World Wide Web (2017) 20:1507–1525
DOI 10.1007/s11280-017-0449-x

Multi-window based ensemble learning for classification of imbalanced streaming data

Hu Li[1] · **Ye Wang**[1,2] · **Hua Wang**[2] · **Bin Zhou**[1,3]

Received: 20 June 2016 / Revised: 16 February 2017 /
Accepted: 23 February 2017 / Published online: 8 March 2017

Abstract Imbalanced streaming data is commonly encountered in real-world data mining and machine learning applications, and has attracted much attention in recent years. Both imbalanced data and streaming data in practice are normally encountered together; however, little research work has been studied on the two types of data together. In this paper, we propose a multi-window based ensemble learning method for the classification of imbalanced streaming data. Three types of windows are defined to store the current batch of instances, the latest minority instances, and the ensemble classifier. The ensemble classifier consists of a set of latest sub-classifiers, and the instances employed to train each sub-classifier. All sub-classifiers are weighted prior to predicting the class labels of newly arriving instances, and new sub-classifiers are trained only when the precision is below a predefined threshold. Extensive experiments on synthetic datasets and real-world datasets demonstrate that the new approach can efficiently and effectively classify imbalanced streaming data, and generally outperforms existing approaches.

Keywords Streaming data · Class imbalance · Multi-window · Ensemble learning

✉ Hu Li
lihu@nudt.edu.cn

Ye Wang
ye.wang10@live.vu.edu.au

Hua Wang
hua.wang@vu.edu.au

Bin Zhou
binzhou@nudt.edu.cn

[1] College of Computer, National University of Defense Technology, Changsha 410073, China

[2] Centre for Applied Informatics, Victoria University, Melbourne, Victoria, Australia

[3] State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China

1 Introduction

Classification is one of the most important problems in the fields of data mining and machine learning, and has attracted much attention in recent years. In conventional methods of solving the classification problem, a classifier is generally trained on a dataset that does not change over time. As such, the dataset is assumed to have all the information required to learn the underlying concepts. However, in many real-world scenarios, including spam filtering [8], credit card fraud identification [25], intrusion detection [18], and webpage classification [26], datasets are not static. New instances may arrive one by one or batch by batch, and incoming instances must be classified within a finite period with finite resources. Regardless of whether new instances arrive incrementally or in batch, only data received prior to the time step t can be used to train the classifier and predict the instances arriving at the time step $t + 1$. In other words, the classifier can only be trained on an incomplete portion of the information.

The problem of class imbalance can be very common in streaming data. For instance, in credit card fraud identification, a small minority of customers typically engage in fraud. Thus, class imbalance is a condition of data streams that cannot be ignored when dealing with real-world problems. Because we are more interested in rare class instances, these are usually denoted as positive instances and majority instances are denoted as negative instances.

In an earlier stage of development, classification of streaming data and imbalance problems were studied separately, although, in recent years, increasing attention has been dedicated to addressing these two problems together. However, as discussed elsewhere [12], most research efforts have combined algorithms designed for streaming and imbalanced data in a relatively simple manner, which are almost equivalent to address these two problems separately.

In our earlier work [29], we analyzed this problem in a novel way and proposed a method using multi-window ensemble learning (MWEL) for classification of imbalanced streaming data. In this paper, we extend MWEL in several ways. First of all, a more carefully designed multi-window ensemble learning framework is presented, in which four different kinds of windows are used to model the imbalanced data stream. Secondly, two different window update policies correspond to minority window and sub-classifier window are designed in detail. Thirdly, we improved majority weighted voting approach so that it can deal with more complicated cases. In addition, we conduct experiments on more datasets from different domains to further analyze impact of parameters used in our algorithms. Finally, we compare our improved approach with others methods under 8 different evaluation criteria. The main contributions of the present work are as follows:

- A multi-window framework is proposed to record the current batch of instances, selected positive instances, and the ensemble classifier along with the corresponding instances used to train each sub-classifier. The framework enables us to accumulate the latest positive instances, and enhances the weight of the positive instances accordingly. Furthermore, concept drift in data streams can be detected by the error rate together with similarities between the current window and the history windows used to train each sub-classifier.
- A novel ensemble learning mechanism is designed to classify the incoming instances. The weight of each sub-classifier is determined by the classification error rate and the window similarity. New instances are classified using weighted majority voting. Adjusting the weight according to the error rate can improve the classification accuracy, and weight adjustment based on window similarity can solve the reoccurring concept drift issue [12] to a certain extent.

- Extensive experiments on both synthetic datasets and real-world datasets in different application domains are conducted, from which optimal parameters for each dataset are obtained, and the proposed approach is demonstrated to generally outperform alternative methods.

To simplify the model, we focus only on single-label two-class classification problems, where each instance can belong only to a single class and only two classes are considered. Single-label two-class classification tasks are found in many real-world applications, for example, email messages can be classified into spam and normal messages; credit card users can be labeled as fraudulent and regular users. Moreover, multi-label classification problems can be divided into several single-label two-class classification problems [20]. Multi-class problems can also be subjected to divide and conquer strategies [9, 17, 19]. In addition, we mainly focus on low dimension problems because high dimension problems can always be reduced, as discussed in [21].

The remainder of the paper is organized as follows. Works related to streaming data classification and imbalanced classification are presented in Section 2. Section 3 proposes the multi-window based ensemble learning approach and describes it in detail. The experiments on both real-world and synthetic datasets, and the discussion of the results are provided in Section 4, followed by a conclusion and outlook in Section 5.

2 Related work

Classification of streaming data has attracted much attention for some time, and has been widely studied, particularly with problems related to concept drift. Meanwhile, numerous researchers have focused on the imbalanced data classification problem. In recent years, an increasing number of scholars have addressed problems involving both class imbalance and concept drift. We briefly review these works in the following.

2.1 Classification of streaming data

The attribute values of newly incoming instances in streaming data may vary over time, resulting in concept drift. Previous methods employed to mitigate this problem can be roughly divided into three categories: adaptive-based learning, the training set modification method, and ensemble learning.

Adaptive-based learning attempts to modify existing classifiers to provide adaptability to the occurrence of concept drift in streaming data. Based on very fast decision tree (VFDT), Hulten et al. proposed the concept-adaptive VFDT learner (CVFDT) [13], which maintains an up-to-date classifier by computing new split attributes and comparing them with old attributes using a sliding window. Other works based on VFDT include the Hoeffding window tree (HWT) and Hoeffding adaptive tree (HAT) [3]. Both of these methods aim at quickly adapting to concept drift in different ways. HWT re-computes split attributes once a new instance arrives rather than waiting for full window-sized instances, and HAT employs an adaptive window.

