# Online Ensemble Learning of Data Streams with Gradually Evolved Classes

Yu Sun, *Student Member, IEEE*, Ke Tang, *Senior Member, IEEE*, Leandro L. Minku, *Member, IEEE*, Shuo Wang, *Member, IEEE*, and Xin Yao, *Fellow, IEEE*

**Abstract**—Class evolution, the phenomenon of class emergence and disappearance, is an important research topic for data stream mining. All previous studies implicitly regard class evolution as a transient change, which is not true for many real-world problems. This paper concerns the scenario where classes emerge or disappear gradually. A class-based ensemble approach, namely Class-Based ensemble for Class Evolution (CBCE), is proposed. By maintaining a base learner for each class and dynamically updating the base learners with new data, CBCE can rapidly adjust to class evolution. A novel under-sampling method for the base learners is also proposed to handle the dynamic class-imbalance problem caused by the gradual evolution of classes. Empirical studies demonstrate the effectiveness of CBCE in various class evolution scenarios in comparison to existing class evolution adaptation methods.

**Index Terms**—Data stream mining, class evolution, ensemble model, on-line learning, imbalanced classification

## 1 Introduction

With the rapid development of incremental learning and online learning, mining tasks in the context of data stream have been widely studied [1], [2]. Generally, data stream mining refers to the mining tasks that are conducted on a (possibly infinite) sequence of rapidly arriving data records. As the environment where the data are collected may change dynamically, the data distribution may also change accordingly. This phenomenon, referred to as concept drift [3], [4], is one of the most important challenges in data stream mining. A data stream mining technique should be capable of constructing and dynamically updating a model in order to learn dynamic changes of data distributions, i.e., to track the concept drift.

For classification problems, concept drift is formally defined as the change of joint distribution of data, i.e., $p(\mathbf{x}, y)$, where $\mathbf{x}$ is the feature vector and $y$ is the class label. Over the past few decades, concept drift has been widely studied [5], [6], [7]. The majority of the previous works focus on the concept drift caused by the change in class-conditional probability distribution, i.e., $p(\mathbf{x}|y)$. In comparison, *class evolution*, which is another factor that induces concept drift, has attracted relatively less attention. Briefly speaking, class evolution is concerned with certain types of change in the prior probability distribution of classes, i.e., $p(y)$, and usually corresponds to the emergence of a novel class and the disappearance of an outdated class. Class evolution occurs frequently in practice. For example, new topics frequently appear on *Twitter* and outdated topics are forgotten with time. Besides, old topics, e.g., topics on festivals, may also become popular again. Such phenomena can also be observed from other types of data streams, such as the click-through data of news or advertisements since the interests of clients may change over time. In some literature, class evolution is also called class-incremental learning [8] or concept evolution [9], [10], [11]. More formally, let $C_t$ denote the set of classes whose prior probability is positive at time stamp $t$. Class evolution involves the following forms:

- Class emergence represents an example of an unknown class is received at the current time. That is, class $c$ emerges at time $t$ if $c \notin C_1 \cup C_2 \cup \cdots \cup C_{t-1}$ and $c \in C_t$. Such a class is called a novel class.
- Class disappearance describes the situation in which the example of an existing class would not be received in the next time stamp. That is, if class $c$ disappeared at time $t$, then $c \in C_{t-1}$ and $c \notin C_t$.
- Class reoccurrence defines the point where a disappeared class recurs later in the data stream. Class $c$ is a recurring class at time $t$, if $c \in C_1 \cup \cdots \cup C_{d-1}$, $c \notin C_d \cup \cdots \cup C_{t-1}$, and $c \in C_t$.

Since the number of classes may change when class evolution happens, the model needs to be adapted not only to capture the distribution of existing classes, but also to identify that of the novel classes. At the same time, the effects of disappeared classes need to be removed from the model. Hence, in comparison to the change of class-conditional probability, class evolution brings additional challenges to data stream mining.

In literature, a few approaches have been proposed to address class evolution problems, e.g., Learn$^{++}$. NC [12], ECSMiner [13] and CLAM [14]. Although they have shown

- *Y. Sun and K. Tang are with the USTC-Birmingham Joint Research Institute in Intelligent Computation and its Applications, School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China. E-mail: sunyu123@mail.ustc.edu.cn, ketang@ustc.edu.cn.*
- *L.L. Minku is with the Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom. E-mail: leandro.minku@leicester.ac.uk.*
- *S. Wang and X. Yao are with the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom. E-mail: {s.wang, x.yao}@cs.bham.ac.uk.*

*Manuscript received 23 Jan. 2015; revised 31 Dec. 2015; accepted 27 Jan. 2016. Date of publication 8 Feb. 2016; date of current version 27 Apr. 2016. Recommended for acceptance by M. Spiliopoulou.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TKDE.2016.2526675*

promising performance, they implicitly assume that classes emerge or disappear in a transient manner. In other words, the example generation rate (EGR, i.e., the number of examples generated per-unit time) of a class switches between two states, i.e., a constant positive value and zero. However, in a real-world scenario, it is more likely that classes evolve in a gradual manner. For example, in an early stage, an event may be discussed by a few participants on *Twitter*; the topic grows in popularity over a period of time and then eventually fade away from attention. Motivated by this consideration, this work investigates the class evolution problem with gradually evolved classes. Gradual evolution of classes refers to the case that classes appear or disappear in a gradual rather than transient manner, i.e., the EGR changes more smoothly. A novel class-based (CB) ensemble approach, namely Class-Based ensemble for Class Evolution (CBCE), is proposed. In contrast to the above-mentioned existing approaches, which process a data stream in a chunk-by-chunk manner and build a base learner for each chunk, CBCE maintains a base learner for every class that has ever appeared and updates the base learners whenever a new example arrives (i.e., in a one-pass manner). Furthermore, a novel under-sampling method is also designed to cope with the dynamic class-imbalance problem induced by gradual class evolution.

The remainder of this paper is organized as follows: Section 2 presents the problem description and discusses related work. Section 3 presents our adaptation approach. Empirical studies on the proposed as well as the existing approaches are reported in Section 4. Section 5 concludes the paper with directions for future work.

## 2 Problem Description and Related Work

### 2.1 Problem Description

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t), \ldots\}$ denote a data stream, where $\mathbf{x}_t$ and $y_t$ are the example received at time stamp $t$ and its corresponding class label, respectively. Each $\mathbf{x}_t$ is regarded as being generated from the data source of class $y_t$. By these definitions, class evolution is just the evolution of the data sources, i.e., a data source starts or suspends generating example. In gradual class evolution, the example generation rate of a data source changes gradually. That is, the EGR of an evolved class gradually increases from 0 in class emergence (reoccurrence), and decreases from a positive value to 0 in class disappearance. We denote $C_t = \cup_i \{c_i\}$ as the set of classes with positive EGR at time $t$. Furthermore, let $C_t^{novel}$, $C_t^{recurring}$ be the set of novel and recurring classes at time $t$ (i.e., their EGRs are 0 at time $t-1$ and positive at time $t$), respectively. Let $C_t^{disappeared}$ denote the set of the disappeared classes at time $t$ (i.e., their EGRs are positive at time $t-1$ and 0 at time $t$). We have $C_t = C_{t-1} \cup C_t^{novel} \cup C_t^{recurring} - C_t^{disappeared}$. The novel (recurring) classes grow and the outdated classes fade away gradually. Therefore, this leads the underlying class set to be unfixed in the mining process.

### 2.2 Related Work

Since class evolution concerns a special case of concept drift, we will first briefly review the typical strategies for dealing with concept drift [3]. Then, we will proceed with the previous works dedicated to class evolution.

A sliding window method stores in memory a number of the most recent examples; the window size can be fixed [15] or variable [16]. The model is updated based on new data, which are stored in the window. Old data, which tend to be affected by concept drift, are forgotten. In the presence of class evolution, although this method is able to adapt a model to class evolution by dropping previous data, it also forgets potentially useful information of the non-evolved classes, inevitably resulting in a negative impact on the mining performance.

Ensemble methods mainly include chunk-based ensembles, on-line ensembles, and hybrid ones [17]. A chunk-based ensemble constructs each base learner by training it with a different chunk of data [18], [19]. A weighted combination of the base learners is applied to handle the concept drift. In the chunk-based ensemble strategy, class evolution would cause the base learners to have different sets of classes. Taking class emergence as an example, this would cause the collective votes of the earlier base learners to outweigh the correct votes for the novel class [12]. On-line ensembles, e.g., on-line bagging and boosting [20], update each base learner in an on-line manner. This scheme would take a long time for class evolution adaptation. Hybrid ensemble methods aim to combine chunk-based ensembles and on-line ensembles, so as to have the advantages of both in a single framework. For example, the recently proposed AUE2 [21] algorithm employs each chunk of data to initialize a new base learner and to update all existing ones. Then, base learners are weighted according to their accuracies to adapt to the concept drift. Considering class emergence, since the base learner is mainly trained by the non-evolved class, the novel class is highly imbalanced in the existing base learners. Moreover, the examples from the novel classes are not enough in the early stage of gradual class evolution. Hence, it is still difficult to recognize novel class efficiently when class evolution occurs.

Apart from the previous strategies, drift detection methods explicitly determine the drift of concept and update the model accordingly [5], [22], [23]. In order to adapt to the new concept, most of these approaches [22], [23] forget any information learnt before the detected drift. Similarly to the sliding window strategy, for class evolution, this means that useful information will be forgotten. DDD [5] is a special type of drift detection method that keeps old ensembles while they are useful. However, DDD can only keep old ensembles corresponding to one of the previous concepts. Therefore, in the case of class evolution, DDD will also forget information when more than one class evolution behavior happens over time.

Class reoccurrence in class evolution is relevant to recurrent concept drift, which represents the case where a past concept reoccurs again in the data stream [24], [25], [26]. However, the two cases are substantially different. Recurrent concept means a reoccurred joint distribution for all data, and thus the whole class set involved in the concept also reoccurs. On the other hand, when class reoccurrence happens, the current concept may not be identical to any previous concept since some other classes might have disappeared. Hence, class reoccurrence may not lead to a recurrent concept, and thus might not be handled effectively with existing algorithms for recurrent concept drift.

