Decision Support Systems 50 (2011) 469–479

Contents lists available at ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss

Robust ensemble learning for mining noisy data streams

Peng Zhang [a,*], Xingquan Zhu [b], Yong Shi [c,d], Li Guo [a], Xindong Wu [e,f]

[a] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China
[b] Centre for Quantum Computation & Intelligent Systems, University of Technology Sydney, Broadway, NSW 2007, Australia
[c] Research Center on Fictitious Economy and Data Science, Chinese Academy of Sciences, Beijing, China
[d] College of Information Science & Technology, Univ. of Nebraska at Omaha, Omaha, NE 68182, USA
[e] School of Computer Science & Information Eng., Hefei University of Technology, Hefei 230009, China
[f] Department of Computer Science, University of Vermont, Burlington, VT 05405, USA

ARTICLE INFO

Article history:
Received 1 August 2009
Received in revised form 9 October 2010
Accepted 1 November 2010
Available online 5 November 2010

Keywords:
Data stream
Classification
Ensemble learning
Noise
Concept drifting

ABSTRACT

In this paper, we study the problem of learning from concept drifting data streams with noise, where samples in a data stream may be mislabeled or contain erroneous values. Our essential goal is to build a robust prediction model from noisy stream data to accurately predict future samples. For noisy data sources, most existing works rely on data preprocessing techniques to cleanse noisy samples before the training of decision models. In data stream environments, these data preprocessing techniques are, unfortunately, hard to apply, mainly because the concept drifting in a data stream may make it very difficult to differentiate noise from samples of changing concepts. Accordingly, we propose an aggregate ensemble (AE) learning framework. The aim of AE is to build a robust ensemble model that can tolerate data errors. Theoretical and empirical studies on both synthetic and real-world data streams demonstrate that the proposed AE learning framework is capable of building accurate classification models from noisy data streams.

1. Introduction

Recent advances in networking, data collection, storage, and transmission have promoted a new type of data-intensive applications which rely on data streams for decision making [7,30]. Examples of such applications include wireless sensor networks, traffic management, telephone call records, online transactions, web servers' logs and so on. In order to discover knowledge from data streams, many stream mining based methods have been proposed. These methods, depending on the data characteristics and data collection objectives, can be roughly distinguished into three categories: continuous query and clustering data streams [2,6,8,9,12,20,22,27], frequent pattern mining from data streams [13,25,26,28,39], and generating predictive models for data streams [1,17–19,24,34,41,43–46].

From the classification perspective, building classification models on data streams usually confronts two challenges: (1) tremendous volumes of streaming data; and (2) continuous change of the decision concepts underneath the stream data, which is commonly referred to as *concept drifting*. In data stream environments, concept drifting usually happens in different ways: (1) gradual and moderate changes, and (2) abrupt and severe changes, which are illustrated in Fig. 1. From Fig. 1(a) to (b), the classification boundary c_1 gradually changes to c_2; on the other hand, from Fig. 1(b) to (c), the classification boundary abruptly changes from c_2 to c_3, and there is no connection (or correlations), at the observed moment, between these two boundaries (*i.e.*, c_2 and c_3).

The challenges from *large volumes of data* and *concept drifting* raise the needs of designing effective classification models with high accuracy and good efficiency. Motivated by the above challenges, existing classification models in the field can be roughly categorized into two groups: online incremental learning [16,23,31,37] and ensemble learning [24,34,35,38,47]. The incremental learning strategy tries to design a single learning model to represent an entire data stream by continuously updating the model with the newly arriving data, such that the model can always capture the most recent decision logics in the data stream. On the other hand, ensemble learning regards a data stream as separated data chunks, and builds several base classifiers from separated data chunks to generate an ensemble classifier for prediction. Although these models were proved to be effective and accurate, an inherent limitation is that they were mainly designed for quality stream data without an explicit consideration of the data errors. Consequently, these learning frameworks are likely to suffer a great loss when handling real-world data streams containing erroneous data values.

Indeed, in traditional data mining tasks, a large number of methods exist for tackling noise or erroneous attribute errors [10,29,32,36,49]. These methods can be roughly categorized into two types: (1) data preprocessing methods, and (2) robust learning methods. Data preprocessing focuses on identifying and cleansing noisy data, such that the cleansed data can be used to build accurate

* Corresponding author. Tel.: +86 10 6260 0979.
E-mail address: zhangpeng@ict.ac.cn (P. Zhang).

0167-9236/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2010.11.004

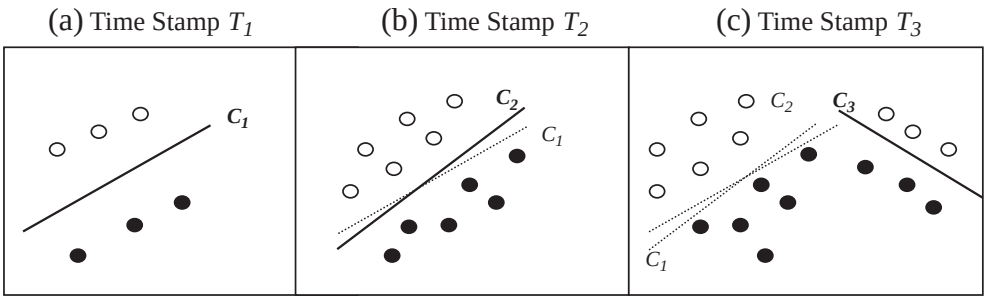

Fig. 1. A conceptual view of concept drifting in data streams. In three consecutive time stamps, the classification boundary drifts from c_1 to c_2, and finally to c_3. From T_1 to T_2, the classification boundary changes gradually, while from T_2 to T_3, the classification boundary changes abruptly.

prediction models. For instance, Brodley and Friedl [10] concluded that identifying and removing mislabeled training examples can help generate more accurate prediction models than the ones trained from raw data. Quinlan [32] and Zhu et al. [49] studied the impact of data errors on inductive learning and asserted that data errors are the main sources of the classification errors, and data cleansing is an effective tool to help a learning algorithm achieve performance gain. On the other hand, a robust learning method tries to build robust models that can greatly, if not completely, reduce the noise impact. Two representative robust learning approaches include pruning for decision trees and prototype selection for instance-based learning [3]. The essential idea behind these approaches is to simplify the prediction models and prevent the learner from overfitting to the noisy data. In addition to the single learner based approach, classifier ensembling [4,15,21] is another type of robust learning which shows good performance on noisy data.

For noisy data streams, most existing research focuses on designing effective data preprocessing algorithms to cleanse noise from data streams, such that the cleansed data can be used to build accurate models. For example, Chu et al. [14] proposed a statistical estimation framework to identify outliers in data streams. Zhu et al. [50] proposed a maximum variance margin (MVM) based filtering framework to cleansing noise. Wang et al. [40] proposed a clustering-based method to wipe off noise. Although these methods are effective in their own problem definitions, most of them share two common disadvantages. First, most of them implicitly make some statistical assumptions to describe data streams, whereas the assumed statistical models may not always exist in reality. Second, all these methods usually employ a multi-scan learning approach to cleanse noise, and then learn from the cleansed data. In dynamic data stream environments, it is urged that the training of prediction models should only require one scanning of the stream data.

The above observations motivate our research on robust learning from noisy data streams using a new aggregate ensemble (AE) framework. In our proposed design, AE first trains base classifiers using different learning algorithms on different data chunks. It then combines all the base classifiers to form a classifier ensemble through model voting. By doing so, AE is supposed to be robust for both concept drifting and noisy data problems. Experimental results on both synthetic and real-world data streams show that the AE framework is superior to other ensemble-based learning frameworks for noisy data streams.

The remainder of this paper is structured as follows. Section 2 describes noise in data streams in general. Section 3 introduces the proposed AE framework. Section 4 provides theoretical studies on the AE framework. Section 5 empirically studies the AE framework on both synthetic and real-world data streams. We conclude the paper in Section 6.

2. Noisy description for data streams

According to the characteristics of the stream data, existing work roughly describes data streams into the following two styles: stationary data streams [16,23,38,43] and dynamic data streams [18,44–46].

According to the stationary description, if data streams are divided into data chunks as shown in Fig. 2, then training data chunks (which include both historical data chunks and the up-to-date chunk) will have a similar or identical distribution to the yet-to-come data chunk. So classifiers built from the training data chunks will have reasonably good performance in classifying data from the yet-to-come data chunk. The advantage of the stationary description is that we may directly apply traditional classification techniques to the data streams. For example, since the up-to-date data chunks have the same distribution as the yet-to-come data chunk, we can collect all historical classifiers to build a classifier ensemble. However, this stationary description takes no consideration of the concept drifting in stream data, so it can hardly, if not impossible, be used to describe most real-world data streams.