The training set modification method aims at adjusting the relative weights of different class instances to fit the target concept. Because the method is independent of existing classifiers, it is widely used in many applications. This type of method consists of windowing and weighting techniques. In the windowing techniques, the data stream is segmented into

windows, and a classifier is trained on each window. The easiest approach is to employ a fixed window size, although an optimal window size is difficult to determine. Thus, the self-adaptive method FLORA3 [30] was proposed to fit the present concept using a variable window size. Also, ADWIN [2] was proposed to determine the optimal window size by calculating the similarity between two different windows. On the other hand, the weighting techniques impart different weights to different class instances. For example, all instances are weighted according to their distance from the current concept [1].

Ensemble learning integrates the results from multiple sub-classifiers to obtain better performance than a single classifier. For example, SEA [22] trains a sub-classifier on each non-overlapping window. A newly trained sub-classifier is added to the ensemble classifier or replaces the worst sub-classifier depending on a predefined maximum ensemble size. Other works [10, 14] weight each classifier according to its performance (e.g., error rate).

2.2 Classification of imbalanced datasets

As discussed above, when dealing with imbalanced datasets, we are generally more concerned with the minority class. Therefore, the minority class must be emphasized. Related works can be divided into three categories depending upon the level at which the method functions [4, 11]. The first category functions at the data-level, which seeks to balance the size of each class by increasing or decreasing the number of minority or majority instances. In this case, the minority instances can simply be oversampled or the majority instances can be under-sampled randomly, although some heuristic methods have been proven to be more stable, e.g., SMOTE [5], which creates synthetic minority instances rather than duplicating instances to avoid over-fitting, and Tomek-line [24], which is used to locate redundant majority instances. The second category functions at the algorithm level, which attempts to modify a specific algorithm, e.g., CCPDT [16]. The third category functions on the basis of ensemble learning, which is widely used to handle imbalanced datasets, e.g., SMOTEBoost [6] and AdaC1, AdaC2, and AdaC3 [23].

2.3 Classification of imbalanced streaming data

The boundary definition (BD) approach [15] was proposed to build classifiers based on boundary instances, which are more easily misclassified. The approach divides the majority instances into a correctly classified set and a misclassified set. Random under-sampling is performed on each set separately to guarantee distribution consistency. Eleftherios et al. proposed maintaining two windows to record the positive and negative classes separately for multi-label stream classification problem [31]. A learning framework for an online imbalanced classification problem, including an imbalanced class detector, a concept shift detector, and an online learner, was proposed [27]. The researchers also proposed the so-called oversampling-based online bagging (OOB) and under-sampling-based online bagging (UOB) methods to improve classification accuracy. In the work, concept drift was simplified to a change of the imbalance ratio among classes. Thereafter, the authors proposed sampling-based online bagging (SOB) [28], which aimed at maximizing the G-mean and balanced classes by adjusting the parameter λ of the Poisson distribution in onling bagging. Recently, a selective re-training approach based on clustering has been proposed [32]. In this work, a new sub-classifier is trained when new data arrives, and, if the overall classification performance is unsatisfied, the new sub-classifier replaces the worst performing sub-classifier. Because a new classifier is trained each time a new instance arrives, the method is computationally complex.

3 Framework

3.1 Problem definition

Assume that, at time step t, the existing instances form a data sequence $D = \{(X_0, l_0), (X_1, l_1), \ldots, (X_t, l_t)\}$ in chronological order, where X_i is a d-dimensional vector and corresponds to a class label l_i. All class labels constitute a label sequence $L = \{l_1, l_2, \ldots, l_t\}$. In a two-class classification task, $l_i \in \{+, -\}$, where '+' and '−' represent positive (minority) and negative (majority) classes, respectively. The goal is to train a single classifier or set of classifiers on existing instances so that the classifier(s) can accurately predicate the incoming instance X_{t+1} at time step $t+1$, and, once X_{t+1} is predicated, we are aware of the true label of X_{t+1}, i.e., l_{t+1}. In addition, we take the dataset as imbalanced if the ratio of the number of different class instances, i.e., $\#\{(X_i, l_i) | l_i = '+'\} / \#\{(X_j, l_j) | l_j = '-'\}$, $i, j = 0, 1, \ldots, t$, exceeds a predefined threshold. The task is to build the classifier(s) on an imbalanced data stream and obtain acceptable results.

In the present study, we adopt a windowing or batching approach, where the window size is pre-calculated by experiments, and then fixed. The sliding window moves forward once window-sized instances have arrived. Here, the windows on the data stream are non-overlapping and in chronological order. Suppose the window size is M_b, then, at time step t, existing instances are divided into $\lceil t/M_b \rceil$ windows or batches. When $M_b = 1$, the batching approach is equivalent to incremental learning, where a single instance is processed at a time.

3.2 Multi-window mechanism

In this section, we describe our multi-window ensemble learning algorithm in detail. First of all, we must determine the size of the sliding window (*WB*; window of batch). An overly small window cannot adequately represent the class characteristics, and may result in a classifier with poor generalization. On the other hand, an overly large window size will require much more acquisition time and resources, which are restricted in practical applications. For instance, a window may not have acquired sufficient instances within the limited time available after which a prediction is expected. Scant theoretical guidance exists for determining an optimal window size, which must be obtained through experiments.

Under imbalanced conditions, positive instances are sparse, and may even be absent from some sliding windows. As a result, classifiers trained on these windows may be unable to represent the positive class. Therefore, we use a minority window (*WM*; window of minority) to store newly incoming minority or positive instances. Minority window method has also been used in the BD approach [15], although the *WM* employed in BD consumed excessive time and resources; moreover, early acquired minority instances ran the risk of becoming meaningless due to concept drift. We adopt a similar strategy to that used elsewhere [7], which fixes the minority window size and adds only minority instances residing near the current positive class into the *WM*. However, it is time consuming to select only the nearest instances when the window size is very large, and the nearest instances may not be of the same concept. Consequently, we utilize a simple substitution policy here with fixed size, and add minority instances into the *WM* prior to reaching its upper size limit; otherwise, the oldest instance will be replaced by the newest. Thus, the *WM* always represents up-to-date positive instances over time.

In addition, we use a classifier window (*WC*; window of classifier) to preserve the requisite number of newly trained sub-classifiers and their corresponding weights WC_{weight}, as well as the windows of instances WC_{ins}used to train each sub-classifier. The size of *WC* is determined in advance through experiments, and is fixed for streaming future instances. Newly trained sub-classifiers are added to *WC* prior to reaching the predefined size; otherwise, the oldest sub-classifier is replaced.