To summarize, although the research progress on general concept drift provides inspirations for tackling class evolution, few approaches proposed therein are directly applicable in this particular case. Hence, it is unsurprising that the dedicated research on class evolution can be dated back to more than one decade ago, when Zhou and Chen [8] put forward the concept of class-incremental learning (C-IL). Since then, two major families of methods have been developed for class evolution.

The first family of algorithms includes MineClass [9], ECSMiner (ECSM, [13]), CLAM [14], MCM [10], [27] and SCANR [11]. All of them process data streams chunk by chunk. They consider class evolution from two perspectives, i.e., novel class detection and class evolution adaptation. The former task is to detect a potentially unknown class and assist human experts in data labeling. The second one, which is the focus of this work, aims to effectively maintain the model to adapt to class evolution. For class evolution adaptation, MCM and SCANR simply employ the chunk-based ensemble approach for concept drift. MineClass and ECSM extend this to a new model selection method to select related learners for voting and drop outdated ones. CLAM develops a class-based structure, where the examples for each class are trained separately. The model selection method and class-based structure are specifically designed for the main characteristic of class evolution, i.e., an unfixed class set in the learning process. However, the above strategies still have their drawbacks: (1) For class emergence, ECSM ignores the unconfident votes of aged models trained without the novel class. However, the judgment on the confidence of a vote, which relies on the outlier detection, is nontrivial. Since the example size of the novel class in each chunk increases in the class emergence stage, the base learners tend to mark the examples of novel classes in the later chunks as outliers. This will cause ECSM to misjudge the votes from the early base learners as being unconfident. For class disappearance, it removes the outdated models from the ensemble. However, if the class reoccurs later, the model needs a re-training of this class, and this makes the model inefficient. (2) In CLAM, when learning each chunk, the examples of each class are grouped into $k$ clusters to make decision. CLAM uses $k$-means [28] to generate the decision boundary, but it is difficult to set a generally suitable $k$ value for each chunk, especially for the gradual class evolution. In particular, in the early stage of emergence (reoccurrence) and the late stage of disappearance, the examples of the evolved class may be too few to be clustered. A large $k$ value is unsuitable when a class emerges or disappears, and a small one may lead to an unsatisfactory performance when its example size becomes large enough.

The other family of algorithms related to class evolution are the variants of the Learn$^{++}$ [29], i.e., Learn$^{++}$.NC (LNC, [12]), Learn$^{++}$.UDNC (LUDNC, [30]) and Learn$^{++}$.NCS (LNCS, [31]). They are inspired by AdaBoost[32], and construct a set of base learners for each chunk. In traditional chunk-based ensembles, when a novel class emerges, the former base learners that have been trained without this class will outvote the most recent ones. In order to overcome this problem, a novel weight assignment mechanism, called a dynamically weighted consult-and-vote (DW-CAV), is presented in LNC. In order to learn imbalanced data, LUDNC and its more general version, LNCS, are proposed. The SMOTE [33] oversampling strategy acts as a wrapper to preprocess the training data in LNCS. Shortcomings for these algorithms are discussed: (1) The weights for base learners are difficult to tune, especially in complicated evolution scenarios. For example, a novel class emerges with another class disappearing. In classifying the example of the novel class, the weight for the later base learner may still be pulled down, if the earlier learners classify it as the disappeared class. (2) In the learning process of these algorithms, each base learner should guarantee that the cumulative weight for the misclassified examples in its chunk is below 0.5; if this is the case, a new one should be trained instead. However, this requirement is hard to meet for the dynamically imbalanced data, especially when the data is complicated and multiple classes exist in the data stream. In this situation, the algorithm may never end. (3) Due to the dynamic class-imbalance problem in gradual class evolution, the example size may be large enough in some chunks while very limited in others. Although the minority class is considered in LNCS, the chunk-based learning method cannot effectively make use of the data. Furthermore, since all base classifiers are maintained, it is considerably time-consuming to dynamically calculate the weights of these classifiers for each test example.

## 3 The Proposed Approach

In this section, the problem of class evolution adaptation is analyzed first. Then, the new approach as well as the details of each component will be described. Finally, the approach is analyzed and summarized.

### 3.1 Problem Analysis

To further clarify the problem of class evolution adaptation, the risk of misclassification is evaluated for the case of 0-1 loss. Gradual class evolution leads the data stream to be dynamically imbalanced; in addition, the prior probability of each class may even fluctuate dramatically. In this situation, examples tend to be classified as majority classes, and the examples of minority classes are hard to identify. To eliminate this influence, a weight $e_t^i$ at time $t$ for misclassifying the example of class $c_i$ is set, as $e_t^i = 1/P_t(c_i)$, where $P_t(c_i)$ is the prior probability of class $c_i$ at time $t$. For class $c_i$ at time $t$, the risk for classifying $\mathbf{x}_t$ as class $c_i$ is

$$R_t(c_i|\mathbf{x}_t) = \sum_{j\neq i} \left(e_t^j \cdot P_t(c_j|\mathbf{x}_t)\right), \tag{1}$$

where $P_t(c_j|\mathbf{x}_t)$ is the posterior probability of class $c_j$ given example $\mathbf{x}_t$.

To maximize the learning performance, the classification risk (i.e., Eq. (1)) at each time step $t$ needs to be minimized. Since $e_t^i$ is set as $1/P_t(c_i)$, the minimization problem turns to be

$$\min_i \sum_{j\neq i} \left(\frac{1}{P_t(c_j)} \cdot P_t(c_j|\mathbf{x}_t)\right). \tag{2}$$

Since $P_t(\mathbf{x}_t)$ is the same for all classes, Eq. (2) is equivalent to

$$\min_i \sum_{j\neq i} \left(\frac{1}{P_t(c_j)} \cdot P_t(c_j|\mathbf{x}_t) \cdot P_t(\mathbf{x}_t)\right), \tag{3}$$

which is equivalent to

$$\min_i \sum_{j \neq i} P_t(\mathbf{x}_t|c_j). \tag{4}$$

By dividing each item in Eq. (4) by $\sum_j P_t(\mathbf{x}_t|c_j)$, the problem is transformed into

$$\min_i \left(1 - \frac{P_t(\mathbf{x}_t|c_i)}{\sum_j P_t(\mathbf{x}_t|c_j)}\right). \tag{5}$$

In other words, the original problem of minimizing misclassification risk transforms into the problem of finding the maximal likelihood, which is

$$\max_i P_t(\mathbf{x}_t|c_i). \tag{6}$$

## 3.2 Class-Based Ensemble for Class Evolution

Eq. (6) suggests that the optimal classification strategy is to assign an example according to the likelihood that it belongs to a class. Therefore, a natural approach to this problem is to maintain a model for each class so that the likelihood can be explicitly estimated. For this reason, the CBCE approach is proposed. Each class-based model (CB model) is maintained for a certain class $c_i$ and an example **x** is classified according to

$$\arg\max_i CBMClassify(\mathbf{x}, CBM_i), \tag{7}$$

where the function $CBMClassify$ returns the likelihood $P(\mathbf{x}|c_i)$ or scores that can be used to estimate $P(\mathbf{x}|c_i)$. Depending on the current class evolution state, the CBCE algorithm manages the CB models in mining tasks.

Specifically, it may create a new CB model for a novel class, inactivate an outdated CB model for a disappeared class and re-activate the CB model when the class reoccurs again. Since the class conditional probability is also likely to change in a real-world data stream, the previously built model for a class could become invalid later. Hence, CBCE also involves a scheme to detect and handle the invalid CB model.

### *3.2.1 Class-Based Model*

A class-based model is one that is specifically constructed for a certain class to get the likelihood (or related score) of a test example. A variety of models are possible candidates for a CB model, e.g., one-class classifier and clustering model.

In this work, the CB model is implemented as a binary classifier that is able to output its classification posterior probability. In each CB model, with the one-versus-all strategy, the represented class is the positive class (+1) and the others are the negative one ($-1$) as a whole. According to Bayesian theory, the posterior probability $P_t(+1|\mathbf{x}_t)$ for the positive class at time $t$ is

$$P_t(+1|\mathbf{x}_t) = \frac{P_t(+1)}{P_t(\mathbf{x}_t)} \cdot P_t(\mathbf{x}_t|+1), \tag{8}$$

where $P_t(\mathbf{x}_t)$ is the same for all classes. If the training data are balanced in CB models, $P_t(+1)$ is a constant 1/2. In this condition, the posterior probability for positive class is proportional to the likelihood of the positive class, i.e., the specific class the CB model is maintained for. In other words, the probability can be used as the score to represent the likelihood for making decisions.

The positive and negative classes are likely to be imbalanced in a CB model. Although class-imbalanced problem has been intensively investigated, most previous studies [31], [34] focus on static class-imbalanced problems. In our case, the prior distribution may change over time, leading to a dynamic class-imbalanced problem. To address this issue, an under-sampling strategy is embedded in each CB model. The sampling probabilities for the positive and negative classes are different. As each CB model acts as an "expert" for its corresponding class, all of the examples received from this positive class are selected. The data size of the negative classes is usually larger than the positive one. Furthermore, the size of each class dynamically changes due to the gradual class evolution. These negative examples are sampled by under-sampling with a dynamic probability, which aims to select the negative data with the same size as the positive ones. Denoting $w_t^i$ as the prior probability of class $c_i$ at time $t$, the probability of sampling the negative examples for $c_i$ is calculated as

$$p_i = \min(w_t^i/(1 - w_t^i), 1). \tag{9}$$

In on-line learning, the underlying prior probability $w_t^i$ is hard to be observed. To quickly and accurately estimate $w_t^i$, it is tracked by the time decay method [35], [36] as:

$$w_t^i = \beta w_{t-1}^i + (1-\beta)\mathbf{1}[y_t = c_i], \tag{10}$$

where $\beta$ $(0 < \beta < 1)$ denotes the decay factor, and $\mathbf{1}[y_t = c_i] = 1$ if $y_t$, the true class label of $\mathbf{x}_t$, is $c_i$, otherwise 0. To conveniently apply CBCE in practice, a constant decay factor is used for the prior probabilities of all classes. Since the estimated prior probability will be updated exponentially, it will quickly achieve its underlying value. The appropriate value for $\beta$ is 0.9, which has been determined after comprehensive experiments.