Noticing the limitations of the stationary description, a recent work [18] describes the data streams in a dynamic scenario where training chunks have different distributions $p(x,y)$ (where x denotes the feature vector and y denotes the class label) from that of the yet-to-come data chunk, and classifiers built on the training set may perform only slightly better than random guessing or simply predicting all examples to belong to one single class. Comparing to the stationary description, the dynamic description emphasizes on the situation that training data chunks do not necessarily have the same distribution as the yet-to-come data chunk. Under this description, building classifiers from the up-to-date data chunk to predict the yet-to-come data chunk is better than building classifiers from the aggregation of all historical chunks because the buffered chunks (probably outdated with respect to the newly arrived data chunk) will deteriorate the ensemble performance. In a narrow sense, this dynamic description is much looser than the stationary description, which makes it more applicable for mining concept drifting data streams. However, the disadvantage of the dynamic description is also obvious, in the sense that it doesn't discriminate concept drifting from data errors. If the up-to-date data chunk contains noisy samples, building classifiers on this noisy data chunk to predict the yet-to-come data chunk may cause more errors than using a classifier ensemble built on previously buffered data chunks. Consequently, although the dynamic description is more reasonable than the stationary description for data streams, in practice, it is still not capable of describing all the realistic data streams.

Consider a data stream management system whose buffer contains five consecutive data chunks as shown in Fig. 3. The stationary description can only cover the process from D_1 to D_2, where the distribution $p_1(x,y)$ remains unchanged. The dynamic description covers the process from D_2 to D_3, where the concept drifts from $p_1(x,y)$ to $p_2(x,y)$ without being interrupted by noisy data chunks. A more general situation, as depicted in the process from D_3 to D_5, is that the concept drifting ($p_2(x,y)$ evolves to $p_3(x,y)$) is mixed with noise (a noisy data chunk D_4 is observed). To explicitly describe this type of data streams, we define a noisy description of data streams as follows:

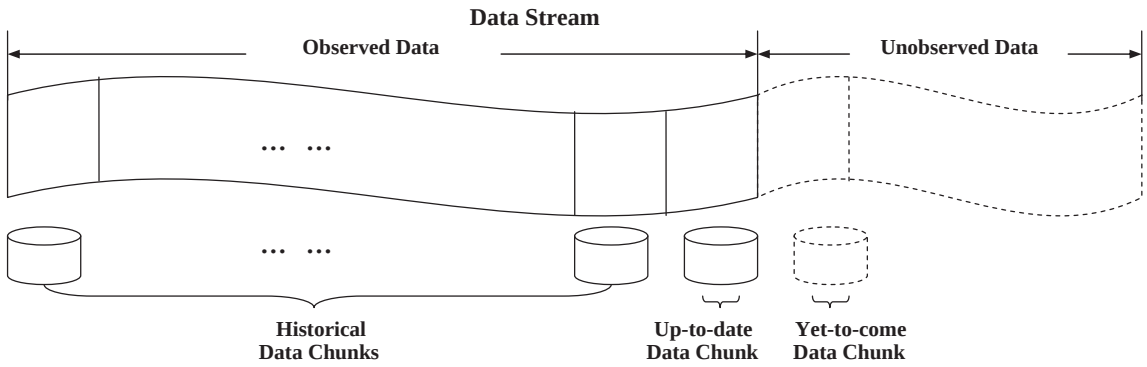

Fig. 2. An illustration of the "historical", "up-to-date" and "yet-to-come" data chunks. A data stream can be split into two parts: the observed data stream (which is denoted by the solid lines) and the unobserved data stream (which is denoted by the dotted lines). Assume the data stream is processed chunk-by-chunk. The observed data stream can be further categorized into two types: the latest data chunk is called the "up-to-date" chunk, while the remaining data chunks are called the "historical" data chunks. Besides, the "yet-to-come" data chunk is the first data chunk of the unobserved data streams.

Noisy description for data streams: Mining from real-world data streams may confront the challenges of concept drifting and data errors simultaneously.

The noisy description addresses both concept drifting and data errors in a data stream management system. It is much more general than the stationary and dynamic descriptions. So it can be adapted for generic data streams.

3. Ensemble frameworks for mining data streams

The nature of continuous volumes of the stream data raises the needs of designing effective classifiers with high accuracy in predicting future testing instances as well as good efficiency in handling massive volumes of training instances. In the past few years, many solutions have been proposed to build prediction models from data streams. An early solution is to build model by using online incremental methods [16,23] which update a single model by incorporating newly arrived data. During the learning process, incremental methods continuously revise the model to discover new patterns in the most recent data chunk. For example, Domingos and Hulten [16] introduced an ultra fast decision tree learner VFDT which incrementally builds Hoeffding trees from the high-volume data streams. Similar approach was extended to CVFDT [23] which handles time changing and concept drifting streams. By doing so, most of the incremental methods violate the efficiency rule because updating a classifier according to the newly arrived data can be a time-consuming process. An alternative solution is to build a single and simple classifier on the up-to-date chunk without considering historical data chunks, *i.e.*, discarding old classifiers and rebuilding a new classifier on the new data chunk. This build-then-discard method, unfortunately, may not work well because of the important loss incurred by the discarded classifiers. To overcome this challenge, a number of ensemble methods have been proposed.

Different from the incremental learning where the goal is to deliver a single model, ensemble learning intends to produce a number of models and relies on their voting for final predictions. Such design brings two advantages for ensemble learning to handle data streams: (1) because models are trained from a small portion of stream data, it can efficiently handle streams with fast growing data volumes; and (2) because the final predictions are the voting of a number of base models, the concept drifting in the stream can be adaptively and rapidly addressed by changing the weight value of each voting member. For example, Street and Kim [35] proposed a SEA algorithm, which combines decision tree models using majority-voting. Kolter and Maloof [24] proposed an ensemble method by using weighted online learners to handle drifting concepts. Wang et al. [38] proposed a weighted ensemble, in which they assign each classifier a weight reversely proportional to the classifier's accuracy on the most recent data chunk. Yang et al. [43] proposed proactive learning where concepts (models) learnt from previous chunks are used to foresee the best model to predict data in the current chunk. Zhu et al. [48] proposed an active learning framework to selectively label instances for concept drifting data streams. Gao et al. [18]

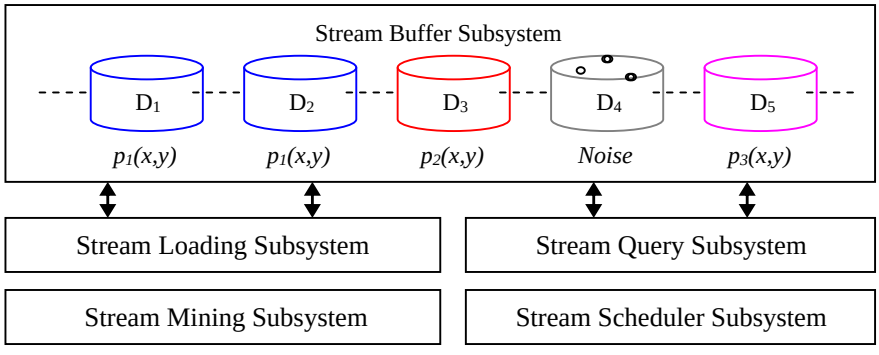

Fig. 3. A conceptual view of noisy data in data stream management system. The data stream management system can be separated into five parts: a stream buffer subsystem, a stream loading subsystem, a stream query subsystem, a stream mining subsystem, and a stream scheduler subsystem. In the stream buffer subsystem, there are five buffered data chunks, D_1, D_2, …, D_5, of which D_4 is a noise data chunk. D_1 and D_2 share the same distribution $P_1(x,y)$. From D_2 to D_3, the underlying concept changes from $P_1(x,y)$ to $P_2(x,y)$. From D_3 to D_4 and finally to D_5, the concept changes from $P_2(x,y)$ to $P_3(x,y)$, meanwhile, a noisy chunk D_4 is observed between D_3 and D_5. The stationary description of data streams can only cover the process from D_1 to D_2, while the dynamic description of data streams only covers the process from D_2 to D_3. Our noisy description covers a much more common process from D_3 to D_5.

proposed to build different base classifiers on a most recent data chunk to construct the classifier ensemble.

In summary, the above ensemble frameworks for stream data mining can be roughly categorized into the following two categories, according to their ways of forming the base classifiers: horizontal ensemble (including weighted ensemble) frameworks which build base classifiers using several buffered data chunks (as illustrated in Fig. 4(a)), and vertical ensemble framework which build base classifiers on the up-to-date data chunk using different algorithms (as illustrated in Fig. 4(b)).