3.3 *WC* update policy

Because accuracy is not always a reliable metric for imbalanced data (for example, given 99 normal email messages and 1 spam in the dataset, the classifier can treat all 100 messages as normal and still obtain an accuracy of 99%), it is necessary to evaluate the classification result for each class simultaneously under imbalanced conditions. In the present study, we use the precision of both the minority and majority classes to evaluate the classification performance. If the precisions of both the majority class $Precision_{maj}$ and minority class $Precision_{min}$of the newly trained classifier are greater than 0.5, indicating better than random guess, we add it into the *WC*. In addition, WC_{weight} is set as the current accuracy, and the current window instances used to train the new classifier are saved in the WC_{ins}. Otherwise, failing the precision test, the classifier is discarded.

3.4 Multi-window ensemble learning

The overall processing conducted by our Multi-window ensemble learning (MWEL) is shown in Algorithm 1. When the first window arrives, the *WC* and *WM* are initially empty, and the first classifier is trained on the current window. Then, it is determined whether or not to add the classifier to the *WC* by the metric described in Subsection 3.3, which is given by **Algorithm 1**. Simultaneously, we update the *WM* according to rules described in Subsection 3.2, which is given by **Algorithm 3**.

When the next window arrives, we must update the weights of the sub-classifiers to fit the concept in the window. We firstly compute and normalize the similarity between the current window and each of the existing windows that correspond to existing sub-classifiers to modify the weights based on the following observation.

Observation 1: The greater the similarity between window W_1 and window W_2, the closer are the corresponding concepts within W_1 and W_2. Therefore, those sub-classifiers trained on windows more similar to the current window should be given larger weights as follows.

$$WC_{\text{weight}\cdot i} = WC_{\text{weight}\cdot i}/(1-sim(WB, WC_{ins\cdot i}) + \lambda) \;\; i = 1, 2, \ldots, M_c \tag{1}$$

Here, $WC_{weight\cdot i}$ is the weight of the *i*th sub-classifier in the *WC*, *WB* is the current sliding window, $WC_{ins\cdot i}$ is the window of instances corresponding to the *i*th sub-classifier in *WC*, $sim(WB, WC_{ins\cdot i})$ is the similarity between the current window and $WC_{ins\cdot i}$, and λ is a constant. Moreover, this observation can be used to address reoccurring concept drift [12] because an existing sub-classifier corresponding to a reoccurring concept will obtain a larger weight, which is expected to provide better results.

For each instance in the *WB*, we use a majority weighted voting method to predicate the class label as follows.

$$l'_{\text{newIns}} = \begin{cases} 1, & \sum_{i=1}^{|WC|} WC_{\text{weight}\cdot i} \cdot WC_i(\text{newIns}) > 0.5 \\ 0, & \text{else} \end{cases} \tag{2}$$

Here, $WC_i(newIns)$ denotes the classifying instance *newIns* accompanying the *i*th sub-classifier in the *WC*. The classification result can be the probability with which an instance belongs to a class or an exact result of 0 or 1 corresponding to a majority or minority instance, respectively.

Algorithm 1. Multi-window ensemble learning (MWEL)

Input: ordered instances $D = \{(X_0, l_0), (X_1, l_1), ..., (X_t, l_t)\}$;
real class labels $L = \{l_0, l_1, ..., l_t\}$;
maximum size of batch M_b;
maximum size of classifier ensemble M_c;
maximum size of minority window M_m.

Output: window of minority instances WM;
window of classifiers WC;
window of weight of classifiers WC_{weight};
window of instances used to train each classifier WC_{ins};
predicted labels $L' = \{l'_0, l'_1, ..., l'_t\}$.

Initialize: window of batch of instances $WB = \{\}$;
window of minority instances $WM = \{\}$;
window of classifiers $WC = \{\}$;
$WC_{weight \cdot i} = 1,\ i = 0, 1, ..., M_c$.

```
1: FOR each ordered instance in D
2:     WB ← getBatchOfInstances(M_b)
3:     IF |WC| = 0 THEN
4:         c ← trainClassifier(WB)
5:         {precision_min, precision_maj, errorRate} ← classify(WB, c)
6:         IF precision_min > 0.5 && precision_maj > 0.5 THEN
7:             updateClassifierWindow(WC, c, WB, errorRate)
8:         ELSE
9:             WB_s ← sample(WB)
10:            WB_s ← WB_s ∪ WM
11:            c' ← trainClassifier(WB_s)
12:            {precision_min, precision_maj, errorRate} ← classify(WB, c')
13:            IF precision_min > 0.5 && precision_maj > 0.5 THEN
14:                updateClassifierWindow(WC, c', WB, errorRate)
15:            END IF
16:        END IF
17:    ELSE
18:        FOR each window instances in WC_ins
19:            sim ← getSimScore(WB, WC_ins·i)
20:            WC_weight·i = WC_weight·i / (1 − sim(WB, WC_ins·i) + λ)
21:        END FOR
```

22: $\{precision_{\min}, precision_{maj}, errorRate\} \leftarrow classify(WB, WC)$
23: IF $precision_{\min} < 0.5 \,||\, precision_{maj} < 0.5$ THEN
24: $WB_s \leftarrow sample(WB)$
25: $WB_s \leftarrow WB_s \cup WM$
26: $c' \leftarrow trainClassifier(WB_s)$
27: $\{precision_{\min}, precision_{maj}, errorRate\} \leftarrow classify(WB, c')$
28: IF $precision_{\min} > 0.5$ && $precision_{maj} > 0.5$ THEN
29: $updateClassifierWindow(WC, c')$
30: END IF
31: END IF
32: END IF
33: $WM \leftarrow updateMinorityWindow(WM, WB)$
34:END FOR

The *WC* is comprised of sub-classifiers, the history of the window of instances used to train each sub-classifier, and the weights of each sub-classifier. All three components are updated simultaneously, and the primary procedure is described in **Algorithm 2**.

Algorithm 2. Update the *WC* (updateClassifierWindow)

Input: classifier window WC;
new classifier c';
maximum size of classifier ensemble M_c;
error rate $errorRate$;
window of batch of instances WB.
Output: updated classifier window WC;
updated weight of classifiers WC_{weight};
updated window of instances used to train each sub-classifier WC_{ins}.

1:IF $|WC| < M_c$ THEN
2: $WC_{|WC|} \leftarrow c'$
3: $WC_{weight \bullet |WC|} \leftarrow 1 - errorRate$
4: $WC_{ins \bullet |WC|} \leftarrow WB$
5:ELSE
6: FOR $i = 0, 1, ..., M_c - 2$
7: $WC_i \leftarrow WC_{i+1}$
8: $WC_{weight \bullet i} \leftarrow WC_{weight \bullet i+1}$
9: $WC_{ins \bullet i} \leftarrow WC_{ins \bullet i+1}$
10 END FOR
11: $WC_{M_c - 1} \leftarrow c'$
12: $WC_{weight \bullet M_c - 1} \leftarrow 1 - errorRate$
13: $WC_{ins \bullet M_c - 1} \leftarrow WB$
14:END IF

As show in line 33 of **Algorithm 1**, the *WM* is updated at the end of each window. Here, to save time and resources, we simplified the procedure and saved only the latest minority

instances rather than store all minority instances [21] or select the nearest instance to the current window [1]. The primary procedure is described in **Algorithm 3**.