The learning procedure is summarized in Algorithm 1. When a new example is received, every CB model will update the estimation of prior probability of its class (lines 2 and 5). For the class that the currently received example belongs to, its CB model uses it for updating directly (line 3). For the other CB models, the example is first sampled with the dynamic sampling probability, and then used to update the models as a negative training example (lines 6 and 7).

**Algorithm 1.** UpdateCBModel

```
Input: $(\mathbf{x}_t, y_t)$, the example at time $t$; $CBM_i$, the CB model of
    class $c_i$; and $w_{t-1}^i$, the prior probability of $c_i$ at time $t-1$
Output: $CBM_i$, the updated CB model
1: if $CBM_i$ is the corresponding CB Model for $y_t$ then
2:     $w_t^i = \beta w_{t-1}^i + (1-\beta)$
3:     update $CBM_i$ with $(\mathbf{x}_t, +1)$
4: else
5:     $w_t^i = \beta w_{t-1}^i$
6:     $p_i = w_t^i/(1 - w_t^i)$
7:     update $CBM_i$ with $(\mathbf{x}_t, -1)$ under probability $p_i$
8: end if
```

In the CBCE framework, a CB model is required to provide its output in the form of score and can be updated on-the-fly. Quite a few classical base leaners satisfy the first requirement, and logistic regression might be the model that has been mostly investigated with regard to the second issue. Hence, the online Kernelized Logistic Regression (KLR, [37]) is employed in this work as the base learner. It should be noted that CBCE does not necessarily require to establish only one CB model for each class, and in some cases an ensemble model might be more suitable than a single model for a class. For example, if the minority class may comprise small disjuncts of data, a possibly better option for the CB model is to employ cluster over-sampling techniques [38], [39] and build a model for each disjunct of data.

The KLR adopted in this work takes the form as:

$$f_t^i(\mathbf{x}) = \sum_{j=1}^{n_i} \alpha_j^i k(\mathbf{x}_j, \mathbf{x}), \tag{11}$$

where $t$ is the time stamp, $n_i$ is the number of examples trained in $CBM_i$, $\alpha_j^i$ is the coefficient for the $j$th term in $CBM_i$, and $k(\cdot,\cdot)$ is the kernel function. The posterior probability for the $i$th CB model is

$$P_t^i(+1|\mathbf{x}_t) = 1/(1+\exp(-f_t^i(\mathbf{x}_t))). \tag{12}$$

After being fed with each training example, the online KLR algorithm updates the current classifier by stochastic gradient descent with model truncation [37]. With this implementation, the CB model can predict the probability of the classification and learn the data stream with linear time complexity.

### 3.2.2 Class Evolution Adaptation

Class evolution has three basic elements, i.e., the emergence of novel classes, the disappearance of outdated classes, and the reoccurrence of disappeared classes.

When a novel class $c_i$ emerges at time stamp $t$, CBCE first estimates its prior probability $w_t^i$, and then initializes a new CB model $CBM_i$ for it. The prior probability is initially estimated after receiving the first two examples of this class. Denoting $ExampleSize$ as the example size of the negative classes between these two examples, the prior probability is estimated as follows:

$$w_t^i = 1/(ExampleSize + 1). \tag{13}$$

Based on the two examples of novel class and the negative examples between them, the CB model is initialized. Next, the CB model participates in classifying the subsequent data stream.

For class disappearance, the approach has to determine the disappearance when a class is shrinking; following this, its CB model should be managed to ensure not to affect the recognition of other classes. Since the evolution state is tracked in CB models, a sufficiently small prior probability threshold, e.g., $\beta^{1,000}$ ($\beta$ is the decay factor in Section 3.3), can be used for disappearance confirmation. That is, if the class has been absent for 1,000 consecutive time stamps, it is thus considered to have disappeared. The decision boundary of the CB model, as implemented by binary classifier, merely separates one class from another. In this case, if the class-conditional probability distribution changes or a novel class emerges on the boundary, the original CB model for the disappeared class would be inaccurate and also influence the novel class. Therefore, the CB model of the disappeared class is inactivated in classification. Besides, when a class is considered to have disappeared, its estimated prior probability is set to be 0, which also means its CB model is suspended for updating.

Class reoccurrence means that an example with the label of a disappeared class is received again. Effective handling of class reoccurrence could make use of past training efforts. For the inactivated CB model of a disappeared class, it can be used again for classification when an example with an old label arrives, which makes CBCE efficient. Once class reoccurrence happens, the model re-estimates the prior probability in the same way as class emergence, and activates the CB model in classification.

This mechanism to deal with the three key components of class evolution is wrapped around each CB model, which equips CBCE to track gradually evolved classes effectively. The procedure of class evolution adaptation is summarized in Algorithm 2. Depending on the change of prior probability, class evolution behavior can be determined. The active CB models are updated by the sampled data, and the inactive ones are stored additionally in case of class reoccurrence.

**Algorithm 2.** ClassEvolutionAdaptation

```
Input: $(\mathbf{x}_t, y_t)$, the example at time $t$; $CBM_i$, the obtained CB models at $t-1$, $i = 1, 2, \cdots, |CBM|$; $C_t$, the class set at $t$; and $w_t^i$, the prior probability of $c_i$ at time $t$
Output: $CBM$, the class-based ensemble
 1: $C_t \leftarrow C_{t-1}$
 2: if no $CBM_i$ is available for $y_t$ then
 3:    // class emergence
 4:    $C_t \leftarrow C_t \cup \{y_t\}$
 5:    if $\mathbf{x}_t$ is the first example of class $y_t$ then
 6:       buffer the incoming examples for class $y_t$
 7:    else if $\mathbf{x}_t$ is the second example of class $y_t$ then
 8:       initialize the $w_t^y$ of $y_t$
 9:       initialize a CB model for class $y_t$
10:    end if
11: else if $CBM_y$ is a CB model for $y_t$ and $w_t^y = 0$ then
12:    // class reoccurrence
13:    $C_t \leftarrow C_t \cup \{y_t\}$
14:    if $\mathbf{x}_t$ is the first (recurring) example of class $y_t$ then
15:       activate $CBM_y$ for classification
16:       buffer the incoming examples for class $y_t$
17:    else if $\mathbf{x}_t$ is the second (recurring) example of $y_t$ then
18:       initialize the $w_t^y$ of $y_t$
19:    end if
20: end if
21: for each $c_i$ in $C_t$ do
22:    // class disappearance
23:    if $w_t^i < disappearance\ threshold$ then
24:       $C_t \leftarrow C_t - \{y_t\}$
25:       $w_t^i \leftarrow 0$
26:       inactivate $CBM_i$ for classification
27:    end if
28: end for
29: $UpdateCBModel(\mathbf{x}_t, y_t, CBM)$ for each active CB model
```

#### *3.2.3 Class-Conditional Probability Change Adaptation*

Although CBCE focuses on class evolution adaptation in data stream mining, it is also very likely that class conditional probability distribution changes over time.

In CBCE, the change of class-conditional probability distribution means that the CB model is no longer able to correctly identify its corresponding positive examples. To handle this problem, a simple and yet effective drift detection method, DDM [22], is applied to check a CB model's validity. If a CB model was significantly affected by this type of change, it would be re-initialized. Each example used for training the CB model is incorporated for the detection of the change. If the warning level is reached, the CB model is likely to be outdated and the following sampled examples are stored. If DDM detects a drift in a CB model, the model is re-initialized by these examples. Through this method, the likelihood value obtained with each CB model is avoided to be affected by the change of class-conditional probability distribution.

### 3.3 Summary and Analysis

In CBCE, when a new example is received, the ensemble model first predicts its label for practical use. After obtaining the true label of this example, each CB model is updated to track the up-to-date concept. If a novel class emerges, a new CB model corresponding to this class is initialized. A sufficiently small prior probability of a class implies its disappearance. In this case, the corresponding CB model is inactivated but still conserved. If a disappeared class reoccurs, the corresponding CB model will be re-activated with the prior probability of the class being re-estimated from the current data. In order to handle the dynamic class-imbalance problem caused by the gradual process of class evolution, CB models use under-sampling with a dynamic probability to sample the examples to balance the training data. It is noted that all active CB models are used for classification, with decision determined by choosing a class whose CB model outputs the highest score. A change detection method is used to monitor changes in the class-conditional probability distributions corresponding to each CB model. If a change is detected, the corresponding CB model is reset.

As mentioned before, most existing approaches for class evolution, such as LNCS and ECSM, process a data stream chunk by chunk. The class-based framework adopted by CBCE has a number of advantages in comparison to the existing methods. First, since a CB model is specifically maintained for a certain class, it is flexible to be created or removed to adapt to class evolution. This also decouples the whole model, and makes each CB model simple and concentrate on a single class. Second, by using the CB model, only a few of base learners need to be maintained, equal to the number of classes. Third, for massive-volume data streams, the master-slave structure (CB model – ensemble strategy) of the learning system is also very convenient for parallelization and distributed implementation.

The loss of the classification result for each example is bound by the online learning approach. Scaled in $[0, 1]$, the output score of a CB model is ideally 1 for the correct class and 0 for others. In the case of binary classifiers, the expected score can be viewed as the posterior probability that an example belongs to the positive class, i.e., $P(+1|\mathbf{x}_t, CBM_i)$. For testing example $\mathbf{x}_t$ from class $c_i$, the weighted 0-1 loss is $e_t^i$ for incorrect classification and 0 for the correct one. It can be found that the loss $L(\mathbf{x}_t)$ is bounded as follow:

$$L(\mathbf{x}_t) \leq e_t^i \cdot ((1 - P(+1|\mathbf{x}_t, CBM_i)) + \sum_{j \neq i} P(+1|\mathbf{x}_t, CBM_j)), \tag{14}$$

where $(1 - P(+1|\mathbf{x}_t, CBM_i))$ represents the gap of $CBM_i$ to the optimum, and $\sum_{j \neq i} P(+1|\mathbf{x}_t, CBM_j)$ is the sum of values from all other $CBM$s. Then,

$$L(\mathbf{x}_t) \leq e_t^i \cdot ((1 - P(+1|\mathbf{x}_t, CBM_i)) + \sum_{j \neq i} (1 - P(-|\mathbf{x}_t, CBM_j))). \tag{15}$$

For class $c_i$, $P(+1|\mathbf{x}_t, CBM_i)$ is the posterior probability for correct classification by $CBM_i$, and $P(-|\mathbf{x}_t, CBM_i)$ is the posterior probability for correct classification in other CB models. If $P_i^{correct}(\mathbf{x}_t)$ is used instead as the probability of correct classification for any CB model, then the loss is bounded as follow:

$$L(\mathbf{x}_t) \leq e_t^i \cdot (|C_t| - \sum_i P_i^{correct}(\mathbf{x}_t)), \tag{16}$$

where $|C_t|$ is the number of classes at $t$. The values of $e_t^i$ and $|C_t|$ are the same for each class. For each CB model, with more training data, the confidence of correct classification ($P_i^{correct}(\mathbf{x}_t)$), is expected to be increased.