3.1. Horizontal ensemble and weighted ensemble frameworks

Consider a data stream containing an infinite number of data chunks $\{D_i\}_{i=1}^{+\infty}$. Due to the limitation of the storage space, the system buffer can only accommodate at most n consecutive chunks each of which contains a certain number of instances. Assume at the current time stamp we are observing the nth chunk D_n, and the buffered data chunks are denoted by D_1, D_2, …, D_n. In order to predict data in a newly arrived chunk D_{n+1}, one can choose a learning algorithm L to build a base classifier f_i from each of the buffered data chunks D_i, say

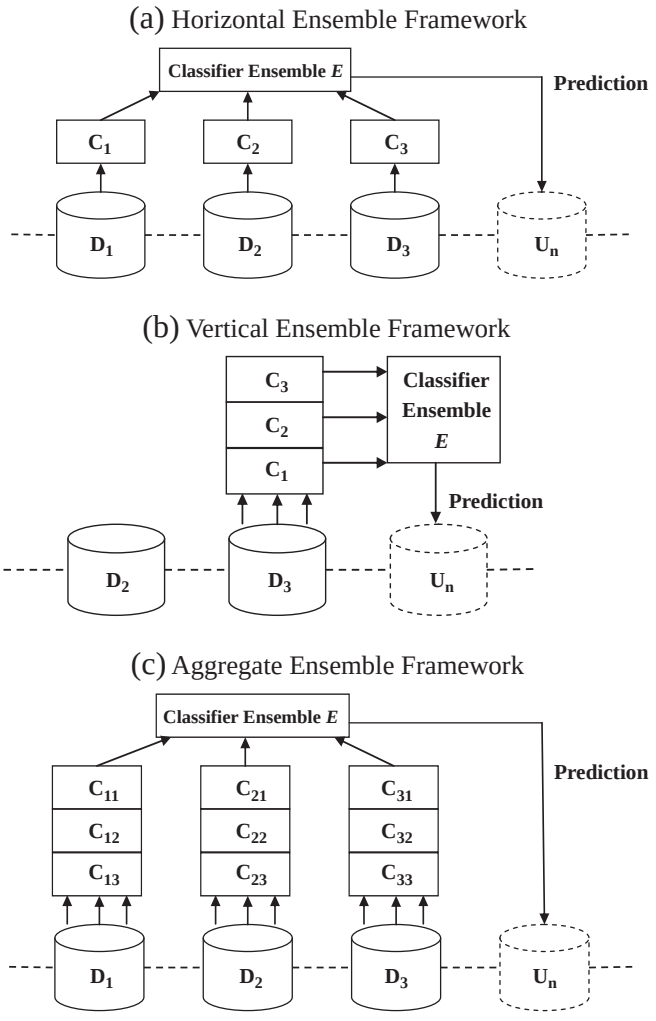

Fig. 4. A conceptual flowchart of the classifier ensemble framework for stream data mining where (a) shows the horizontal ensemble framework, which builds different classifiers on different data chunks; (b) shows the vertical ensemble framework, which builds different classifiers on the up-to-date data chunk with different learning algorithms; and (c) shows the aggregate ensemble framework, which builds classifiers on different data chunks using different learning algorithms.

$f_i = \mathcal{L}(D_i)$, and then predict each instance x in D_{n+1} by combining the predictions of the base classifiers f_i ($i = 1, 2, \ldots, N$) to form a classifier ensemble through the model averaging mechanism shown in Eq. (1) [15,24,38,48].

$$f_{HE}(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x) \tag{1}$$

An alternative version of the horizontal ensemble is to add weight values to the base classifiers [38,48]. Different from the model averaging, a weighted ensemble minimizes the variance error e_v of each base classifier on the up-to-date data chunk, then assigns each classifier a weight that is reversely proportional to the error rate e_v. The advantage of the horizontal ensemble and weighted ensemble is twofold: (1) they can reuse information of the buffered data chunks, which may be beneficial for the testing data chunk; and (2) they are robust to noisy streams because the final decisions are based on the classifiers trained from different chunks. Even if noisy data chunks may deteriorate some base classifiers, the ensemble can still maintain relatively stable prediction accuracy. The disadvantage of such an ensemble framework, however, lies in the fact that if the concepts of the stream continuously change, information contained in previously buffered classifiers may be invalid to the current data chunk. As a result, combining old-fashioned classifiers may not improve the overall prediction accuracy. In summary, both horizontal and weighted ensembles, in fact, are based on the stationary description of the data streams that buffered data chunks share similar or identical distributions to the yet-to-come data chunk, such that information in the buffered data chunks can be used to predict the yet-to-come data chunk.

3.2. Vertical ensemble framework

Assume we have m learning algorithms L_j ($j = 1,2,\ldots,m$), a vertical ensemble [18,45] builds base classifiers using each algorithm on the up-to-date data chunk D_n as $f_j = \mathcal{L}_j(D_n)$, and then combines all base classifiers through model averaging as given in Eq. (2),

$$f^n_{VE}(x) = \frac{1}{m}\sum_{i=1}^{m} f_{in}(x). \tag{2}$$

In the case that prior knowledge of the yet-to-come data chunk is unknown, model averaging on the most recent chunk can achieve minimum expectation error on the test set. In other words, building classifiers using different learning algorithms can decrease the expected bias error compared to any single classifiers. For example, assuming a data stream whose joint probability $p(x,y)$ evolves continuously, if we only use a stable learner such as SVM, then SVM may perform better than an unstable classifier when $p(x)$ changes while $p(y|x)$ remains unchanged. On the other hand, if we only use an unstable learner such as decision trees, then decision trees may perform better than SVM when $p(x)$ does not evolve much but $p(y|x)$ changes dramatically. When we have no prior knowledge on whether the evolving of $p(x,y)$ is triggered by $p(x)$ or $p(y|x)$, it is difficult to determine whether a stable classifier or an unstable classifier is better, so combining these two types of classifiers is likely to be a better solution than simply choosing either of them. Although the vertical ensemble has a much looser condition (distribution $p(x,y)$ may continuously change) than the stationary description (distribution $p(x,y)$ remains unchanged), it also has a severe pitfall for realistic data streams. The vertical ensemble builds classifiers only on a single up-to-date data chunk, but as we have discussed before, a realistic data stream system may contain data errors. If the up-to-date data chunk is a noisy data chunk, the results may suffer from severe performance deterioration. Without realizing the noise problems, the vertical

ensemble limits itself merely to the concept drifting scenarios, but not to the realistic data streams.

3.3. Aggregate ensemble framework

The disadvantages of the above two ensemble frameworks motivate the proposed aggregate ensemble framework (which is illustrated in Fig. 4(c)). We first use m learning algorithms L_i ($i=1, 2, \ldots, m$) to build classifiers on n buffered data chunks j ($j=1,\ldots, n$), and then train m-by-n base classifiers $f_{ij} = \mathcal{L}_i(D_j)$, where i denotes the ith algorithm, and j denotes the jth data chunk. Then we combine these base classifiers to form an aggregate ensemble through model averaging defined in Eq. (3), which indicates that the aggregate ensemble is a mixture of the horizontal ensemble and vertical ensemble, and its base classifiers constitute a *classifier matrix* (*CM*) in Eq.(4).

$$f_{AE} = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}(x) \tag{3}$$

$$CM = \begin{bmatrix} f_{11}\ f_{12}\cdots\cdots f_{1n} \\ f_{21}\ f_{22}\cdots\cdots f_{2n} \\ \cdots\cdots\cdots\cdots \\ f_{m1}\ f_{m2}\ \cdots\cdots f_{mn} \end{bmatrix}_{m*n} \tag{4}$$

In Eq. (4), each element f_{ij} in *CM* represents a base classifier built by using algorithm i on data chunk j. As we have mentioned in the vertical ensemble, classifiers on each column of *CM* (*i.e.*, classifiers built on the same data chunk using different learning algorithms) are used to reduce the expected classifier bias error on unknown test data. Classifiers on each row of *CM* (*i.e.*, classifiers built on different data chunks using the same learning algorithm) are used to reduce the impact of noisy data chunks. For example, when the up-to-date training chunk is a noisy chunk, combining classifiers built from the historical data chunks may alleviate the noisy impact. By building a classifier matrix *CM*, the aggregate ensemble is capable of solving a realistic data stream containing both concept drifting and data errors.