Algorithm 3. Update the WM (updateMinorityWindow)

Input: minority window WM ;
window of batch instances WB .
Output: updated minority window WM'.
1: FOR each instance $wb \in WB$
2: IF $classLabel(wb) = '+'$ THEN
3: IF $|WM| < M_m$ THEN
4: $WM_{|WM|} \leftarrow wb$
5: END IF
6: IF $|WM| = M_m$ THEN
7: FOR $i = 0,1,...,M_m - 2$
8: $WM_i \leftarrow WM_{i+1}$
9: END FOR
10: $WM_{M_m-1} \leftarrow wb$
11: END IF
12: END IF
13:END FOR

Because an imbalanced condition can exist at any time, it must be detected and addressed at all times. We compare the predicated label with the true label of each instance in the current window. If one or both of $Precision_{maj}$ and $Precision_{min}$ is less than 0.5, an imbalance is very likely to exist within the current sliding window, and all instances within the window will be resampled. The sampling procedure is presented in **Algorithm 4**. Rightly classified positive instances remain unchanged because they are not likely to be of assistance in increasing the precision, while wrongly classified positive instances are oversampled to increase their weights when used to train a new sub-classifier. Here, SMOTE [5] is used to oversample minority instances and random under-sampling is applied on correctly classified majority instances. After several iterations, the imbalance ratio gradually decreases to the predefined threshold. Finally, correctly classified minority instances, wrongly classified majority instances, oversampled wrongly classified minority instances, and under-sampled correctly classified majority instances are combined together as the new training set.

Algorithm 4. The sampling procedure

Input: batch of instances WB ;
expected imbalance rate IR .
Output: sampled instances WB_s .
1: $\{correctClassifiedMinority, wrongClassifiedMinority, correctClassifiedMajority, wrongClassifiedMajority\} \leftarrow getSubset(WB)$
2:WHILE $|Minority| / |Majority| < IR$
3: $sampledMinority \leftarrow SMOTE(wrongClassifiedMinority)$
4: $sampledMajority \leftarrow randomUnderSample(correctClassifiedMajority)$
5:END WHILE
6: $WB_s = correctClassifiedMinority \cup sampledMinority \cup wrongClassifiedMajority \cup sampledMajority$

Note that the new classifier will receive the largest weight of 1 by default on the basis of the observation below.

Observation 2: Considering the influence of time factor, the latest classifier is more likely to accord with the concept of the current window, and, therefore, this classifier deserves a larger weight.

In general, the factors of time, similarity between windows and the precision of sub-classifiers are all considered by our method, which should be more reasonable and comprehensive.

Regarding the ensemble classifier, it is very important to increase its diversity to enhance its robustness and performance. However, if the current window exhibits little concept drift, using this window to train a new sub-classifier brings no diversity, but does add to the computational burden. Therefore, we assume that little benefit is gained by building a new sub-classifier on each sliding window. In contrast, our approach builds new sub-classifiers only when $Precision_{maj}$ and/or $Precision_{min}$ is less than 0.5, which ensures diversity, and reduces the computational load.

4 Experiment

We evaluated our approach on both real-world and synthetic datasets using a variety of metrics. Experiments were initially conducted to obtain optimal window sizes for the different datasets. The results of the proposed method were compared with those of two existing approaches.

4.1 Datasets

As shown in the first eight rows of Table 1, we conducted experiments on eight real-world datasets. The Elec, Forest, Airlines, Poker1, and Pocker2 datasets are publicly available at MOA datasets.[1] The Mushroom, Thyroid1, and Thyroid2 datasets are publicly available at the UCI Machine Learning Repository.[2]

- Elec (*ele*) was collected from the Australian New South Wales electricity market to predict the rise and fall of electricity prices, where prices are affected by market supply and demand, and are set every five minutes.
- Forest (*for*) contains the forest cover type for 30 × 30 m cells obtained from the US Forest Service with 7 classes in the original dataset. We extracted classes 4 and 2 as the minority and majority classes.
- Airlines (*air*) was used to predict whether or not a flight would be delayed.
- Mushroom (*mus*) includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms, where each species is identified as either edible or poisonous.
- Thyroid datasets (*th1*, *th2*) were used to identify whether or not a patient has thyroid disease. To form naturally imbalanced data, we selected class 1 and class 3 to form *th1* and class 2 and class 3 to form *th2*.

[1] http://moa.cs.waikato.ac.nz/datasets/
[2] http://archive.ics.uci.edu/ml/datasets.html

Table 1 Dataset statistics

ID	Name	#All instances	#Positive instances	#Negative instances	#Attributes	Positive index	Negative index	Imbalance rate
ele	Elec	45, 312	19, 237	26, 075	8	1	2	1:1.35
for	Forest	286, 048	2747	283, 301	54	4	2	1:103
air	Airlines	539, 383	240, 264	299, 119	7	2	1	1:1.24
mus	Mush	8, 124	3, 936	4, 188	23	1	2	1:1.06
th1	Thyroid1	6, 832	166	6, 666	21	1	3	1:40
th2	Thyroid2	7, 034	368	6, 666	21	2	3	1:18
po1	Poker1	454, 958	39, 706	415, 252	11	3	1	1:11
po2	Poker2	367, 967	17, 473	350, 494	11	4	2	1:21
gcd	GCD	100, 000	24, 652	75, 348	20	2	1	1:3
scd	SCD	100, 000	25, 178	74, 822	20	2	1	1:3
rcd	RCD	100, 000	24, 280	75, 720	20	2	1	1:3

- Poker datasets (*po1*, *po2*) consist of five playing cards drawn from a standard deck of 52 cards. Each card is described using its suit or rank.

Moreover, experiments were conducted on three types of synthetic datasets generated by algorithms provided in MOA[3] which is shown in Figure 1:

- Gradual concept drift (*gcd*), where a gradual concept drift begins at the 30,000th instance to the 100,000th instance.
- Sudden concept drift (*scd*) takes place suddenly at the 50,000th instance, and the dataset maintains the new concept until the 100,000th instance.
- Reoccurring concept drift (*rcd*) occurs at the 30,000th instance, and begins to shift back to the original concept at around the 50,000th instance until the 100,000th instance.

During the instance generation, we randomly removed some instances from one class to form imbalanced datasets.