From the above analysis, it can be found that the loss is bounded based on the performance of each CB model. With more training data for each CB model, the models would be more accurate, with each CB model having a higher confidence in its prediction. The bound would gradually get tighter and the performance better.

## 4 Experimental Studies

The properties and performance of CBCE were observed through two types of experiments, i.e., the visualization experiment and the comparative experiment.

### 4.1 Visualization Experiment of CBCE

This experiment aims to visualize the learning process of CBCE to gain a deeper understanding of its behavior. For this purpose, a two-dimensional synthetic data stream is generated, which involves 10,000 examples from four classes. The data distribution and the class evolution behaviors of the four classes are shown in Fig. 1. Different types of class evolution behaviors are designed for the four classes. Class 1 is a stable class without evolution. Class 2 firstly disappears and then reoccurs. Class 3 is a novel class that gradually emerges. Class 4 represents a sudden event in which a class emergence is closely followed by a disappearance. Assume the EGR is 1 when in a stable condition for each class. Part of the Gaussian bell curve is employed to simulate the gradual increase or decrease of the EGR in Fig. 1b. The peak position of a Gaussian curve represents EGR = 1. The 3-sigma position approximately

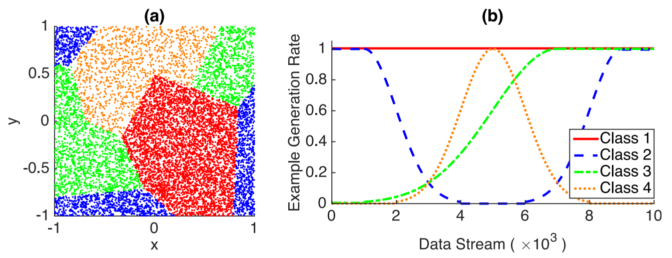

Fig. 1. (a) Data distribution and (b) class evolution behavior of the 2D data stream used in Section 4.1.

represents EGR = 0, which corresponds to the beginning of class emergence (reoccurrence) or the end of the disappearance process.

Gaussian Kernel is chosen as the kernel function in the online KLR of a CB model, and the kernel width $\sigma$ is set as the 5th percentile of the pairwise distances between all pairs of examples [40]. As the class-conditional probability distribution is stable in the synthetic data stream, CBCE, without the detector for the change of class-conditional probability, is applied in this experiment. To tune the parameters in KLR, an initial fraction of the synthetic data stream is utilized with a variant of five-fold cross validation, i.e., leaving out an example from every five examples to construct the training stream. By this method, the parameters are set as $\eta = 0.01$, $\lambda = 0.1$, and $\sigma = 0.1$.

The decision boundaries corresponding to the four CB models are plotted in Fig. 2.

Specifically, after the first 1,000 examples have been processed, the CB models for class 1 (red) and 2 (blue) are constructed. When 2,000 and 3,000 examples have been processed, it can be found that the CB models for class 3 (green) and class 4 (orange) have been initialized. Meanwhile, the CB models for classes 1 and 2 are updated. After the 4,000th example is processed, class 2 disappears (no new example belongs to class 2 in diagram) and the corresponding CB model remains unchanged until class 2 reoccurs again after the 6,000th example. The final CB models obtained after processing the entire data stream are shown in the last diagram of Fig. 2. It can be observed that the four CB models effectively separate the examples of different classes. Besides, it can also be found that CBCE incrementally adjusts each CB model to be a good local "expert" and is capable of adapting itself to the evolution of classes.

## 4.2 Comparative Experiment

To verify the performance of CBCE, a comprehensive comparison between CBCE and other approaches is carried out in the comparative experiment.

### 4.2.1 Data Set

Two sets of synthetic data streams and one set of real-world data streams are used in the experiment.

*Synthetic data.* Letter recognition data set (16 numeric attributes) and Statlog (landsat satellite) data set (36 numeric attributes) from the UCI Machine Learning Repository [41] are modified to compose the synthetic data streams. They are generated by re-arranging the examples to fit the class evolution setting. For both data sets, examples of four classes are extracted as the source of the data streams. In the Letter recognition set, letters "a", "b", "c" and "d" represent these four classes, respectively. In the Statlog set, "red soil" is used as class 1, "grey soil" as class 2, "cotton crop" as class 3, and class 4 is represented by "damp grey soil".

Three fundamental class evolution scenarios (Fig. 3) are considered, i.e., class emergence, class disappearance and reoccurrence, and multiple class evolution. Almost all complicated class evolution scenarios can be decomposed into the three basic ones. Furthermore, the use of the three basic scenarios also allows a close observation of the performance of each approach.

Class 1 to 3 from the synthetic data sets are used in scenarios $a$ and $b$, and all the four classes are used in scenario $c$. As shown in Fig. 3, class 3 (green) is designed as an evolved class with class emergence in scenario $a$, and class disappearance and reoccurrence in scenario $b$. In scenario $c$, class 3 and class 4 (orange) successively emerge and then disappear, as a more complex situation. As the description in the previous experiment, 1,650 examples are extracted from each Letter recognition data stream, and 2,750 examples from each Statlog data stream.

*Real-world data.* UDI TwitterCrawl Dataset [42], including 50 million tweets posted mainly from 2008 to 2011, is involved. Each record in this data set has its own time stamp and the order of examples in the data stream is completely genuine, without any modification. Since the hashtag roughly describes the tweet's topic, it was used as the class for each tweet record. If more than one hashtags exist in a tweet, one of them is selected randomly as its label.

Four tweet stream fragments from the whole tweet set are captured by selecting different topics as the classes of interest, i.e., tweet stream $a$, $b$, $c$, and tweet stream-20 classes. The first three tweet streams correspond to the three basic class evolution scenarios $a$, $b$ and $c$ described in the synthetic data, for further observation. Specifically, tweet stream $a$, involving 39,600 tweets, represents the class emergence scenario. It has the topic of "royal wedding" (class 3, between Prince William and Kate Middleton) acting as the novel class. Corresponding to the class disappearance and

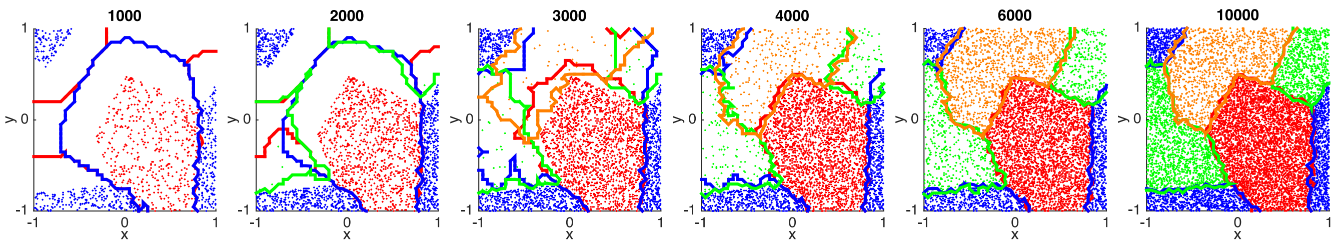

Fig. 2. Visualization of the classification behavior of CBCE on 2D synthetic data stream. The diagrams show the decision boundaries and the processed data points at different time steps.

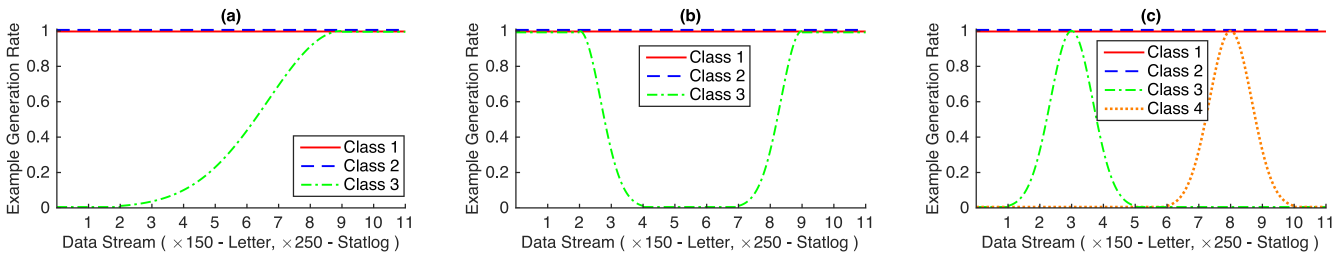

Fig. 3. Three class evolution scenarios in synthetic data of comparative experiment.

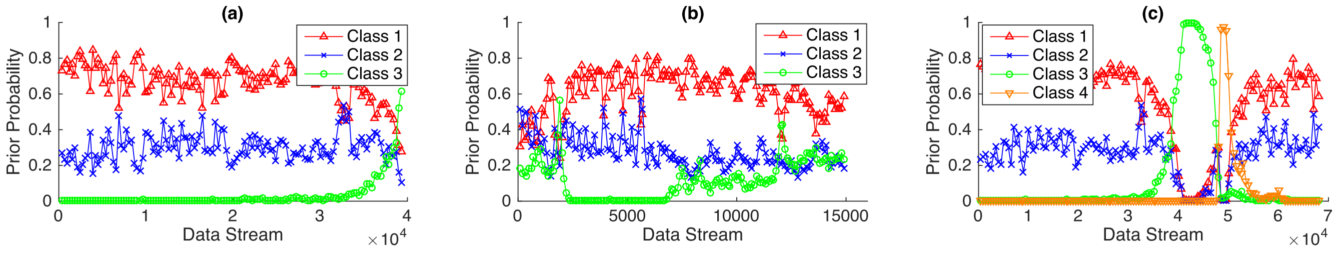

Fig. 4. Three class evolution scenarios of the comparative study on tweet data.