4. Theoretical studies of the aggregate ensemble

4.1. Performance study of AE framework

In this subsection, we explore why and when AE performs better than HE and VE methods. As we have described in the earlier section, on each data chunk, the aggregate ensemble builds m classifiers by using m different learning algorithms. For a specific test instance x in the yet-to-come data chunk, the horizontal ensemble uses classifiers on a row in matrix *CM* to predict x, *i.e.*, if we choose learning algorithm i ($1 \le i \le m$), then the horizontal ensemble can be denoted by Eq. (5)

$$f^i_{HE}(x) = \frac{1}{n}\sum_{j=1}^{n} f_{ij}(x). \tag{5}$$

The vertical ensemble can be denoted by model averaging on the last column (column n) of the Matrix *CM*, which is given in Eq. (6)

$$f^n_{VE}(x) = \frac{1}{m}\sum_{i=1}^{m} f_{in}(x). \tag{6}$$

An aggregate ensemble combines all classifiers in *CM* as base classifiers, through the averaging rule defined by Eq. (2). Accordingly, the horizontal ensemble and vertical ensemble, as special cases of the aggregate ensemble, are, in fact, two special cases of the aggregate ensemble. Gao et al. [18] proved that in data stream scenario, the performance of a single classifier within a classifier ensemble is expected to be inferior to the performance of the entire classifier ensemble. The horizontal ensemble and vertical ensemble, as special cases of the aggregate ensemble, are not expected as good as the aggregate ensemble. For example, when combining each column in *CM*, one can have a variant of *CM* as $CM_c = [g_1, g_2, \ldots, g_n]$, where each $g_i = [f_{1i}, f_{2i}, \ldots, f_{mi}]^T$ is independent of each other and shares the same distribution, say $p(g)$. Then the mean squared error of the horizontal ensemble (with the ith learning algorithm) on a test instance x (with class label y) can be denoted by

$$MSE^i_{HE}(x) = E_{p(g)}(y - g_i(x))^2 = y^2 - 2y\cdot E_{p(g)}g_i(x) + E_{p(g)}g_i^2(x). \tag{7}$$

For the aggregate ensemble, the mean squared error on x can be calculated as

$$MSE_{AE}(x) = E_{p(g)}\left(y - E_{p(g)}g_i(x)\right)^2 = y^2 - 2y\cdot E_{p(g)}g_i(x) + E^2_{p(g)}g_i(x). \tag{8}$$

So the difference between Eqs. (7) and (6) is denoted by Eq. (9),

$$MSE_{AE}(x) - MSE^i_{HE}(x) = E^2_{p(g)}g_i(x) - E_{p(g)}g_i^2(x) \le 0. \left(\text{since } E^2(x) \le E\left(x^2\right)\right) \tag{9}$$

Accordingly, we assert that the error rate of the aggregate ensemble is expected to be less or equal to the error rate of the horizontal ensemble. Similarly, if we regard *CM* as a column vector where each element is a combination of different rows in *CM*, we can show that the mean squared error of the aggregate ensemble is also expected to be less or equal to that of the vertical ensemble.

In the following we provide some intuitive explanations on why and when AE performs better than HE and VE by using two toy examples in Figs. 5 and 6. Note that our comparisons here are rather intuitive and qualitative, and rigorous numeric comparisons will be reported in the experimental results in the next section. As shown in Fig. 5, assume that AE is trained using three learning algorithms M_1, M_2, and M_3, where HE(M_i) denotes an HE model trained using learning algorithm M_i. For each model, we list three results: (1) training accuracy at time A, (2) test accuracy at time A, and (3) test accuracy at time B which immediately follows A. We can observe that for concept drifting data streams, it is difficult to find a single "optimal" learning algorithm with the best performance across the whole stream. For example, model HE(M_2) has the best prediction accuracy at time stamp A, but unfortunately, it has the worst performance at the next time stamp B. Model HE(M_3) has the worst performance at time A, but it performs the best at time stamp B. On the other hand, AE can guarantee the most reliable performance by combining different learning algorithms. This is because in dynamic data stream environments it is essentially difficult to know which learning algorithm performs the best at a particular time point. By

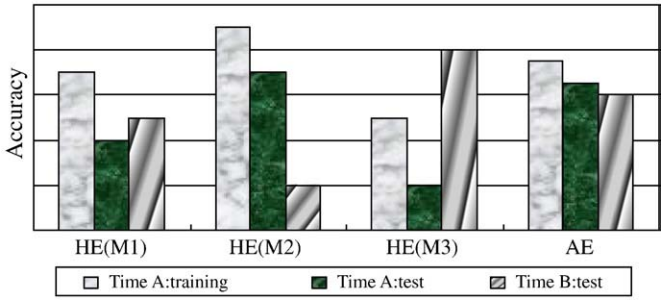

Fig. 5. A toy example for comparisons between AE and three HE ensemble methods trained with different learning algorithms (*i.e.*, algorithms M_1, M_2, and M_3). For each ensemble method, three results (bars) are listed for comparisons. The left bar denotes the training accuracy at time A, the bar in the middle denotes the test accuracy at time A, and the bar on the right denotes the test accuracy at time B which follows time stamp A. It is obvious that at time A, the higher the training accuracy, the better the prediction result. However, this result doesn't hold when the concept drifts at the next time stamp B.

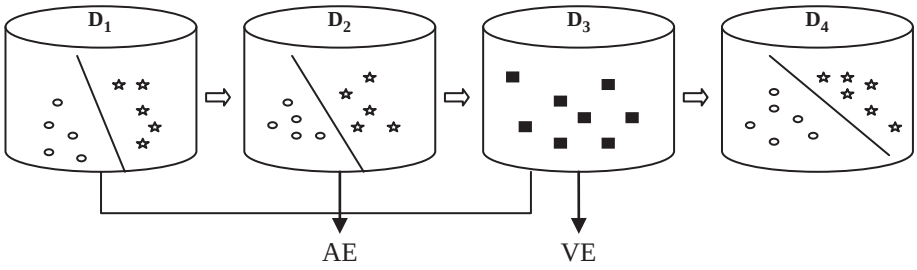

Fig. 6. A toy example for comparison between AE and VE. The concept (*i.e.*, the classification boundary) drifts marginally from chunk D_1 to D_2, and finally to D_4. Notice that the up-to-date chunk D_3 is a noisy chunk that carries useless or erroneous information when predicting the yet-to-come data chunk D_4.

integrating different learning algorithms as a unified model, we can expect AE to have the smallest variance error and thus have the best prediction accuracy.

AE performs better than VE when the concept drifts marginally and the up-to-date training chunk contains a significant amount of noisy samples. As illustrated in Fig. 6, assume that the concept drifts slightly along data chunks, and the up-to-date chunk D_3 is a noisy chunk. VE built on the up-to-date chunk D_3 will show deteriorated performance in predicting D_4. On the other hand, AE can largely avoid such a limitation by incorporating information from classifiers trained from the historical data chunks D_1 and D_2.

Although we have demonstrated that AE, on average, outperforms HE and VE, we are not claiming that AE always performs the best in data stream scenarios. For example, HE may outperform AE if the concept drifts marginally in data streams. In this case, the joint probability distribution $p(x,y)$ will stay stable across the data streams, and thus we can select a strong learning algorithm (*i.e.*, SVM) to construct HE and expect HE to outperform AE. On the other hand, VE may outperform AE if the concept drifts significantly and the up-to-date chunk contains very few noisy samples. In such a case, old-fashioned historical information in AE will deteriorate the learner performance even worse.

4.2. Time complexity analysis

In this subsection, we study the time complex of the AE framework and discuss whether it is a suitable model, from computational cost perspective, for mining noisy data streams. As discussed earlier, compared to its peers, AE combines much more base classifiers to build an ensemble predictor. This raises the concern on the efficiency of AE due to its additional cost for training extra base classifiers.

To study AE's time complexity, let's consider the following example. Assume the buffer of the system contains d data chunks, each of which contains N instances. Assume further that m learning algorithms are used to build models. Each time when a new data chunk arrives, we need to follow two steps to update an ensemble: (1) build new base classifier(s) on the new data chunk; and (2) update classifier ensemble by incorporating new base classifier(s). Without loss of generality, we assume that training a new base classifier needs $O(N \lg N)$ time on average, while updating the classifier ensemble to include one base classifier requires $O(\Gamma)$ time, where Γ is related to the dimensionality of attributes. Then the updating of the HE ensemble for each new data chunk needs to (1) build a new base classifier (which costs $O(N \lg N)$ time), and then (2) combine the most recent d base classifiers (which costs $O(d\Gamma)$ time) together for prediction. The total time cost can be calculated by Eq. (10),

$$O(HE) = O(N \lg N) + O(d\Gamma). \tag{10}$$

Since training a base classifier dominates the total cost (*i.e.*, $O(\Gamma) \ll O(N \lg N)$), and the number of data chunks d in the buffer is rather small. The time complexity of the HE ensemble can be simplified as in Eq. (11),

$$O(HE) = O(N \lg N) + O(d\Gamma) = O(N \lg N). \tag{11}$$

In comparison, VE builds m base classifiers for each new data chunk. Accordingly, its time complexity $O(\mathrm{VE})$ can be calculated by Eq. (12),

$$O(VE) = O(m) * (O(N \lg N) + O(\Gamma)) = O(mN \lg N). \tag{12}$$

For AE, it first builds m classifiers when a new data chunk arrives, and combines all the d^*m base classifiers to build an ensemble. So its time complexity can be calculated by Eq. (13),

$$O(AE) = O(mN \lg N) + O(dm\Gamma) = O(mN \lg N). \tag{13}$$

Combining Eqs. (11), (12), and (13), we have the following two conclusions: (1) AE is, asymptotically, as efficient as VE. Both of them have the same time complexity $O(mN \lg N)$; (2) AE requires more time complexity than HE because AE needs to train m base classifiers for each new data chunk. This limitation, in practice, can be alleviated by using a multi-core or multi-processor computing system, where base classifiers can be dispatched and trained on different computation units in parallel.