4.2 Evaluation criteria

We employ accuracy, precision, recall, F1, and G-mean to evaluate our method in this paper. As discussed in Subsection 3.3, the accuracy may be skewed by the prominence of the negative class. However, $G\text{–mean} = \sqrt{\text{Recall}_{\text{min}} \times \text{Recall}_{maj}}$ considers the recalls of both the minority ($Recall_{min}$) and majority ($Recall_{maj}$) classes together, and can only be large when both $Recall_{min}$ and $Recall_{maj}$ are large, which is a better choice under imbalanced conditions.

When evaluating the classification performance on a data stream, we adopt a previously proposed strategy [15], which evaluates the classifier using the average performance over all batches in the data stream as follows.

$$F = \frac{1}{\lceil t/M_b \rceil} \sum_{i=1}^{\lceil t/M_b \rceil} f_i \qquad f, F \in \Big\{ \text{Accuracy}, \text{Precision}_{min}, \text{Precision}_{maj}, \text{Recall}_{min}, \text{Recall}_{maj}, F1_{\text{min}}, F1_{maj}, G\text{–mean} \Big\} \tag{3}$$

[3] http://moa.cms.waikato.ac.nz/

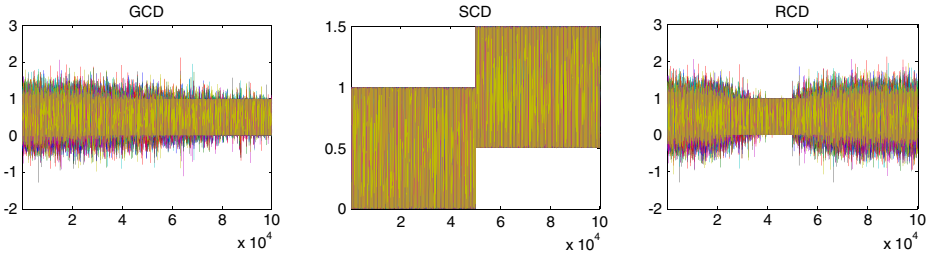

Figure 1 Synthetic datasets given in Table 1

Here, f is the indicator of each sliding window, and F is the average indicator of the data stream. We compared all the indicators in our experiments, but, owing to length restrictions, we report here only the results regarding accuracy, G-mean, $Recall_{min}$, and processing time. Many real-world applications are expected to accomplish necessary processing within a finite period, where a batch must at least be processed before the next batch arrives. Therefore, data streaming algorithms require a tradeoff between efficiency and effectiveness.

4.3 Sliding window size setup

As discussed in Subsection 3.2, no widely accepted standards are available for selecting an optimal sliding window size with regard to different types of datasets. A larger window size provides a smaller number of windows with respect to a specific dataset length, which reduces the frequency of classifier training, whereas, contrarily, a smaller window size introduces a greater number of windows with respect to a specific dataset length, increasing the frequency of classifier training. Moreover, a smaller window size also provides less training time for each classifier. Therefore, in present study, experiments were first conducted to determine the optimal sliding window size for different datasets.

Figure 2 shows how the sliding window size affects the classification results, and substantial differences in the various indicators are observed for different datasets. For instance, in Figure 2(a), the *for*, *th1*, *th2*, *po1*, and *po2* datasets exhibit quite high accuracy when the window size is 1000, and the accuracy remains relatively stable with increasing window size with the *for*, *th1*, *po1*, and *po2* datasets, but decreases slowly for the *th2* dataset.

However, as shown in Figure 2(b), the maximum G-mean value for the *for* dataset occurs at a window size of 500, whereas maximum values are obtained at 400 and 600 for the *th1* and *th2* datasets, respectively. By comparing the $Recall_{min}$ values shown in Figure 2(c), we find that, as the window size increases, the $Recall_{min}$ values for the *th1* and *th2* datasets, which lack positive instances, decline sharply, leading to decreasing G-mean values. This indicates that the density of minority instances becomes increasingly sparse with increasing window size. Furthermore, we note from Figure 2(d) that an increasing window size initially decreases the training and classifying time, but, for a window size greater than 1000, the time cost exhibits a general trend of slow growth for all datasets considered except *scd*. For a fixed data stream length, the processing time is the product of two parts: the training and classifying time of each window and the number of windows. The initial reduction in the processing time is due to the decreasing number of windows, whereas the longer processing time required for each window results in the general rise in later stages with increasing window size.

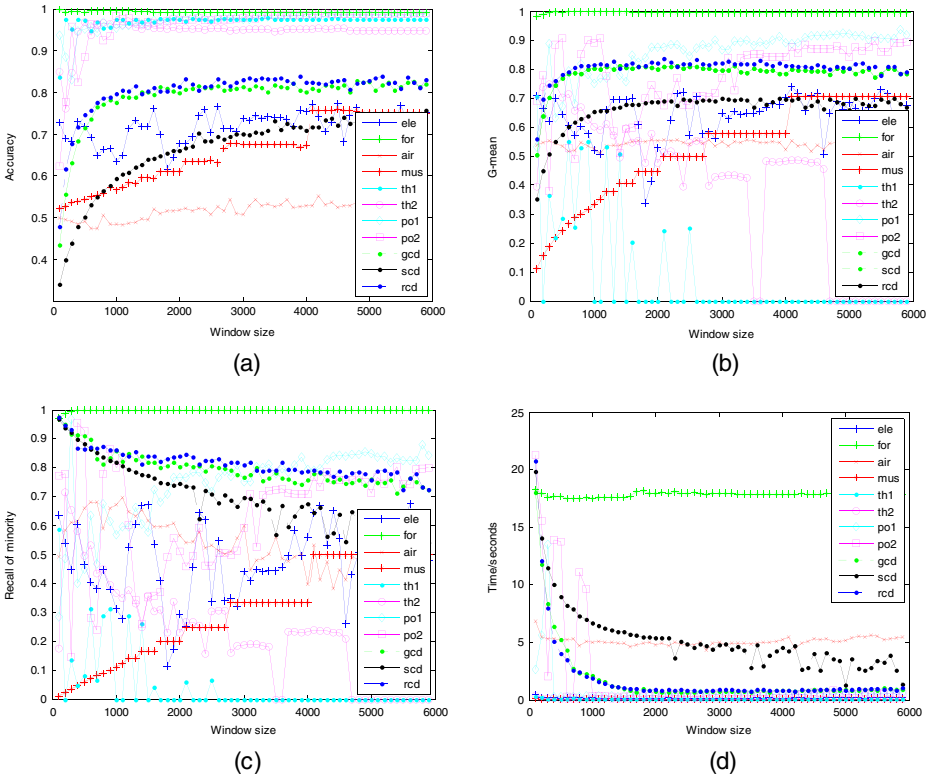

Figure 2 Sliding window size for different datasets

Based on considerations of the effectiveness and processing efficiency, we established the optimal window size for each dataset. The values are listed in Table 2, and the following experiments were conducted according to this standard.