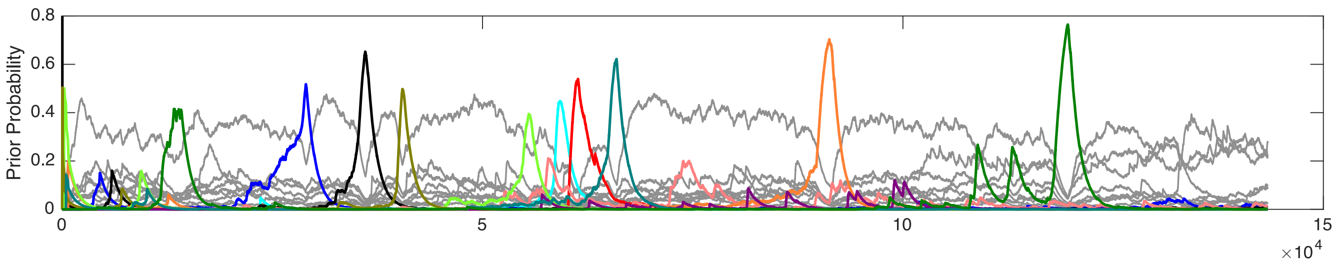

Fig. 5. Class evolution behaviors of the 20 classes in tweet stream-20 classes. The grey lines represent non-evolved classes, and the others represent evolved classes.

reoccurrence scenario, tweet stream $b$ takes a fragment of 15,004 tweets, with the topic of "Christmas" (class 3) as the evolved class. Tweet stream $c$ (the multiple novel classes scenario) covers 68,750 tweets, where the topics of "royal wedding" and "bin Laden" (class 4, the news about the hunt for Osama Bin Laden) are the novel classes. In the three streams, the topics of "job" (class 1) and "music" (class 2) act as the "stable" classes. To obtain a high fidelity simulation of real class evolution scenarios, tweet stream—20 classes is generated. It involves 143,381 tweets with 20 classes, including 9 "stable" classes and 11 evolved classes. Since the class evolution state along the tweet streams is implicit, the prior probabilities of classes through the tweet streams is estimated by Eq. (10) ($\beta = 0.99$ to make the line smooth) and visualized in Figs. 4 and 5.

After getting the tweet streams, the text of each tweet is transferred into the TF-IDF vectors. 242, 247, 242 and 524 numerical features are generated, respectively, for the tweet streams $a$, $b$, $c$ and the tweet stream - 20 classes. From Fig. 5, it can be seen that class evolution may occur frequently in tweet stream. Besides, according to the visualization of tweet stream in Fig. 6, it can be observed that the class-conditional probability distribution also changes over time in tweet stream.

### 4.2.2 Compared Approaches

The synthetic data streams are constructed from data sets with fixed distribution. For these streams, CBCE without class-conditional probability change adaptation is tested. Since the class-conditional probability changes in tweet data, CBCE with the distribution change adaptation (named as CBCE$^d$) is tested as well on tweet streams.

To the best of our knowledge, none of the existing approach for class evolution is designed to process data streams in an online manner. Hence, four state-of-the-art approaches for class evolution, including ECSM [13], CLAM [14], LNCS [31] and AUE2 [21], are employed in our comparative studies. These approaches all mine data streams in a chunk-by-chunk manner. In the experiment, they were given the advantage of collecting the examples first, i.e., they update the corresponding models when a chunk of examples have been collected. It is noteworthy that CLAM trains a model using the $k$-means clustering method, and implicitly presumes that each class comprises at least $k$ examples. Thus, if a class comprises less than $k$ examples in a chunk, the class will be regarded as a single cluster directly.

To verify how much better the sophisticated methods perform, kNN over a sliding window of a fixed size (SKNN), is also tested as a baseline. Considering the feature of the sliding window strategy, SKNN would work nicely for the concept drift with abrupt changes, because the change of data distribution is fast and will not cause lasting impacts on model construction.

### 4.2.3 Parameter Settings

The parameters for CBCE and CBCE$^d$ are set according to the parameter setting description in the previous experiment.

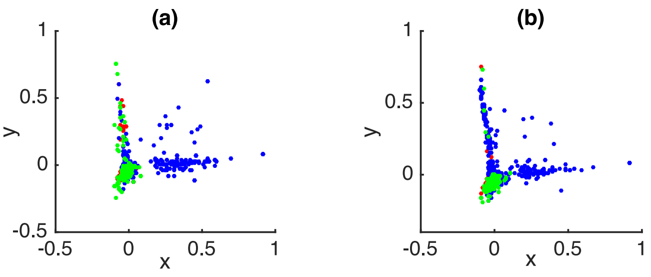

Fig. 6. An illustration of the change of class-conditional probability in tweet stream-$c$. The left and right figures illustrate the examples (after dimensionality reduction) of the same classes at different time points in the stream.

For the synthetic data streams, the parameters are set $\eta = 0.01$, $\lambda = 1$ and $\sigma = 1$ and 14, respectively, for the Letter recognition streams and the Statlog streams. For the tweet data streams, $\eta = 0.3$, $\lambda = 0.0005$ and $\sigma = 0.13$. To speed up KLR, in formula (5), the term whose coefficient is small enough (i.e., $10^{-5}$) will be dropped by the truncation operation[37]. For the tweet streams, 5,000 examples would be stored at most, and the exceeding examples would also be truncated.

All parameters of other compared approaches are set either according to the default setting or by trial-and-error to get an overall satisfactory performance. Specifically, for each algorithm, the default values for its parameters (as suggested in the original publication) were adopted as the initial choices. Then, a grid search was applied to the values around the default settings. SKNN is online trained with a sliding window, and all other compared approaches process data streams chunk-by-chunk. The same chunk and window sizes are tested for all algorithms for the sake of fairness. The setting details of these approaches are described as follows.

In LNCS, the number of base learners for each chunk of data is set as 10 according to [12], [31]. For the synthetic data streams, 10 Multi-layer Perceptrons (MLP, 1 hidden layer with 20 neurons, 0.05 error goal) are trained for each chunk as the suggested default setting [12], [31]. Due to the complexity of tweet data streams, the base learner may not meet the requirements of LNCS, thus causing the algorithm never to end. After testing MLP, KLR and decision tree, we chose decision tree as the based learner in tweet streams, as it is most likely to finish the mining of the tweet streams. The $k$ value (number of the nearest neighbors) for SMOTE wrapper in LNCS is set as 3.

Other chunk-based ensemble methods, i.e., CLAM and ECSM, and the hybrid one, i.e., AUE2, all involve the ensemble size, $k$, as a parameter. The above-mentioned grid search procedure confirmed that a relatively small value of $k$ (e.g., around 3-5) as suggested in the original publications generally performs well. Hence, $k$ was set to 3 (default setting in [14]), 3, and 5 for CLAM, ECSM, and AUE2, respectively. The other parameters of CLAM and ECSM were also fine-tuned by grid search. Specifically, the cluster number of CLAM was set to 5, the number of pseudo-points in ECSM was set to 10 and 100 for the synthetic and twitter data streams, respectively. Besides, the number of nearest neighbors was set to 3 according to a line search from 1 to 10.

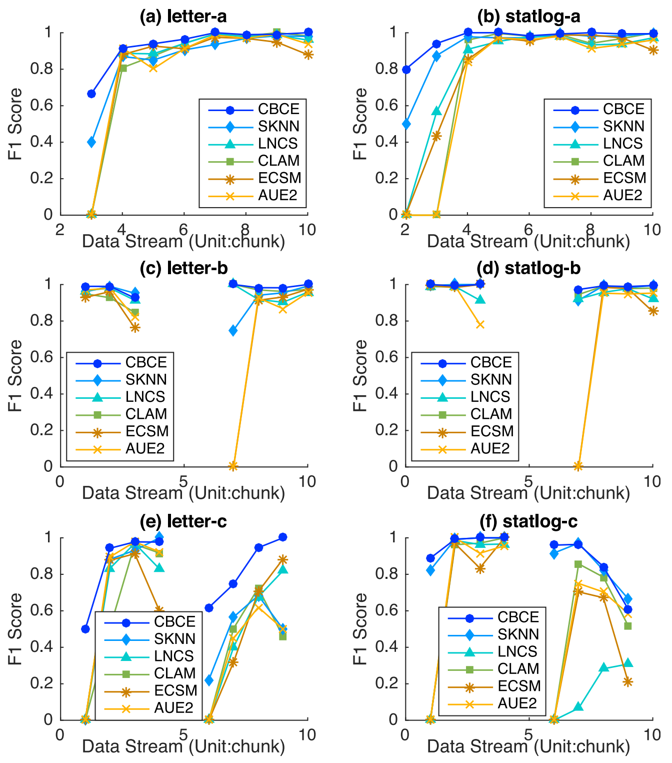

Fig. 7. F1 scores on the synthetic streams.

### 4.2.4 Evaluation

The comparative studies are conducted mainly from two perspectives. First, to provide a detailed analysis on the performance in different types of class evolution, a fixed chunk/window size (i.e., one eleventh of the stream size) is applied. The $a$, $b$ and $c$ scenarios of synthetic streams and tweet streams were used for this purpose. We apply F1 score on the evolved class to check the approaches' ability in adapting to class evolution, and use the G-mean for multiple classes [43] to measure the overall mining performance, i.e., $G\text{-}mean = (\prod_{i=1}^{k} R_i)^{1/k}$, where $R_i$ is the recall for class $c_i$. The G-mean is a better overall performance measure than accuracy for imbalanced data and is insensitive to the degree of imbalance.

Second, to investigate the impact of chunk size (or window size) on the compared algorithms, experiments have also been conducted with different sizes for all algorithms. The average G-mean for multiple classes is used to measure the performance of each approach. To be fair, the first chunk (chunk 0) is just used for model initialization. Except for the evaluation of classification ability, the time efficiency is also compared as a metric.