5. Experiments

To evaluate the performance of AE, we carry out experimental studies on both synthetic and real-world data streams, by implementing all algorithms in Java and the WEKA [42] data mining package. Unless specified otherwise, we use Decision Tree (Tree) [33], Logistic Regression (LR) classifier, and Libsvm (SVM) [11] to build AE. All tests are carried out on a PC machine with a 1.7 G CPU and 2 GB memory.

5.1. Assessment criteria

For ease of comparisons, we first summarize the assessment criteria of the ensemble-based data stream mining models. Due to the importance of prediction accuracy in assessing a classification model, many existing ensemble-based models [34,35,45,48] compare the average prediction accuracy to its peers. Recently, Gao et al. [18] proposed to provide chunk-by-chunk comparisons and proposed several other measurements such as the average ranking (AR), number of wins (#W) and loses (#L). On the other hand, considering that a good ensemble classifier should have high prediction accuracies and low computational overhead, Wang et al. [38] evaluated their method with respect to both the prediction accuracy and system training time. Similar work can be found in many other data stream classification methods [17,19,23,34,35]. In our experiments, we first compare the ensemble-based models with respect to the prediction accuracy on a synthetic data stream and the real-world KDDCUP'99

network intrusion data set. We then compare the models on another real-world wireless sensor data stream with respect to both prediction accuracy and the system runtime performance.

Three assessment criteria employed in our experimental study are as follows. Suppose a data stream has n data chunks, $D_1, D_2, \ldots, D_n$. We aim to build a classifier to predict all instances' labels in the yet-to-come chunk D_{i+1}. Consider an instance x in D_{i+1}, if the predicted class label of x is the same as the label of x with the highest posterior probability, we regard x as a correctly classified instance. Accordingly, we define *Accuracy* (*acc*) as the percentage of the number of correctly classified instances in D_{i+1}. Furthermore, we rank all algorithms in an order from 1 to 5 according to their accuracies, with the most accurate algorithm ranked as 1 and the least accurate algorithm ranked as 5. In the case that two classifiers or more have the same accuracy, we will assign the same ranking order to them. In addition, we define the other two measures, *number of wins* (#W) and *losses* (#L) as follows: if a classifier is ranked as 1, then we increase its #W by 1; on the contrary, if a classifier is ranked as 5, we add its #L by 1. Following the above process for $n-1$ times, we have *the average accuracy* (*Aacc*), *average ranking* (*AR*), *standard deviation of the ranks* (*SR*), and *the total numbers of* #W and #L for all the algorithms. Ideally, a good classifier should have a high prediction accuracy, a ranking order close to 1, a large #W, a small #L, and a small SR value.

5.2. Experiments on synthetic data streams

We create synthetic data streams using Matlab 7.0 as follows. Firstly, we generate instances x_t at time stamp t by using a Gaussian distribution $x_t \sim N(\mu_t, \Sigma_t)$, where μ_t denotes the distribution center and Σ_t is the covariance matrix (in our experiments, each instance has two dimensions, with μ_0 starting from $[\pi,\pi]$ and $\Sigma_t = \begin{bmatrix} \pi, 0 \\ 0, \pi \end{bmatrix}$ for each time stamp t). Then we define the potential pattern $p(y|x)$ at time stamp t as follows,

$$y_t = \frac{1}{r}\sum_{i=1}^{r} a_t \, sin(x_t) + \frac{1}{r}\sum_{i=1}^{r} b_t x_t^2 + \varepsilon \tag{14}$$

where r is the number of dimensions, *at* and *bt* are r-dimensional vectors. The first two nonlinear terms are used to generate a complex nonlinear classification boundary. To simulate real-world data streams, we generate stream data containing both concept drifting and noise, where ε accounts for noise with a Gaussian distribution $\varepsilon \sim N(0, 0.3^2)$. To simulate the concept drifting, we let $p(x,y)$ change randomly. Since $p(x,y) = p(x) \cdot p(y|x)$, changing $p(x,y)$ is equivalent to changing $p(x)$ or $p(y|x)$. To evolve $p(x)$, we let x's distribution center μt vary with time as $\mu_{t+1} = \mu_t + (-1)^s d$, where s denotes the direction (which has 10% of chance to reverse its direction), and d denotes the step length (which is set to be 0.1). To evolve $p(y|x)$, we let b_t have 50% of chance to become $b_{t+1} = b_t + 1$. On the other hand, to add noise into the data, we let y_t have 20% of chance to change its label. For two-class classification at time stamp t, the decision boundary is set using the rules that, if $y_t \geq \frac{1}{r}\sum_i b_t x_t^2$, the class label is "+1"; otherwise the class label is "−1". For multi-class classification, supposing there are l classes $\{c_1, c_2, \ldots, c_l\}$, we can assign class labels by equally dividing y_t into l parts. For better understanding, major notations used to generate our data streams are summarized in Table 1.

Table 1
Major parameters used to generate synthetic data streams.

Variable	Description
r	Number of attributes
x_t	Example generated at time stamp t
y_t	Class label of example x_t
a_t, b_t	Coefficient vectors for generating label y_t
ε	Noise
μ_t	Distribution center of x_t
Σ_t	Covariant matrix of x_t
s	Controls concept drifting direction
d	Controls concept drifting step length

In Tables 2 and 3, we report the experimental results for binary and multi-class streams, where the first row in the table shows the number of chunks (N) and the chunk size (B). From the results in Tables 2 and 3, it is clear that among the five stream mining algorithms, AE always has the highest average prediction accuracy, the best ranking, and the least number of losses. When comparing five algorithms based on their average accuracies, we can observe that VE follows AE as the second best method, HE and WE have the same accuracy, both of them are listed as the third best methods together, and the single tree is the least accurate method. Accordingly, the order of the average prediction accuracies suggests the ranking of all methods as: $Aacc_{AE} > Aacc_{VE} > Aacc_{WE} = Aacc_{HE} > Aacc_{Tree}$. When considering the ranking measures (AR and SR), we find that AE is listed at the first place, followed by VE, HE and WE, and the single decision tree, respectively. As for the standard deviation of the ranking orders, we find that HE and WE have the minimal AR. When comparing AE with HE, we can observe that AE is much more stable than HE. The single tree is ranked at the bottom with the most unstable ranking ($AR_{AE} < AR_{VE} < AR_{HE} = AR_{WE} < AR_{Tree}$). When considering the measures #W and #L, we find that VE always has the most frequent wining chance, while AE follows after VE as the second, HE and WE have the same winning chance, and the single tree has the smallest chance of winning. Based on the above observations, we can conclude that: among the five algorithms, AE performs the best, VE performs the second best, HE and WE are considered the third tier with a tie, and the single tree is the least accurate method for stream data.

In summary, the above observations suggest the following conclusions: (1) Using the same base learners, HE and WE appear to perform similarly, and including weight values to each base classifier does not seem to be very helpful; (2) HE and WE mostly have the least average ranking, and they are consistently ranked inferior to AE and VE, but superior to the single tree; (3) VE always has the best winning chance, whereas AE always has the least chance to lose; and (4) compared to other four methods, the single decision tree is the least accurate method for stream data mining, which has the lowest accuracy, lowest ranking, minimal winning chance and maximum chance to lose.

Intuitively, we suspect that AE should perform the best for all the measures, whereas, in practice, VE appears to have a better chance of

Table 2
Binary data stream, $p(x,y)$ evolves with 20% noisy data chunks.

Measure	N = 100, B = 100					N = 1000, B = 100				
	Tree	HE	WE	VE	AE	Tree	HE	WE	VE	AE
Aacc	0.572	0.575	0.575	0.596	**0.614**	0.680	0.679	0.679	0.701	**0.704**
AR	3.192	2.697	2.697	2.505	**2.495**	3.323	3.010	3.010	2.392	**2.194**
SR	0.033	**0.001**	**0.001**	0.023	0.003	0.028	**0.000**	**0.000**	**0.000**	0.020
#W	9	14	14	**45**	33	8	9	9	**54**	35
#L	33	31	31	26	**20**	30	37	37	23	**12**

Table 3
Multi-class data stream, $p(x,y)$ evolves with 20% noisy data chunks.