It is necessary to point out that, in actual applications, it is unrealistic to calculate the optimal sliding window size in advance. As an alternative, the optimal sliding window size can be determined based on a small set of available instances. Here, we focus on examining the effect of window size on different datasets, and a self-adaptive optimal sliding window is reserved for future study.

4.4 Minority window size setup

We examined the influence of the minority window size on the classification performance, and present the results in Figure 3. The results indicate that most datasets are insensitive to the minority window size. However, the *mus*, *gcd*, *scd*, and *rcd* datasets exhibit a decreasing accuracy with increasing minority window size, as shown in Figure 3(a). As shown in

Table 2 Optimal sliding window sizes for different datasets

ID	*ele*	*for*	*air*	*mus*	*th1*	*th2*	*po1*	*po2*	*gcd*	*scd*	*rcd*
Window size	1300	1300	800	900	100	200	600	1000	1400	1400	1300

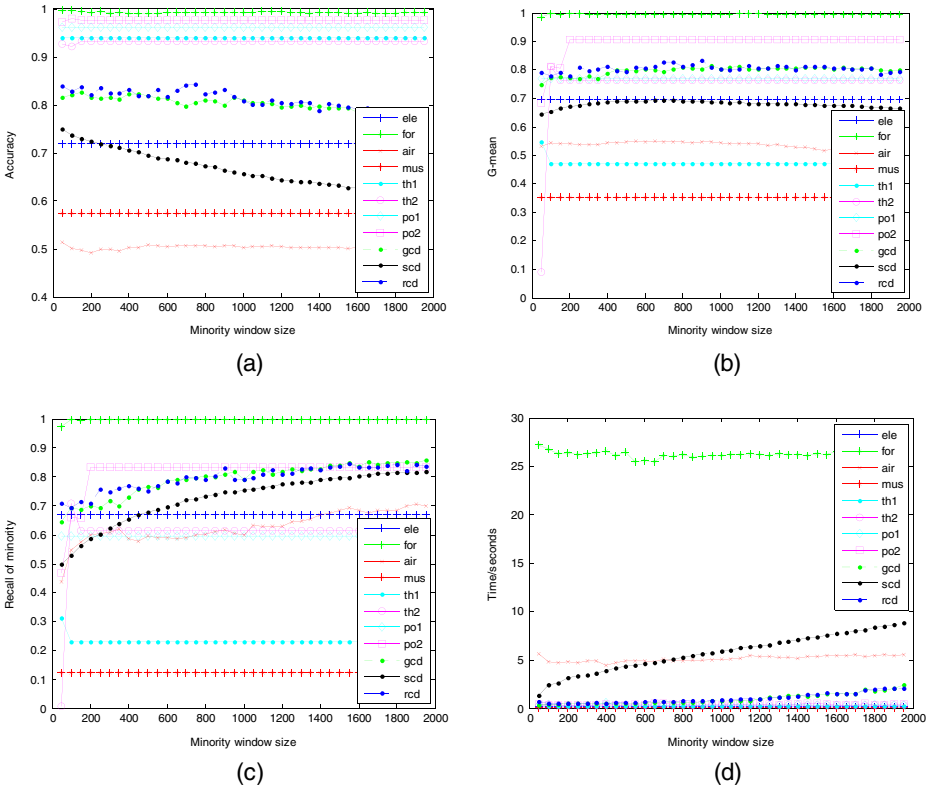

Figure 3 Minority window size for different datasets

Figure 3(b), the G-mean values of most datasets, except *air*, *gcd*, *scd*, and *rcd*, remain stable around a window size of 100. Because the minority window is designed to improve the probability of identifying positive class instances under imbalanced conditions, the results in Figure 3(c) demonstrate that an increasing window size increases the $Recall_{min}$ values for most datasets. However, the *for*, *ele*, *th1*, and *mus* datasets remain nearly unchanged regardless of the window size because minority instances within these four datasets are distributed more evenly than in the other datasets. In addition, the *ele* dataset has a low imbalance rate (1:1.35), and, thus, the window size has little influence on $Recall_{min}$.

Figure 3(d) indicates that the processing time typically increases as the minority window size increases, although the processing times of *th1* and *th2* remain nearly unchanged because the total numbers of minority instances within these two datasets are only 166 and 368. Based on considerations of effectiveness and efficiency, we established the optimal size of the minority window for each dataset, as listed in Table 3, and the following experiments were conducted according to this standard.

Table 3 Optimal minority window sizes for different datasets

ID	*ele*	*for*	*air*	*mus*	*th1*	*th2*	*po1*	*po2*	*gcd*	*scd*	*rcd*
Window size	500	700	700	100	300	300	100	200	800	800	900

4.5 Classifier window size setup

Because we employ an ensemble classifier and weighted majority vote, experiments were conducted to determine the optimal classifier window size, and the results are shows in Figure 4.

As shows in Figure 4(a), the accuracies of *th2* and *scd* initially increase with increasing classifier window size, and then remain stable, whereas *gcd* and *rcd* are observed to decrease, particularly for small window sizes. The other datasets however appear to be insensitive to the classifier window size. The G-mean values shown in Figure 4(b) exhibit very similar trends to those of the accuracy. However, the $Recall_{min}$ values showns in Figure 4(c) decrease slowly with increasing classifier window size, indicating that a large number of sub-classifiers is not always a good choice. The processing time consumed with respect to the classifier window size is show in Figure 4(d). Based on considerations of effectiveness and efficiency, the optimal classifier window sizes for most of the datasets considered are less than 6.

4.6 Comparison with existing methods

The performance of the proposed MWEL (MW) method was compared with those of two existing approaches, denoted as the BD [15] and CS [32] approaches. The authors claimed that their approaches performed better than those to which they were compared, but these two

