural classifiers, *Pattern Recognitions*, 1996.

[22]. R. Kohavi & D. Wolpert, Bias plus variance decomposition for zero-one loss function, in *Proc. of ICML* 1996.

[23]. X. Zhu and X. Wu, Class Noise vs Attribute Noise: A Quantitative Study of Their Impacts *Artificial Intelligence Review*, 22 (3-4):177-210, November 2004.

[24]. J. Snyman, *Practical Mathematical Optimization.* Springer, 2005.

[25]. I. Witten & E. Frank, *Data mining: practical machine learning tools and techniques*, Morgan Kaufmann, 2005.

[26]. J. Quinlan, C4.5: *Programs for Machine learning*, M. Kaufmann, 1993.

[27]. D. Newman, S. Hettich, C. Blake, C Merz. *UCI Repository of machine learning*, 1998.

[28]. P. Domingos & M. Pazzani, *On* the Optimality of the Simple Bayesian Classifier under Zero-One Loss, *Machine Learning*, 1997.

[29]. L. Breiman, Bias, variance, and arching classifiers, *Technical report #460*, Statistics Department, Univ. of California at Berkeley (1996).

[30]. J. Friedman, On bias, variance, 0/1-loss, and the curse-of-dimensionality, *Data Mining and Knowledge Discover*, vol. 1, 1996.
[31]. E. Kong and T. Dietterich, Error-correcting output coding corrects bias and variance, In *Proc. of the 12th ICML Conf.*, pp.313-321, CA, 1995.
[32]. D. Moore and G. McCabe, *Introduction to the Practice of Statistics, Fourth Edition.* Michelle Julet, 2002.
[33]. C. Corinna and V. Vapnik, Support-Vector Networks, *Machine Learning, 20, 1995*
[34]. J. Kittler, M. Hatef, R. Duin, and J. Matas, On combining classifiers, *IEEE Trans.* on *Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226-239, 1998.
[35]. T. Ho, J. Hull, and S. Srihari, Decision combination in multiple classifier systems, *IEEE Trans. On PAMI*, 16(1): 66-75, 1994.
[36]. T. Dietterich, An experimental comparison of three methods for constructing ensemble of decision trees: Bagging, Boosting, and Randomization, *Machine Learning*, 40(2):139-157, 2000.
[37]. J. Kittler and F. Alkoot, Sum versus Vote Fusion in Multiple Classifier Systems, *IEEE Trans. On PAMI*, 25(1):110-116, 2003.
[38]. P. Wang, H. Wang , X. Wu , W. Wang , B. Shi, A Low-Granularity Classifier for Data Streams with Concept Drifts and Biased Class Distribution, *IEEE Transactions on TKDE*, 19(9):1202-1213, 2007.
[39]. P. Mitra, C. Murthy, and S. Pal, A Probabilistic Active Support Vector Learning Algorithm, *IEEE Trans. on PAMI*, 26(3):413-418, 2004.
[40]. D. Cohn, Z. Ghahramani, and M. Jordan, Active Learning with Statistical Models. *Artificial Intelligence Research*, 4:129-145, 1996.
[41]. C. Campbell, N. Cristianini, and A. Smola, Query Learning with Large Margin Classifiers. In *Proc. of ICML*. 111-118, 2000.
[42]. S. Ji, B. Krishnapuram, and L. Carin, Variational Bayes for Continuous Hidden Markov Models and Its Application to Active Learning. *IEEE Trans. on PAMI*, 28(4):522-532, 2006.
[43]. D. Lewis and J. Catlett, Heterogeneous uncertainty sampling for supervised learning. In *Proc. of ICML*, 148-156, 1994.
[44]. N. Abe and H. Mamitsuka, Query learning strategies using Boosting and Bagging, in *Proceedings of Fifteenth International Conference on Machine Learning (ICML 98)*, Madison WI, 1998.
[45]. X. Zhu and X. Wu, Class Noise Handling for Effective Cost-Sensitive Learning by Cost-Guided Iterative Classification Filtering, *IEEE Trans. on KDE*, 18(10):1435-1440, 2006
[46]. X. Zhu, P. Zhang, X. Lin, and Y. Shi, Active learning from data streams, in *Proc. of IEEE International Conference on Data Mining* (ICDM), 2007.
[47]. J. Rodriguez, L. Kuncheva, and C. Alonso, Rotation Forest: A New Classifier Ensemble Method, *IEEE Trans. on PAMI*, 28(10), 2006.
[48]. Melville P. and Mooney R. Diverse ensembles for active learning, in *Proceedings of the 21st ICML Conference*, 2004.
[49]. H. Mamitsuka and N. Abe, Active Ensemble Learning: Application to Data Mining and Bioinformatics, Systems and Computers in Japan, 38(11), 2007.
[50]. J. Huang, S. Ertekin, Y. Song, H. Zha, and C. Giles, Efficient Multiclass Boosting Classification with Active Learning, *Proc. of SDM*, 2007.
[51]. N. Pedrajas, C. Osorio, and C. Fyfe, Nonlinear Boosting Projections for Ensemble Construction, *J. of Machine Learning Research*, vol. 8, 2007.
[52]. L. Breiman, Bagging Predictors, *Machine Learning*, pp.123-140, 1996.
[53]. Y. Freud and R. Schapire, Experiments with a New Boosting Algorithm, *Proc. of ICML Conference*, 1996.
[54]. W. Hu, W. Hu, N. Xie, and S. Maybank, Unsupervised Active Learning Based on Hierarchical Graph-Theoretic Clustering, IEEE Trans. on Systems, Man, and Cybernetics (Part B), 39(5), October, 2009.
[55]. G. Stemp-Morlock, Learning more about Active Learning, Communications of the ACM, vol. 52, no. 4, April 2009.
[56]. C. Monteleoni and M. Kaariainen, Practical Online Active Learning for Classification, CVPR Online Learning for Classification Workshop, 2007.
[57]. B. Settles, Active Learning Literature Survey, Computer Science Technical Report 1648, University of Wisconsin-Madison, 2009.
[58]. J. Berger, *Statistical Decision Theory and Bayesian Analysis*, Second ed. Springer, 1985.