Measure	N = 100, B = 100					N = 1000, B = 100				
	Tree	HE	WE	VE	AE	Tree	HE	WE	VE	AE
Aacc	0.401	0.422	0.422	0.476	**0.485**	0.572	0.616	0.616	0.602	**0.646**
AR	3.768	2.849	2.849	2.081	**2.071**	3.091	3.030	3.030	2.404	**2.394**
SR	**0.006**	0.013	0.013	0.012	0.012	0.037	**0.000**	**0.000**	0.020	0.002
#W	7	12	12	**48**	39	8	10	10	**53**	29
#L	51	36	36	15	**11**	15	48	48	26	**9**

winning (#W) than AE. This is because VE is suitable for the dynamic description, which only considers concept drifting but no data errors. In our synthetic data stream, we generate 20% noisy data chunks, whereas the remaining 80% data chunks are clean. Consequently, VE has a much better winning chance. However, even if VE has a better winning chance than AE, VE is still inferior to AE because VE will suffer from the decreasing of the prediction accuracy when building model on noisy chunks (as shown in Table 5, which will be discussed shortly).

To investigate the situations where concept drifting and noise interruption occur simultaneously, we report the accuracies across 100 data chunks in Fig. 7. We can observe that there is always a significant drop in the accuracy once a noisy data chunk emerges. To study the reasons behind, we take two typical data chunks as an example: chunk 6, a normal chunk followed by a noisy chunk 7; and chunk 7, a noisy chunk followed by a normal chunk 8. As shown in Table 4, when using the normal chunk (chunk 6) to predict the noisy chunk (chunk 7), all five methods receive poor performance, which explains the "sudden drop" in Fig. 7. In fact, none of the five methods are able to predict a noisy chunk with a high accuracy, so there is always a sudden decrease of the accuracy. Table 5 shows the second typical situation that a noisy up-to-date chunk 7 is followed by a normal yet-to-come data chunk 8. We can observe that AE has the best performance. This is because in addition to the current noisy chunk 7, AE still uses other normal chunks in the buffer, *i.e.*, the 5th and 6th chunks to predict the 8th chunk (similar to HE and WE), but VE only depends on the 7th noisy chunk (similar to the single tree), so AE, HE and WE, which have used historical data chunks in the buffer, will perform better than VE and the single tree. In summary, classifiers built on a single data chunk may suffer significant loss in prediction accuracies for noisy chunks, and this explains why they are not suitable for realistic data streams. On the other hand, classifiers built on several consecutive data chunks may preserve valuable information, which can help reduce the negative impact of the noisy chunks.

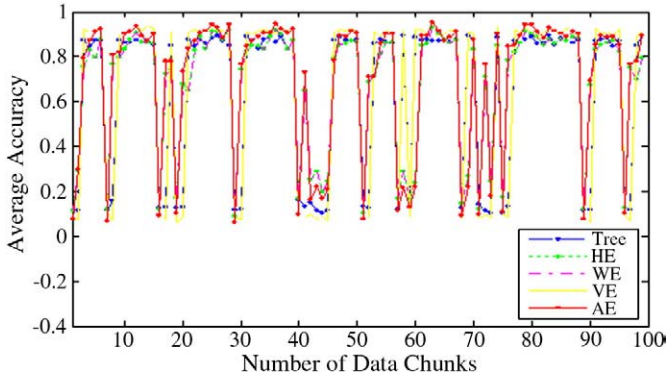

Fig. 7. The two-group synthetic data stream, each chunk has 1000 instances; there are total 100 data chunks.

5.3. Experiments on KDDCUP'99 data stream

In this subsection, we compare all ensemble methods on the KDDCUP'99 intrusion detection dataset, which is a popularly used test bed for stream data mining [5]. Since many research works have reported that concepts underlying this dataset appear to be linearly separable (the average prediction accuracy is over 97% on 10% sampled instances), we complicate the learning task by using the following four approaches to build different types of data streams: (1) random selection — we randomly select 100 data chunks, each of which contains 1000 instances with an equal class distribution (50% of instances in each class); (2) random noisy selection — we randomly select 20% data chunks from (1), and then arbitrarily assign each instance a class label which does not equal its original class label, and finally we put these noisy data chunks back into the stream; (3) rearranged selection — given a training set, we first find the most informative attribute by using the information gain [22] measure (*i.e.*, the 30th attribute), then we sort all instances according to the values on this attribute, and the sorted instances are finally put into 100 data chunks each of which contains 1000 instances; and (4) rearranged noisy selection — we add 20% noisy data chunks in Eq. (3) in a similar way to the procedure in Eq. (2). Major notations of the parameters of this data set are listed in Table 6.

Table 5 lists the results of a random selection of the KDDCUP'99 dataset. We can observe that AE performs the best in terms of the average prediction accuracy (0.995), AR (average ranking 1.475), #W (73 wins) and #L (11 loses). Taking all evaluation criteria into consideration, VE, with an average accuracy of 0.994, 62 wins and 16 loses is the second best method. HE and WE perform the same, with an average accuracy of 0.993, 34 wins and 39 loses. The single decision tree, with an average performance of 0.991, and 21 wins and 70 loses, is the least preferred method.

Table 7 lists the comparison results on a random selection schema. It validates our hypothesis that under realistic data stream scenarios, the weighted ensemble does not have any difference from the horizontal ensemble, and the order of accuracy is still $Aacc_{AE} > Aacc_{AE} > Aacc_{VE} > Aacc_{HE} = Aacc_{WE} > Aacc_{Tree}$. Since random selection selects data chunks from the raw data set without any revisions, we can regard it as a realistic data stream. It is safe to say that AE performs the best on this realistic data stream.

Table 8 reports a random selection with 20% noise and we can observe that AE also performs the best, with the highest average accuracy and ranking, the most winning and least losing chances. The order of accuracy is $Aacc_{AE} > Aacc_{HE} = Aacc_{WE} > Aacc_{VE} > Aacc_{Tree}$. We can see that VE and the single tree are more vulnerable to noise. Compared with Table 5, the accuracy of VE and the single tree

Table 4
Using the 6th normal data chunk to predict the 7th noisy data chunk.

Alg.	Tree	HE	WE	VE	AE
Acc.	0.118	0.123	0.123	0.100	**0.070**

Table 5
Using the 7th noisy data chunk to predict the 8th normal data chunk.

Alg.	Tree	HE	WE	VE	AE
Acc.	0.158	0.767	0.767	0.071	**0.808**

significantly drops, while AE, HE and WE marginally drops. This tells us that buffering a small number of data chunks can prevent a significant drop of accuracy caused by noise.

Table 9 lists the results of a rearranged method, and we can observe that VE performs the best on all of the five measurements. It has the largest average accuracy, the smallest AR, the most winning chance and the least losing chance. The single tree is the second best method. AE is the third best, and HE and WE are listed as the last. The rearrangement procedure, in fact, generates a special data stream according to the dynamic description. So the classifier ensemble built on the most recent data chunk is better than the classifier ensemble built with several buffered data chunks. That is why VE and the single tree perform better than others.

Table 10 reports the results of a rearranged noise selection method, with 20% noise in addition to the change of $p(x,y)$. This is the most difficult situation. We can observe that all the five algorithms suffer a significant drop of accuracy. VE drops the most, from 0.926 to 0.670, the single decision tree drops from 0.911 to 0.669, HE and WE drops from 0.825 to 0.676, and AE drops from 0.879 to 0.682. Among them, AE achieves the best. This is because the rearranged noise selection method generates a data stream that experiences concept drifting and data errors simultaneously, and AE, as we discussed earlier, performs the best under this circumstance.

5.4. Experiments on wireless sensor stream

In this subsection, we carry out experiments and comparisons on a real-world wireless sensor data stream, which is publically available (http://www.cse.fau.edu/~xqzhu/stream.html) and popularly used as the test bed for data stream mining models. The purpose is to address the following two concerns: (1) Whether AE performs better than all its individual learning algorithms, as well as the HE trained using different learning algorithms as the base learners? and (2) How many learning algorithms should be used in AE? To answer the above two questions, we first introduce the wireless sensor stream, and then report the algorithm performance in terms of average prediction accuracy and the system runtime.

The wireless sensor stream contains information (temperature, humidity, light, and sensor voltage) collected from 54 sensors deployed in a lab environment. The data are read every 1–3 min from all sensors, and the whole stream contains information recorded over a two month period. The learning task is to correctly identify the sensor ID (1 out of 54 sensors) purely based on the reading of the sensor data and the recording time. It should be noticed that the concept underlying the sensor stream may change with time. For example, the lighting during the working hours is generally stronger than that in the night, and the temperature of specific sensors (*i.e.*, sensors in a conference room) may suddenly rise during the meeting time. In addition, the wireless sensor stream may also contain random errors. For example, when communication channels of sensor nodes are blocked by moving objects, or the sensor node's hardware may experience malfunction, the data stream generated from the sensor node may contain erroneous or missing values. In our experiments, for simplicity, the stream is transferred into a binary-class classification problem by splitting the 54 IDs into two classes (*i.e.*, if a sensor's ID is less than 28, then it belongs to class "−1"; otherwise, it belongs to class "+1"). Besides, we split this data stream into data chunks, with each chunk containing 100 examples.