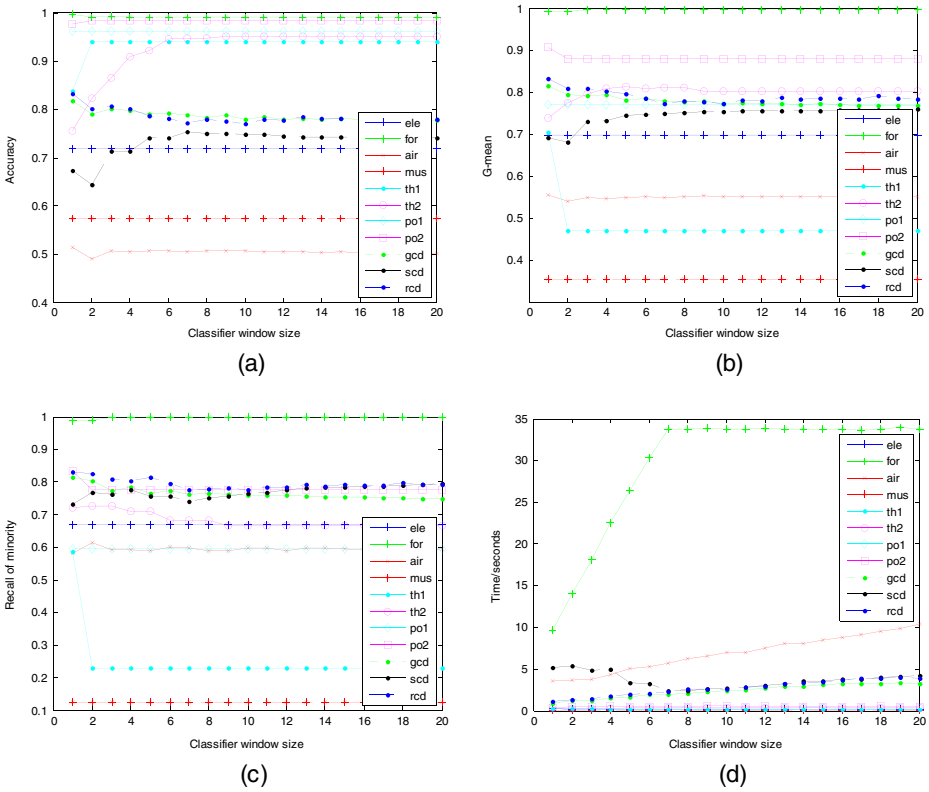

Figure 4 Classifier window size for different datasets

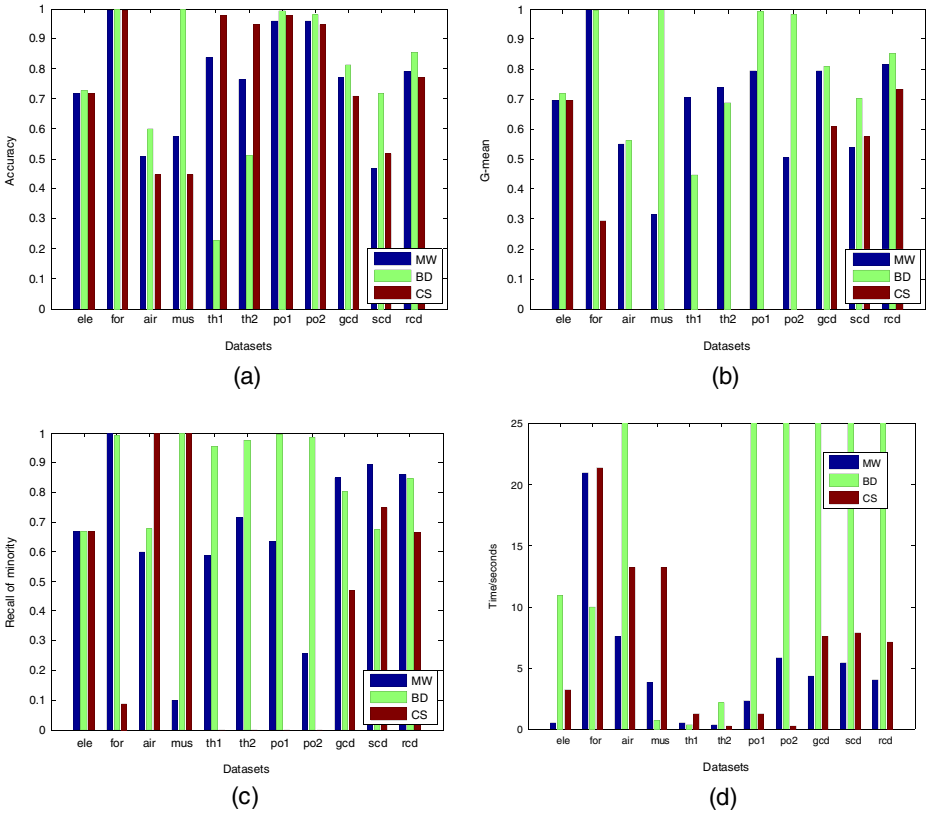

Figure 5 Performance comparison of the proposed MW with BD and CS approaches

methods have not been compared directly with each other under equivalent evaluation metrics. The results of the comparison are shown in Figure 5.

Figure 5(a) shows that the accuracy of MW is comparable those of BD and CS for most of the datasets considered. Because MW focuses greater attention on minority instances, the precision of the majority instances suffers, which affects the accuracy. Figure 5(b) shows that the G-mean values of MW are generally close to those of BD, and that MW outperforms CS for all datasets. When considering G-mean and $Recall_{min}$ in Figure 5(c) together, we note that, for *th1* and *th2*, BD exhibits obvious advantages in $Recall_{min}$ relative to MW, while the advantages are reversed with respect to G-mean. BD obtains better $Recall_{min}$ results because the approach accumulates all positive instances to build successor sub-classifiers. However, an excessive number of minority instances may

Table 4 Wilcoxon signed rank test statistics

	p-value			
	Accuracy	G-mean	Recall of minority	Time
MW vs. BD	0.27832031	0.14746094	0.14746094	0.03222656
MW vs. CS	0.46484375	0.00488281	0.76464844	0.00976563
BD vs. CS	0.14746094	0.00097656	0.36523437	0.32031250

overwhelm the majority, and, thus, the G-mean results suffer. In addition, a large number of minority instances also incur greater processing time, as shown in Figure 5(d). MW is much more efficient than BD, except for datasets *for* and *mus*, which are nearly balanced, and is consistently more efficient that CS except for datasets *po1* and *po2*. In actuality, the processing time for BD on the *air* dataset was greater than twenty hours, but we set it to 25 s for display purposes.

In addition to the visual comparison given in Figure 5, we also applied the Wilcoxon signed rank test to compare the statistical differences among MW, BD, and CS, and the corresponding *p*-values are listed in Table 4. Although BD achieves larger G-mean and $Recall_{min}$ values than MW for some datasets, no statistically significant difference is observed among the three methods in terms of accuracy with a significance level $\alpha = 0.05$. However, for G-mean and $Recall_{min}$, MW is better than CS. As for processing time, MW performs better than BD and CS at the given significance level.