The detailed performance of the approaches under basic scenarios in synthetic data streams is shown in Figs. 7 and 8. Fig. 7 shows the F1 score of the evolved classes. In the class emergence scenario (Figs. 7a and 7b), it can be observed that CBCE is able to adapt to the novel class rapidly, even in the early stage of emergence. CBCE also shows a high F1 score in the disappearance and reoccurrence scenario (Figs. 7c and 7d). Since ECSM drops the outdated base learners when a class disappears, it cannot identify the examples of that class effectively when it reoccurs again. The multiple novel classes scenario (Figs. 7e and 7f) demonstrates the F1 scores of two evolved classes. The left part of the two figures represents

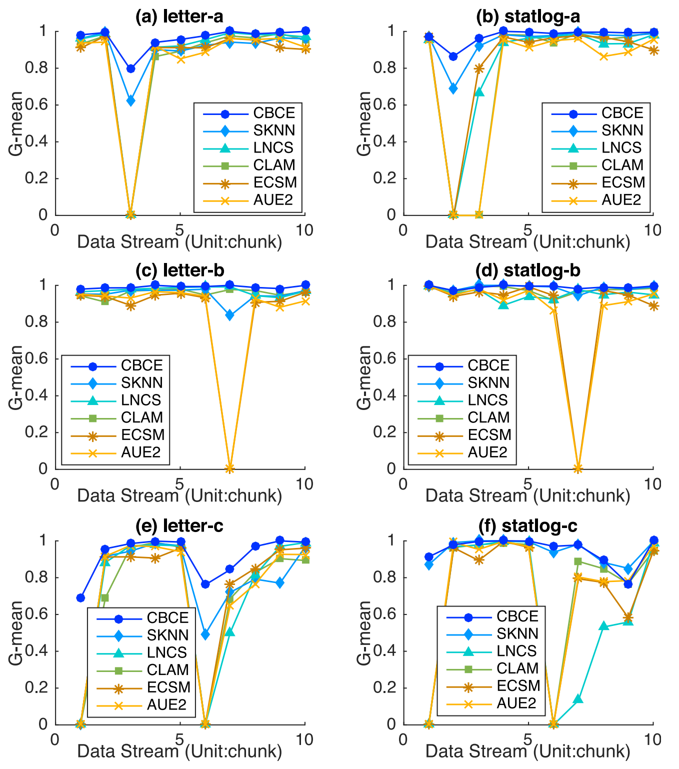

Fig. 8. G-means on the synthetic streams.

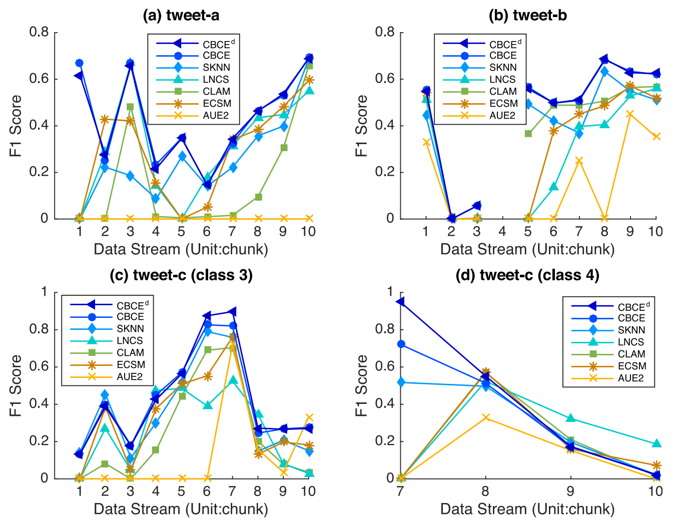

Fig. 9. F1 scores on the tweet streams.

the first novel class (class 3), and the right part shows the F1 scores of the second one (class 4). In classifying the examples of class 3, CBCE stays ahead in the same way as scenario $a$. For the second novel class, CBCE still works well, while the performance of the compared approaches obviously decreases. Fig. 8 shows the results of G-mean for each approach. It can be observed that CBCE performs the best among all the approaches, and the class evolution makes minimal impact on the CBCE. For the other approaches, the second evolved class in scenario $c$ is not only hard to be identified but also deteriorates their overall performance.

The F1 score and G-mean results achieved on tweet streams is shown in Figs. 9 and 10. In addition to CBCE, CBCE$^d$ is also tested in the tweet streams. The results of CBCE and CBCE$^d$ are similar, and CBCE$^d$ improves slightly in general. The result of CBCE is roughly consistent with that in the synthetic streams. For the F1 result in the multiple novel class scenario, the first novel class (Fig. 10c) still performs the best among all compared approaches. However, the F1 scores on the second novel class (Fig. 10d) of CBCE are not as good as the previous results. It might be the reason that class 4 emerges suddenly and almost all the tweets at that time belong to this topic and then the prior probability of class 4 drops down quickly. Interestingly, the suddenly emerged topic exposes the shortcoming of chunk-based approaches, which detect the new class only when it is fading away. Comparing the G-mean result with that of synthetic data streams, the performance of CBCE drops slightly. The reason might be the specificity of tweet data. For example, the tweet and re-tweet share the same topic and are always posted at very close times. Besides, using "job" topic as an example, many job recruitment tweets are posted all at once. The characteristic of tweet data leads to a wild fluctuation of prior probability, and the imbalance problem turns out to be extremely dynamic, instead of evolving smoothly. However, this problem is relieved in the compared approaches, which process the examples in a chunk as a whole. Even so, CBCE and CBCE$^d$ still generally perform better.

To compare the algorithms with different settings of chunk size, the average G-mean [44] over each chunk is adopted to evaluate the approaches for the whole data stream. Tables 1 and 2 summarize the average G-mean result of all approaches under different chunk (or window) sizes. For the synthetic data streams (Table 1), the result clearly shows that CBCE is significantly better than the other compared methods. Although the data sets for synthetic streams are not complicated, some compared approaches still perform similar to and even worse than the simple baseline approach SKNN. A similar result can be obtained from the tweet streams. For tweet stream $a$-$c$, CBCE and CBCE$^d$ are significantly better than other approaches. For tweet stream—20 classes, the evolution behaviors are more complicated. Thus the performance of all the compared algorithms deteriorate significantly on this data stream. However, it can be still observed that CBCE and CBCE$^d$ outperform the other algorithms when the chunk size is relatively small. Since a chunk size of 30,000 might be sufficiently large for building

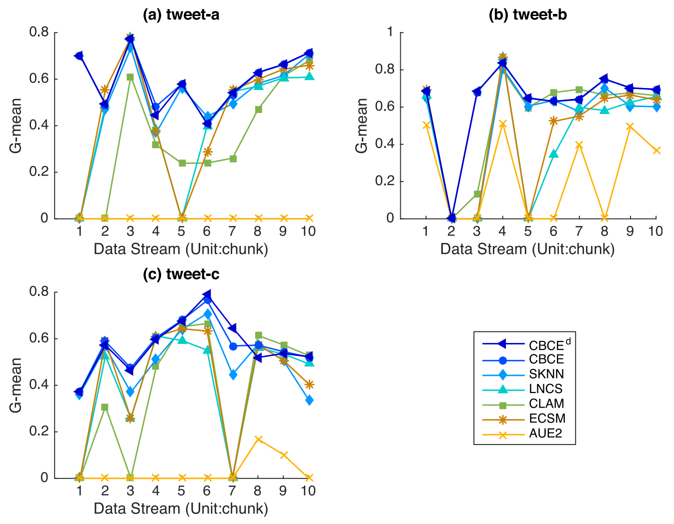

Fig. 10. G-means on the tweet streams.

TABLE 1
G-mean Results of Class Evolution Scenarios with Different Chunk Sizes on the Synthetic Data Streams

| | Letter Stream - A | | | | | Letter Stream - B | | | | | Letter Stream - C | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Learner | 100 | 150 | 200 | 300 | 500 | 100 | 150 | 200 | 300 | 500 | 100 | 150 | 200 | 300 | 500 |
| CBCE | **.9089**† | **.9622**† | **.9595**† | **.9445**† | **.9795**† | **.9899**† | **.9911**† | **.9911**† | **.9929**† | **.9946**† | **.9293**† | **.9189**† | **.9255**† | **.9543**† | **.9432**† |
| SKNN | .8577 | .9096 | .9183 | .8928 | .9643 | .9485 | .9497 | .9584 | .9537 | .9812 | .6509 | .7562 | .6589 | .9061 | .6013 |
| LNCS | .8640 | .8663 | .8377 | .7654 | .9176 | .9698 | .9720 | .9723 | .9623 | .9682 | .7035 | .7081 | .6769 | .7661 | .5983 |
| CLAM | .8081 | .8484 | .8207 | .7071 | .6580 | .9634 | .9588 | .9757 | .9820 | .9829 | .7056 | .6924 | .5854 | .6337 | .3193 |
| ECSM | .8251 | .8347 | .8101 | .7471 | .7702 | .8626 | .8397 | .8102 | .7427 | .6435 | .7757 | .7220 | .6860 | .6504 | .5847 |
| AUE2 | .8149 | .8331 | .8177 | .7374 | .6331 | .8774 | .8418 | .9341 | .9380 | .7878 | .7270 | .7067 | .5495 | .5615 | .5431 |
| | Statlog Stream - A | | | | | Statlog Stream - B | | | | | Statlog Stream - C | | | | |
| Learner | 100 | 250 | 300 | 500 | 1000 | 100 | 250 | 300 | 500 | 1000 | 100 | 250 | 300 | 500 | 1000 |
| CBCE | **.9535**† | **.9763**† | **.9829**† | **.9833**† | **.9943**† | **.9911**† | **.9907**† | **.9913**† | **.9889**† | **.9857**† | **.8858**† | .9495 | **.9446**† | **.9695**† | **.9392**† |
| SKNN | .9178 | .9458 | .9689 | .9664 | .9882 | .9334 | .9860 | .8759 | .9790 | .9791 | .8032 | **.9514**† | .9351 | .9593 | .9038 |
| LNCS | .8512 | .8332 | .7893 | .7177 | .9386 | .9648 | .9520 | .9689 | .9383 | .8672 | .7394 | .6153 | .7417 | .7323 | .3522 |
| CLAM | .7589 | .7782 | .8626 | .7728 | .9597 | .9758 | .9738 | .9724 | .9709 | .9773 | .7685 | .7363 | .7129 | .7553 | .4253 |
| ECSM | .7589 | .8438 | .8626 | .7728 | .9597 | .9758 | .8598 | .9724 | .9709 | .9773 | .7685 | .6912 | .7129 | .7553 | .4253 |
| AUE2 | .8455 | .7742 | .8327 | .7404 | .8681 | .8821 | .8437 | .7304 | .9231 | .8770 | .7505 | .7250 | .6827 | .7241 | .3787 |