Table 6
A list of parameters in the KDDCUP'99 data set.

Variable	Description
Random selection	We randomly pick 100 data chunks, each data chunk containing 1000 examples with balance class labels
Random noisy selection	Based on the random selection, we again randomly choose 20% data chunks and assign wrong class labels to every example in this chunks
Rearranged selection	Based on the random selection, we sort all the examples by the 30th attribute, and then the sorted examples are put into 100 data chunks
Rearranged noisy selection	Based on the rearranged selection, we randomly choose 20% data chunks as noisy data chunks

Table 7
Random selection results.

	Tree	HE	WE	VE	AE
Aacc	0.991	0.993	0.993	0.994	**0.995**
AR	3.232	2.192	2.192	1.778	**1.475**
SR	0.015	0.000	0.000	0.006	0.126
#W	21	34	34	62	**73**
#L	70	39	39	16	**11**

In Fig. 8(a), we report the experimental comparisons among AE, AE's component algorithms, and HE trained with different learning algorithms. For example, the symbol "HE(SVM)" denotes that HE is trained with SVM. From the results, we can observe that compared to its individual classifier and HE trained with different learning algorithms, AE has the best prediction accuracy on average. This is because when the data distribution of the yet-to-come test chunk is unknown, we cannot always find an "optimal" algorithm with the best prediction accuracies across the stream. As a result, combining different learning algorithms is likely to reduce the average prediction error.

In Fig. 8(b), we report the system training time comparisons across different methods. Not surprisingly, AE is the most time-consuming method among all benchmark approaches, mainly because AE has to train more base classifiers than its peers. To reduce AE's runtime, a possible solution is to employ multi-core or multi-processor computing systems to train base classifiers in parallel.

To test AE's performance under different numbers of learning algorithms, we exam AE's performance by using six well-known learning algorithms, including the *Decision Tree* (*Tree*), *Logistic Regression* (*LR*), *SVM*, *NaïveBayes*, *K-NN*, and *Multiple Perceptron*, as the base learners. For ease of description, we use symbol "+*M*" to denote that an extra learning algorithm *M* is added to AE. For example, "+LR" after "Tree" means that in addition to the Decision Trees, we also add a Logistic Regression model as AE's base learning algorithms. From Fig. 9(a), we can observe that when we use Tree, LR, and SVM as base classifiers, the prediction accuracy increases continuously. After that, the prediction accuracy fluctuates with the inclusion of additional learning algorithms. For example, adding NaïveBayes and Multiple Perceptron actually reduce AE's performance. Under this observation, we adjust the order of the learning algorithms and report the new results in Fig. 9(b). We can observe that AE's performance is

Table 8
Random selection with noise.

	Tree	HE	WE	VE	AE
Aacc	0.694	0.822	0.822	0.695	**0.823**
AR	2.929	2.263	2.263	2.182	**2.121**
SR	0.009	0.001	0.001	**0.000**	0.013
#W	20	29	29	**54**	**54**
#L	45	35	35	29	**26**

Table 9
Rearranged selection results.

	Tree	HE	WE	VE	AE
Aacc	0.911	0.825	0.825	**0.926**	0.879
AR	1.525	2.000	2.000	**1.475**	2.267
SR	0.022	0.010	0.010	**0.002**	0.016
#W	**74**	58	58	**74**	59
#L	17	35	35	**8**	26

still unstable with different numbers of learning algorithms. In other words, there does not seem have a single "optimal" number of classifiers for AE. The number of based classifiers should be used in AE may vary, depending on the data characteristics of the underlying data streams. For example, when the concept drifts marginally, a practical solution is to combine several strong algorithms (such as the LR, SVM, and KNN) together to construct the AE framework.

6. Conclusions

Data errors pose a great challenge to data mining models. Such a challenge becomes much more severe in dynamic data stream environments where the erroneous data may mix with the concept drifting problem. In order to build accurate prediction models from noisy data streams, existing solutions largely rely on some data preprocessing algorithms to cleanse noise from data streams, such that the cleansed stream data can be used to build accurate prediction models. Nevertheless, all existing stream data preprocessing models share two disadvantages. First, most of them implicitly make some statistical assumptions, through which noisy data can be differentiated from data of drifting concepts. In practice, such statistical assumptions may not hold in many real-world applications. Second, most existing algorithms require multiple scanning to first cleanse a noisy data stream and then build models from the cleansed data. Such a multi-scan manner may not be appropriate for fast flowing data streams. Alternatively, in this paper, we proposed a robust aggregate ensemble (AE) learning model to assist the knowledge discovery for noisy data streams. AE first trains base classifiers using different learning algorithms on different data chunks, and then combines all the base classifiers to form an ensemble classifier through model averaging. By doing so, AE is capable of handling the concept drifting challenge, as well as tolerating the data errors. Theoretical and empirical studies demonstrated that AE is superior to existing ensemble-based models, such as the horizontal ensemble, the weighted ensemble, and the vertical ensemble models, for noisy data streams.

Acknowledgements

This research was partially supported by the National Science Foundation of China (NSFC) grants (61003167, 60828005, 70621001, and 70921061), a China 973 Project (2007CB311100), a Chinese Academy of Sciences grant (Overseas Collaboration Group), a US National Science Foundation grant (CCF-0905337), and an Australian ARC grant (DP1093762).

Table 10
Rearranged selection with noise.

	Tree	HE	WE	VE	AE
Aacc	0.669	0.676	0.676	0.670	**0.682**
AR	2.424	1.899	1.8990	1.838	**1.737**
SR	0.025	0.008	0.0082	0.047	**0.006**
#W	54	54	54	62	**65**
#L	30	33	33	27	**20**

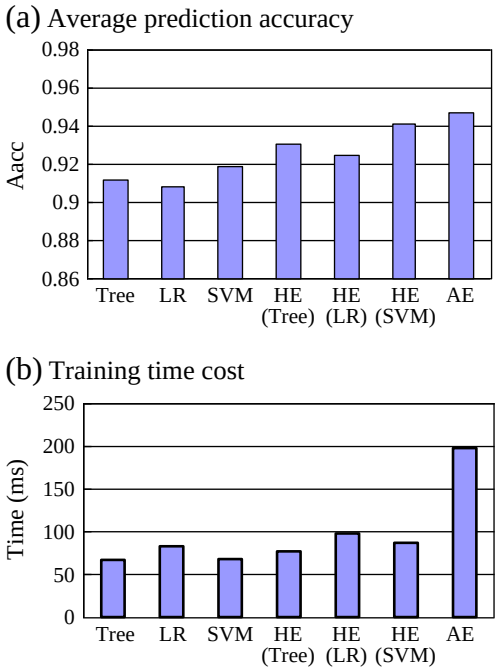

Fig. 8. Comparison results with respect to (a) average accuracy, and (b) system runtime. The wireless sensor stream is divided into successive data chunks, with each chunk containing 100 data records. Besides, both HE and AE use the latest three data chunks to build up the ensemble framework.

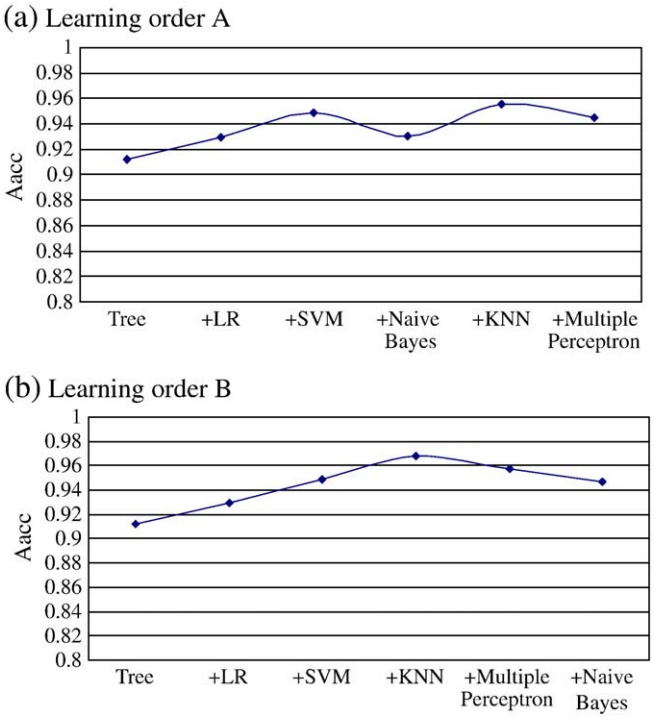

Fig. 9. The results of applying different numbers of learning algorithms to AE on the wireless sensor stream data. The data stream is split into continuous data chunks, with each chunk having 100 data records. AE uses the most recent three data chunks to construct the ensemble framework. It is obvious that the prediction accuracy does not always improve with the number of learning algorithms increase. So finding the "optimal" number of *m* in AE is essentially difficult, if not impossible. When the concept drifts marginally, a practical solution is to combine several learning algorithms (such as the LR, SVM, and KNN) to construct AE.