5 Conclusions

For classification of imbalanced streaming data, we proposed a multi-window based ensemble learning (MWEL) framework to predict the class labels of newly arriving instances. We utilize multiple windows to preserve the current data batch, selected positive instances, and the set of latest sub-classifiers as well as the corresponding sets of instances used to train each sub-classifier. Moreover, before predicting the label of incoming instances, we update the weight of each sub-classifier by calculating the similarity between the current window and previous windows used to train each sub-classifier. A weighed majority voting strategy is then used to predict the class label. A new sub-classifier is trained only when the current ensemble classifier exhibits low precision for one or both minority and majority classes. Under conditions of substantial imbalance, we oversampled minority instances and under-sampled majority instances. Extensive experiments on both real-world datasets and synthetic datasets demonstrated that our method can process imbalanced stream data efficiently and effectively, and, in certain respects, outperforms existing methods, particularly with respect to processing time. Owing to problem complexity, we considered only two classes in the present study. However, many real-world applications involve multi-class or multi-label problems, so imbalanced multi-class and multi-label stream classification will be the focus of future research.

Acknowledgements This work was supported by ARC DP project (DP 130101327), 973 Program (Grant No. 2013CB329601, 2013CB329602, 2013CB329604), 863 Program (Grant No. 2012AA01A401, 2012AA01A402).

References

1. Alippi, C., Boracchi, G., Roveri, M.: Just in time classifiers: Managing the slow drift case. In: International Joint Conference on Neural Networks, 2009. IJCNN 2009. pp. 114–120 (2009).
2. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In, In SIAM International Conference on Data Mining (2007)
3. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J.-F. (eds.) Advances in Intelligent Data Analysis VIII, pp. 249–260. Springer, Berlin Heidelberg (2009)
4. Chawla, N.V.: Data mining for imbalanced datasets: an overview. In: Maimon, O. and Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook. pp. 853–867. Springer US (2005).

5. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
6. Chawla, N.V., Lazarevic, A., Hall, L.O., Bowyer, K.W.: SMOTEBoost: improving prediction of the minority class in boosting. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) Knowledge Discovery in Databases: PKDD 2003, pp. 107–119. Springer, Berlin Heidelberg (2003)
7. Chen, S., He, H.: Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach. Evol. Syst. **2**, 35–50 (2010)
8. Delany, S.J., Cunningham, P., Tsymbal, A., Coyle, L.: A case-based technique for tracking concept drift in spam filtering. In: CEng, P.A.M.Bs., MSc, R.E.Bs., and Allen, D.T. (eds.) Applications and Innovations in Intelligent Systems XII. pp. 3–16. Springer London (2005).
9. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 71–80. ACM, New York (2000).
10. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. IEEE Trans. Neural Netw. **22**, 1517–1531 (2011)
11. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**, 1263–1284 (2009)
12. Hoens, T.R., Polikar, R., Chawla, N.V.: Learning from streaming data with concept drift and imbalance: an overview. Prog. Artif. Intell. **1**, 89–101 (2012)
13. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 97–106. ACM, New York, (2001).
14. Kolter, J.Z., Maloof, M.: Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Third IEEE International Conference on Data Mining, 2003. ICDM 2003. pp. 123–130 (2003).
15. Lichtenwalter, R.N., Chawla, N.V.: Adaptive methods for classification in arbitrarily imbalanced and drifting data Streams. In: Theeramunkong, T., Nattee, C., Adeodato, P.J.L., Chawla, N., Christen, P., Lenca, P., Poon, J., Williams, G. (eds.) New Frontiers in Applied Data Mining, pp. 53–75. Springer, Berlin Heidelberg (2010)
16. Liu, W., Chawla, S., Cieslak, D.A., Chawla, N.V.: A robust decision tree algorithm for imbalanced data sets. In: in SIAM International Conference on Data Mining, 2010. pp. 766–777.
17. Liu, W., Wang, L., Yi, M.: Simple-random-sampling-based multiclass text classification algorithm. Sci. World J. **2014**, 1–7 (2014)
18. Parveen, P., Weger, Z.R., Thuraisingham, B., Hamlen, K., Khan, L.: Supervised learning for insider threat detection using stream mining. In: Proceedings of the 2011 I.E. 23rd International Conference on Tools with Artificial Intelligence. pp. 1032–1039. IEEE Computer Society, Washington, DC, (2011).
19. Rifkin, R., Klautau, A.: In defense of one-vs-all classification. J. Mach. Learn. Res. **5**, 101–141 (2004)
20. Shen, X., Boutell, M., Luo, J., Brown, C.: Multilabel machine learning and its application to semantic scene classification. Presented at the Storage and Retrieval Methods and Applications for Multimedia 2004 December 1 (2003).
21. Shi, J., Luo, Z.: Nonlinear dimensionality reduction of gene expression data for visualization and clustering analysis of cancer tissue samples. Comput. Biol. Med. **40**, 723–732 (2010)
22. Street, W.N., Kim, Y.: A Streaming ensemble algorithm (SEA) for large-scale classification. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 377–382. ACM, New York, (2001).
23. Sun, Y., Wong, A.K.C., Wang, Y.: Parameter inference of cost-sensitive boosting algorithms. In: Perner, P. and Imiya, A. (eds.) Machine Learning and Data Mining in Pattern Recognition. pp. 21–30. Springer Berlin Heidelberg (2005).
24. Tomek, I.: Two modifications of CNN. IEEE Trans. Syst. Man Cybern. **6**, 769–772 (1976)
25. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 226–235. ACM, New York (2003).
26. Wang, X., Jia, Y., Chen, R., Fan, H., Zhou, B.: Improving text categorization with semantic knowledge in wikipedia. IEICE Trans. Inf. Syst. **E96-D**, 2786–2794 (2013a)
27. Wang, S., Minku, L.L., Yao, X.: A learning framework for online class imbalance learning. In: 2013 I.E. Symposium on Computational Intelligence and Ensemble Learning (CIEL). pp. 36–45 (2013b).
28. Wang, S., Minku, L.L., Yao, X.: Online class imbalance learning and its applications in fault detection. Int. J. Comput. Intell. Appl. **12**, 1340001 (2013c)
29. Wang, Y., Li, H., Wang, H., Zhou, B., Zhang, Y.: Multi-window based ensemble learning for classification of imbalanced streaming data. In: 16th International Conference on Web Information Systems Engineering. pp. 78–92. Springer International Publishing, Miami, (2015).
30. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Mach. Learn. **23**, 69–101 (1996)

31. Xioufis, E.S., Spiliopoulou, M., Tsoumakas, G., Vlahavas, I.: Dealing with concept drift and class imbalance in multi-label stream classification. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Two. pp. 1583–1588. AAAI Press, Barcelona, Catalonia, Spain (2011).
32. Zhang, D., Shen, H., Hui, T., Li, Y., Wu, J., Sang, Y.: A selectively re-train approach based on clustering to classify concept-drifting data streams with skewed distribution. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., and Kao, H.-Y. (eds.) Advances in Knowledge Discovery and Data Mining. pp. 413–424. Springer International Publishing (2014).