*The best result is in boldface. If it is significantly better than others (Wilcoxon rank sum test at 95 percent confidence level), it is marked with †.*

TABLE 2
G-mean Results of Class Evolution Scenarios with Different Chunk Sizes on the Tweet Data Streams

| | Tweet Stream - A | | | | | Tweet Stream - B | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Learner | 300 | 1,000 | 3,000 | 3,600 | 10,000 | 300 | 1,000 | 1,364 | 3,000 | 5,000 |
| CBCE$^d$ | .5449 | .5190 | .6053 | .5950 | **.6073**‡ | **.6818**‡ | **.6574**† | **.6277**‡ | **.7265**‡ | **.7858**‡ |
| CBCE | **.5470**‡ | **.5248**‡ | **.6066**‡ | **.5972**‡ | .6067 | .6811 | .6566 | .6272 | .7262 | .7856 |
| SKNN | .3391 | .3504 | .5459 | .4975 | .5729 | .5262 | .5337 | .5185 | .6910 | .7513 |
| LNCS | .2791 | .2709 | .4664 | .4376 | .5540 | .5480 | .4255 | .4330 | .6532 | .7516 |
| CLAM | .3802 | .3721 | .3962 | .3429 | .3780 | .6085 | .5720 | .5583 | .6255 | .7620 |
| ECSM | .3696 | .3907 | .4698 | .4457 | .5277 | .5749 | .4791 | .4589 | .4685 | .7660 |
| AUE2 | .0908 | .0932 | .0646 | .0000 | .0000 | .2139 | .2236 | .2283 | .2829 | .4082 |
| | Tweet Stream - C | | | | | Tweet Stream - 20 Classes | | | | |
| Learner | 300 | 1,000 | 3,000 | 6,250 | 10,000 | 300 | 1,000 | 3,000 | 10,000 | 30,000 |
| CBCE$^d$ | .4916 | .4904 | .5557 | **.5696**‡ | **.5962**‡ | **.0536**† | .0260 | .0183 | .0000 | .0000 |
| CBCE | **.5089**† | **.5154**† | **.5647**† | .5691 | .5944 | .0524 | **.0266**‡ | **.0253**† | .0000 | .0000 |
| SKNN | .2629 | .2786 | .3663 | .5024 | .5599 | .0000 | .0000 | .0000 | .0000 | .0000 |
| LNCS | .1334 | .2985 | .4425 | .4128 | .4203 | – | .0000 | .0000 | .0000 | **.0565**† |
| CLAM | .4145 | .4373 | .4371 | .3820 | .3250 | .0001 | .0020 | .0074 | .0000 | .0529 |
| ECSM | .3214 | .3499 | .4465 | .4223 | .4310 | .0000 | .0000 | .0000 | .0000 | .0000 |
| AUE2 | .1466 | .1460 | .2550 | .0271 | .0188 | .0000 | .0000 | .0000 | .0000 | .0000 |

*1. The best result is in boldface. If it is significantly better than others (Wilcoxon rank sum test at 95 percent confidence level), it is marked with †. For the situation that the best result is from CBCE and CBCE$^d$,if they are not significantly different from each other but significantly better than other results, it is marked with ‡. 2. "–" means LNCS processes a chunk of data over $10^5$ seconds and may never end in experiment.*

TABLE 3
Friedman Test (Nemenyi Test at $\alpha = 0.05$) Result Considering the G-Mean Values in Synthetic & Tweet Streams

| | CBCE$^d$ | CBCE | SKNN | LNCS | CLAM | ECSM | AUE2 | critical difference |
|---|---|---|---|---|---|---|---|---|
| Synthetic Streams | – | 1.0333 | 2.7000 | 3.9667 | 3.9000 | 4.1333 | 5.2667 | 1.3765 |
| Tweet Streams | 1.8000 | 1.8000 | 4.5250 | 4.9750 | 4.0500 | 4.3750 | 6.4750 | 2.0141 |

an accurate model based on a single chunk, such a setting favors chunk-based ensembles. As a result, LNCS and CLAM perform better in this case. Furthermore, since tweet stream $b$ was collected from two separate time spans, it is more likely to involve significant concept drift in terms class-conditional distribution. Thus, the clear advantages of CBCE$^d$ over CBCE on this stream demonstrates the effectiveness of the DDM component. On the other hand, tweet streams $a$ and $c$ were collected within a much shorter period (about 2 or 3 months) and the concept might only drift slightly between two consecutive data chunks. Hence, the difference between CBCE$^d$ and CBCE is not significant in these cases. The effectiveness of DDM also deteriorate on tweet stream—20 classes due to the complexity of this stream. Furthermore, Friedman tests have been conducted to analyze the empirical results, as shown in Table 3. It can be observed that CBCE and CBCE$^d$ are significantly better than all the compared algorithms.

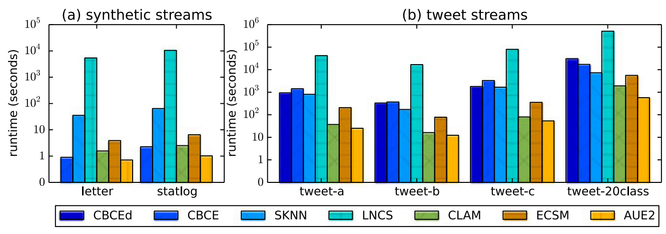

Fig. 11. Runtime of the compared approaches.

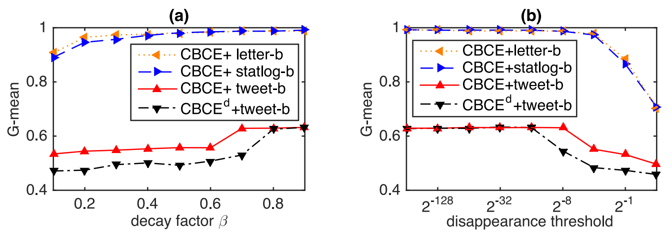

Fig. 12. Influence of (a) decay factor $\beta$ and (b) disappearance threshold.

The runtime of the approaches are compared under the same computing environment (2 CPUs of 2.4 GHz Intel Core i5, 8 GB main memory), as shown in Fig. 11.

The chunk size is selected as one eleventh of the stream size for scenarios $a$ to $c$ and 10,000 for the tweet stream—20 classes. The letter streams and stalog streams share the same data size for different scenarios, and the time is averaged and presented as a whole. CBCE is competitive in terms of runtime in the experiment with synthetic data streams but a little worse in the tweet streams. Due to the chunk-based mining manner and the simple example process method, AUE2, CLAM and ECSM generally perform best in both the synthetic data steams and the tweet data streams.

From the comparison of the mining results, CBCE is shown to outperform other algorithms in adapting different types of class evolutions for both the evolved classes and the whole data streams. The empirical study also confirms that CBCE has a satisfactory time efficiency in mining data streams. Generally speaking, CBCE is able to construct a satisfactory model for handling gradual class evolution. However, the results on tweet stream—20 classes also show that data stream mining with multiple and complex evolved classes is still a tough problem. To further investigate CBCE, the influence of decay factor and disappearance threshold is studied, as shown in Fig. 12. It can be found that a decay factor of 0.9 allows CBCE to achieve a good result in all the data streams. Considering the tracking of prior probability of classes as well, 0.9 is recommended as the default setting of decay factor. Disappearance threshold is a parameter specific to each application. From the result, a small value (e.g., less than $2^{-16}$) is a good initial setting.

## 5 Conclusion

Previous investigations on data stream mining assume class evolution to be the transient changes of classes, which does not hold for many real-world scenarios. In this work, class evolution is modeled as a gradual process, i.e., the sizes of classes increase or shrink gradually. A new data stream mining approach, CBCE, is proposed to tackle the class evolution problem in this scenario. CBCE is developed based on the idea of a class-based ensemble. Specifically, CBCE maintains a base learner for each class and updates the base learners whenever a new example arrives. Furthermore, a novel under-sampling method is designed for handling the dynamic class-imbalance problem caused by gradually evolved classes.

In comparison to existing methods, CBCE can adapt well to all three cases of class evolution (i.e., emergence, disappearance and reoccurrence of classes). Since CBCE mines a data stream in an on-line manner, it is capable of rapidly keeping up with the gradual evolution of the data stream. Moreover, CBCE avoids maintaining a large size of base learners and makes it flexible to class evolution. Empirical studies verify the reliability of CBCE and show that it outperforms other state-of-the-art class evolution adaptation algorithms, not only in terms of the adaptation ability of various evolution scenarios but also the overall classification performance. However, CBCE still suffers from some drawbacks. For example, a disappearing class might be of less importance than non-evolved or emerging classes in some real-world applications. In such cases, since CBCE put more emphasis on evolved classes, its performance may decay on non-evolved classes. Besides, mining task for massive and complex evolved classes (e.g., minority classes with sub-concepts) is still difficult in data stream mining. A potential future work would be to expand CBCE to overcome these difficulties.

## Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61329302 and Grant 61175065, in part by the Program for New Century Excellent Talents in University under Grant NCET-12-0512, and in part by EPSRC grant (Grant No. EP/J017515/1). Xin Yao was also supported by a Royal Society Wolfson Research Merit Award. Ke Tang is the corresponding author of this paper.