References

[1] C. Aggarwal, On classification and segmentation of massive audio data streams, Knowledge and Information Systems: An International Journal 20 (2) (2009) 137–156.
[2] C. Aggarwal, J. Han, J. Wang, Y. Philip, A framework for clustering evolving data streams, Proc. of VLDB (2003) 81–92.
[3] D. Aha, D. Kibler, M. Albert, Instance-based learning algorithms, Machine Learning (1991) 37–66.
[4] K. Ali, M. Pazzani, Error reduction through learning multiple descriptions, Machine Learning (1996).
[5] A. Asuncion, D. Newman, UCI Machine Learning Repository, Irvine, CA, 2007.
[6] R. Avnur, J. Hellerstein, Eddies: Continuously Adaptive Query Processing, Proc. of SIGMOD (2000) 261–272.
[7] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data streams, Proc. of PODS (2002) 1–16.
[8] S. Babu, J. Widom, Continuous queries over data streams, ACM SIGMOD Record 30 (3) (2001) 109–120.
[9] D. Barbara, The New Jersey data reduction report, IEEE Data Engineering Bulletin 20 (4) (1997) 3–45.
[10] C. Brodley, M. Friedl, Identifying Mislabeled Training Data, Journal of Artificial Intelligence Research 11 (1999) 131–167.
[11] C. Chang, and C. Lin, LIBSVM Toolbox, Available online: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.
[12] Y. Chen, L. Tu, Density-based clustering for real-time stream data, Proc. of KDD (2007) 133–142.
[13] Y. Chi, H. Wang, P. Yu, R. Muntz, Moment, Maintaining closed frequent itemsets over a stream sliding window data streams, Proc. of IEEE ICDM (2004) 59–66.
[14] F. Chu, Y. Wang, C. Zaniolo, An Adaptive Learning Approach for Noisy Data Streams, Proc. of IEEE ICDM (2004) 351–354.
[15] T. Dietterich, Ensemble methods in machine learning, Proc. of the first Workshop on Multiple Classifier Systems (2000) 1–15.
[16] P. Domingos, G. Hulten, Mining high-speed data streams, Proc. of KDD (2000) 71–80.
[17] W. Fan, Systematic data selection to mine concept-drifting data streams, Proc. of KDD (2004) 128–137.
[18] J. Gao, W. Fan, J. Han, On appropriate assumptions to mine data streams: Analysis and Practice, Proc. of IEEE ICDM (2007) 143–152.
[19] M. Gaber, P. Yu, Detection and Classification of Changes in Evolving Data Streams, International Journal of Information Technology and Decision Making (IJITDM) 5 (4) (2006) 659–670.
[20] S. Guha, A. Meyerson, N. Mishra, R. Motwani, Clustering Data Streams: Theory and Practice, IEEE Transactions on Knowledge and Data Engineering 15 (3) (2003) 515–528.
[21] T. Ho, J. Hull, S. Srihari, Decision combination in multiple classifier systems, IEEE Transactions on PAMI 16 (1) (1994) 66–75.
[22] P. Hore, L. Hall, D. Goldgof, A Fuzzy C Means Variant For Clustering Evolving Data Streams, Proc. of IEEE International Conference on Systems, Man and Cybernetics (2007) 360–365.
[23] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, Proc. of KDD (2001) 97–106.
[24] J. Kolter, M. Maloof, Using additive expert ensembles to cope with concept drift, Proc. of ICML (2005) 449–456.
[25] S. Laxman, P. Sastry, K. Unnikrishnan, A fast algorithm for finding frequent episodes in event streams, Proc. of KDD (2007) 410–419.
[26] H. Li, et al., Efficient Maintenance and Mining of Frequent Itemsets over Online Data Streams with a Sliding Window, Proc. of IEEE International Conference on Systems, Man and Cybernetics (2006) 2672–2677.
[27] G. Luo, K. Wu, P. Yu, Answering linear optimization queries with an approximate stream index, Knowledge and Information Systems: An International Journal 20 (1) (2009) 95–121.
[28] G. Manku, R. Motwani, Approximate frequency counts over data streams, Proc. of VLDB (2002) 346–357.
[29] M. Mannino, Y. Yang, Y. Ryu, Classification Algorithm Sensitivity to Training Data with Non Representative Attribute Noise, Decision Support Systems 46 (3) (2009) 743–751.
[30] D. Olson, Y. Shi, Introduction to Business Data Mining, McGraw-Hill/Irwin, 2005.
[31] S. Pang, S. Ozawa, N. Kasabov, Incremental Linear Discriminant Analysis for Classification of Data Streams, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 35 (5) (2005) 905–914.
[32] J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.
[33] J. Quinlan, The Effect of Noise on Concept Learning, Machine Learning (1986).
[34] M. Scholz, R. Klinkenberg, An ensemble classifier for drifting concepts, Proc. of ECML/PKDD Workshop on Knowledge Discovery in Data Streams (2005) 53–64.
[35] W. Street, Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, Proc. of KDD (2001) 377–382.
[36] K. Su, H. Huang, X. Wu, S. Zhang, A logical Framework for Identifying Quality Knowledge from Different Data Sources, Decision Support Systems 42 (3) (2006) 1673–1683.
[37] N. Syed, H. Liu, K. Sung, Handling concept drifts in incremental learning with support vector machines, Proc. of KDD (1999) 317–321.
[38] H. Wang, W. Fan, P. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, Proc. of KDD (2003) 226–235.
[39] X. Wang, H. Liu, J. Han, Finding Frequent Items in Data Streams Using Hierarchical Information, Proc. of IEEE International Conference on Systems, Man and Cybernetics (2007) 431–436.
[40] Y. Wang, Z. Li, Y. Zhang, Classifying Noisy Data Streams, Fuzzy Systems and Knowledge Discovery (2006) 548–549.
[41] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, Machine Learning 23 (1996) 69–101.
[42] I. Witten, E. Frank, Data mining: practical machine learning tools and techniques, Morgan Kaufmann (2005).
[43] Y. Yang, X. Wu, X. Zhu, Combining proactive and reactive predictions of data streams, Proc. of KDD (2005) 710–715.
[44] P. Zhang, X. Zhu, L. Guo, Mining Data Streams with Labeled and Unlabeled Training Examples, Proc. of IEEE ICDM (2009) 627–636.
[45] P. Zhang, X. Zhu, Y. Shi, Categorizing and Mining Concept Drifting Data Streams, Proc. of KDD (2008) 820–821.
[46] P. Zhang, X. Zhu, Y. Shi, X. Wu, An Aggregate Ensemble for Mining Concept Drifting Data, Proc. of PAKDD (2009) 1021–1029.
[47] D. Zhu, A Hybrid Approach for Efficient Ensembles, Decision Support Systems 48 (3) (2010) 480–487.
[48] X. Zhu, P. Zhang, X. Lin, Y. Shi, Active Learning from Stream Data Using Optimal Weight Classifier Ensemble, IEEE Transactions on System, Man, Cybernetics, Part B 40 (4) (2010) 1–15.
[49] X. Zhu, X. Wu, Class Noise vs Attribute Noise: A Quantitative Study, Artificial Intelligence Review 22 (3) (2004) 177–210.
[50] X. Zhu, X. Wu, C. Zhang, Cleansing Noisy Data Streams, Proc. of IEEE ICDM (2008) 1139–1144.

Peng Zhang is an Assistant Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing (China). He received his Ph.D. (2009) in computer science from the Graduate University of the Chinese Academy of Sciences, Beijing, China. His research interests include data stream mining, information filtering, and information security.

Xingquan Zhu is an Associate Professor with the Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Sydney, Australia. He has been an Associate Editor of the IEEE Transactions on Knowledge and Data Engineering (TKDE) since 2009.

Yong Shi is the Charles W. and Margre H. Durham Distinguished Professor of Information Technology, College of Information Science and Technology, Peter Kiewit Institute, University of Nebraska, USA. He is also the Executive Deputy Director of the Fictitious Economy and Data Science Research Center, Chinese Academy of Sciences, Beijing, China. He is the Editor-in-Chief of International Journal of Information Technology and Decision Making (SCI), an Area Editor of International Journal of Operations and Quantitative Management, a member of Editorial Board for a number of academic journals, including International Journal of Data Mining and Business Intelligence.

Li Guo is the director of the Information Security Research Center, Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include data stream management and information security.

Xindong Wu is a Yangtze River Scholar in the School of Computer Science and Information Engineering at the Hefei University of Technology (China), and a Professor of Computer Science at the University of Vermont (USA). He is the Editor-in-Chief of Knowledge and Information Systems (KAIS). He was the Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering (TKDE, by the IEEE Computer Society) between January 1, 2005 and December 31, 2008.