## References

[1] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, “Mining data streams: A review,” *SIGMOD Rec.*, vol. 34, no. 2, pp. 18–26, 2005.
[2] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proc. 6th ACM SIGKDD Int. Conf. Know. Discovery Data Mining*, 2000, pp. 71–80.
[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014.
[4] L. Minku, A. White, and X. Yao, “The impact of diversity on online ensemble learning in the presence of concept drift,” *IEEE Trans. Know. Data Eng.*,vol. 22, no. 5, pp. 730–742, May 2010.
[5] L. Minku and X. Yao, “DDD: A new ensemble approach for dealing with concept drift,” *IEEE Trans. Know. Data Eng.*, vol. 24, no. 4, pp. 619–633, Apr. 2012.
[6] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, “New ensemble methods for evolving data streams,” in *Proc. 15th ACM SIGKDD Int. Conf. Know. Discovery Data Mining*, 2009, pp. 139–148.
[7] J. Liu, X. Li, and W. Zhong, “Ambiguous decision trees for mining concept-drifting data streams,” *Pattern Recog. Lett.*, vol. 30, no. 15, pp. 1347–1355, 2009.
[8] Z.-H. Zhou and Z.-Q. Chen, “Hybrid decision tree,” *Know.-Based Syst.*, vol. 15, no. 8, pp. 515–528, 2002.
[9] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, “Integrating novel class detection with classification for concept-drifting data streams,” in *Proc. Eur. Conf. Mach. Learn. Know. Discovery Databases*, 2009, vol. 5782, pp. 79–94.

[10] M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham, "Addressing concept-evolution in concept-drifting data streams," in *Proc. IEEE 10th Int. Conf. Data Mining,* Dec. 2010, pp. 929–934.

[11] M. Masud, T. Al-Khateeb, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham, "Detecting recurring and novel classes in concept-drifting data streams," in *Proc. IEEE 11th Int. Conf. Data Mining,* Dec. 2011, pp. 1176–1181.

[12] M. Muhlbaier, A. Topalis, and R. Polikar, "Learn$^{++}$.NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 152–168, Jan. 2009.

[13] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Trans. Know. Data Eng.,* vol. 23, no. 6, pp. 859–874, Jun. 2011.

[14] T. Al-Khateeb, M. Masud, L. Khan, C. Aggarwal, J. Han, and B. Thuraisingham, "Stream classification with recurring and novel class detection using class-based ensemble," in *Proc. IEEE 12th Int. Conf. Data Mining*, Dec. 2012, pp. 31–40.

[15] G. Widmer and M. Kubat, " Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, 1996.

[16] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. SIAM Int. Conf. Data Mining*, 2007, pp. 443–448.

[17] J. Gama, *Knowledge Discovery from Data Streams*, 1st ed. Boca Raton, FL,USA: CRC Press, 2010.

[18] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proc. 7th ACM SIGKDD Int. Conf. Know. Discovery Data Mining*, 2001, pp. 377–382.

[19] M. Karnick, M. Ahiskali, M. Muhlbaier, and R. Polikar, "Learning concept drift in nonstationary environments using an ensemble of classifiers based approach," in *Proc. IEEE Int. Joint Conf. Neural Netw.,* Jun. 2008, pp. 3455–3462.

[20] N. Oza, "Online bagging and boosting," in *Proc. IEEE Int. Conf. Syst., Man Cybern.,* Oct. 2005, vol. 3, pp. 2340–2345.

[21] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Trans. Neural Netw. Learning Syst.,*vol. 25, no. 1, pp. 81–94, Jan 2014.

[22] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proc. Adv. Artif. Intell. – SBIA 2004*, 2004, vol. 3171, pp. 286–295.

[23] M. Baena-Garca, J. D. Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early drift detection method," in *Proc. 4th ECML PKDD Int. Workshop Know. Discovery Data Streams*, 2006, pp. 77–86.

[24] S. Ramamurthy and R. Bhatnagar, "Tracking recurrent concept drift in streaming data using ensemble classifiers," in *Proc. 6th Int. Conf. Mach. Learning Appl.*, Dec. 2007, pp. 404–409.

[25] J. Gama and P. Kosina, "Recurrent concepts in data streams classification," *Know. Inform. Syst.*, vol. 40, no. 3, pp. 489–507, 2014.

[26] S. Sripirakas and R. Pears, "Mining recurrent concepts in data streams using the discrete fourier transform," in *Proc. 16th Int. Conf. Data Warehousing Knowl. Discovery*, 2014, pp. 439–451.

[27] M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, A. Srivastava, and N. Oza, "Classification and adaptive novel class detection of feature-evolving data streams," *IEEE Trans. Know. Data Eng.*, vol. 25, no. 7, pp. 1484–1497, Jul. 2013.

[28] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, 1988.

[29] R. Polikar, L. Upda, S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.

[30] G. Ditzler, M. Muhlbaier, and R. Polikar, "Incremental learning of new classes in unbalanced datasets: Learn$^{++}$.UDNC," in *Proc. 9th Int. Conf. Multiple Classifier Syst.*, 2010, pp. 33–42.

[31] G. Ditzler, G. Rosen, and R. Polikar, "Incremental learning of new classes from unbalanced data," in *Proc. Int. Joint Conf. Neural Netw.,* Aug. 2013, pp. 1–8.

[32] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Proc. 2nd Annu. Eur. Conf. Comput. Learn. Theory*, 1995, vol. 904, pp. 23–37.

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jan. 2002.

[34] J. Gao, B. Ding, W. Fan, J. Han, and P. Yu, "Classifying data streams with skewed class distributions and concept drifts," *IEEE Internet Comput.*, vol. 12, no. 6, pp. 37–49, Nov. 2008.

[35] S. Wang, L. Minku, and X. Yao, "A learning framework for online class imbalance learning," in *Proc. IEEE Symp. Comput. Intell. Ensemble Learn.*, Apr. 2013, pp. 36–45.

[36] S. Wang, L. L. Minku, and X. Yao, "Online class imbalance learning and its applications in fault detection," *Int. J. Comput. Intell. Appl.*, vol. 12, no. 4, pp. 1340001(19 pages), 2013.

[37] J. Kivinen, A. Smola, and R. Williamson, "Online learning with kernels," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2165–2176, Aug. 2004.

[38] N. Japkowicz, "Concept-learning in the presence of between-class and within-class imbalances," in *Proc. 14th Biennial Conf. Can. Soc. Comput. Stud. Intell.: Adv. Artif. Intell.*, 2001, pp. 67–77.

[39] T. Jo and N. Japkowicz, "Class imbalances versus small disjuncts," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 40–49, Jun. 2004.

[40] P. Mallapragada, R. Jin, and A. Jain, "Non-parametric mixture models for clustering," in *Proc. Int. Conf. Struct., Syntactic, and Statistical Pattern Recog.*, 2010, vol. 6218, pp. 334–343.

[41] K. Bache and M. Lichman. (2013). UCI machine learning repository [Online]. Available: http://archive.ics.uci.edu/ml

[42] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang, "Towards social user profiling: Unified and discriminative influence model for inferring home locations," in *Proc. 18th ACM SIGKDD Int. Conf. Know. Discovery Data Mining*, 2012, pp. 1023–1031.

[43] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Trans. Know. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[44] S. Wang, L. Minku, and X. Yao, "Resampling-based ensemble methods for online class imbalance learning," *IEEE Trans. Know. Data Eng,* vol. 27, no. 5, pp. 1356–1368, May 2015.

**Yu Sun** received the BEng degree in software engineering from the Dalian University of Technology (DLUT), Dalian, Liaoning, China, in 2010, and the MEng degree in software engineering from the University of Science and Technology of China (USTC), Hefei, Anhui, China, in 2013. He is currently working toward the PhD degree in computer science with the USTC-Birmingham Joint Research Institute in Intelligent Computation and its Applications (UBRI), School of Computer Science and Technology, USTC. His current research concerns incremental learning and data stream mining. He is a student member of the IEEE.

**Ke Tang** received the BEng degree from the Huazhong University of Science and Technology, Wuhan, China, in 2002, and the PhD degree from the Nanyang Technological University, Singapore, in 2007, respectively. Since 2007, he has been with the School of Computer Science and Technology, University of Science and Technology of China, where he is currently a professor. He has authored/coauthored more than 100 refereed publications. His major research interests include evolutionary computation, machine learning, and their real-world applications. He is an associate editor of the *IEEE Transactions on Evolutionary Computation*, *IEEE Computational Intelligence Magazine*, and *Computational Optimization and Applications* (Springer), and served as a member of editorial boards for a few other journals. He received the Royal Society Newton Advanced Fellowship. He is a member of the IEEE Computational Intelligence Society (CIS), Evolutionary Computation Technical Committee and the IEEE CIS Emergent Technologies Technical Committee. He is a senior member of the IEEE.

**Leandro L. Minku** received the PhD degree in computer science from the University of Birmingham, United Kingdom in 2010. He is currently a lecturer (assistant professor) in the Department of Computer Science, University of Leicester, United Kingdom. Prior to that, he was a research fellow at the University of Birmingham, United Kingdom. During his PhD, he received the Overseas Research Students Award (ORSAS) from the British government. He was also invited to a six-month internship at Google in 2009/2010. His main research interests are machine learning in nonstationary environments / data stream mining, ensembles of learning machines and computational intelligence for software engineering. His work has been published in internationally renowned venues such as the *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Software Engineering*, and *ACM Transactions on Software Engineering and Methodology*. He is a member of the IEEE.

**Shuo Wang** received the BSc degree in computer science from the Beijing University of Technology (BJUT), China, in 2006, and was a member of Embedded Software and System Institute in BJUT in 2007. She received the PhD degree in computer science from the University of Birmingham, United Kingdom, in 2011, sponsored by the Overseas Research Students Award (ORSAS) from the British Government (2007). She is a research fellow at the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA) in the School of Computer Science, the University of Birmingham United Kingdom. Her research interests include class imbalance learning, ensemble learning, online learning, and machine learning in software engineering. Her work has been published in internationally renowned journals and conferences. She is a member of the IEEE.

**Xin Yao** is a professor of computer science and the director in the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA) at the University of Birmingham, United Kingdom. He was the president (2014-2015) in IEEE Computational Intelligence Society (CIS). His major research interests include evolutionary computation and ensemble learning, especially online learning and class imbalance learning. His work received the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 and 2015 IEEE Transactions on Evolutionary Computation Outstanding Paper Awards, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards. He received the prestigious Royal Society Wolfson Research Merit Award in 2012 and the IEEE CIS Evolutionary Computation Pioneer Award in 2013. He was the Editor-in-Chief (2003-2008) of the *IEEE Transactions on Evolutionary Computation*. He is a fellow of the IEEE and a distinguished lecturer in the IEEE CIS.