Adaptive Chunk-Based Dynamic Weighted Majority for Imbalanced Data Streams With Concept Drift

Yang Lu, *Member, IEEE*, Yiu-Ming Cheung, *Fellow, IEEE*, and Yuan Yan Tang, *Life Fellow, IEEE*

***Abstract*— One of the most challenging problems in the field of online learning is concept drift, which deeply influences the classification stability of streaming data. If the data stream is imbalanced, it is even more difficult to detect concept drifts and make an online learner adapt to them. Ensemble algorithms have been found effective for the classification of streaming data with concept drift, whereby an individual classifier is built for each incoming data chunk and its associated weight is adjusted to manage the drift. However, it is difficult to adjust the weights to achieve a balance between the stability and adaptability of the ensemble classifiers. In addition, when the data stream is imbalanced, the use of a size-fixed chunk to build a single classifier can create further problems; the data chunk may contain too few or even no minority class samples (i.e., only majority class samples). A classifier built on such a chunk is unstable in the ensemble. In this article, we propose a chunk-based incremental learning method called adaptive chunk-based dynamic weighted majority (ACDWM) to deal with imbalanced streaming data containing concept drift. ACDWM utilizes an ensemble framework by dynamically weighting the individual classifiers according to their classification performance on the current data chunk. The chunk size is adaptively selected by statistical hypothesis tests to access whether the classifier built on the current data chunk is sufficiently stable. ACDWM has four advantages compared with the existing methods as follows: 1) it can maintain stability when processing nondrifted streams and rapidly adapt to the new concept; 2) it is entirely incremental, i.e., no previous data need to be stored; 3) it stores a limited number of classifiers to ensure high efficiency; and 4) it adaptively selects the chunk size in the concept drift environment. Experiments on both synthetic and real data sets containing concept drift show that ACDWM outperforms both state-of-the-art chunk-based and online methods.**

***Index Terms*— Concept drift, ensemble methods, imbalance learning, online learning.**

I. Introduction

In RECENT years, the challenge of learning from streaming data with concept drift has attracted more interest in the field of online learning because this phenomenon occurs in real machine learning applications, where the underlying data distribution changes over time [1], [2]. For example, spammers continuously improve the quality of the spam they post on Twitter to avoid being blocked by spam detection systems, and thus, the features and concepts of Twitter spam frequently change [3]. It follows that an online algorithm to process streaming data with concept drift should balance the tradeoff between learning from previous data and adapting to the new concept, which is also known as the stability-plasticity dilemma [4]. Concept drift can occur in the following four components, according to Bayes' theorem [5], [6].

1) *Data Distribution $p(\boldsymbol{x})$:* A type of virtual drift, where the distribution of $\mathbf{x}$ changes without affecting the decision hyperplane.
2) *Class-Conditional Probability (Likelihood) $p(\boldsymbol{x}|y)$:* A type of virtual drift, usually co-occurred with $p(\mathbf{x})$ and $p(y)$ drift.
3) *Posterior Probability $p(y|\boldsymbol{x})$:* A type of real drift, where the decision hyperplane is shifted because the conditional probability changes.
4) *Class Prior $p(y)$:* A type of virtual drift, where the imbalance ratio changes, such that the role of the minority class and the majority class may be switched.

As discussed in [6], only posterior probability drift affects the decision hyperplane. However, it is impossible for only one of the components of drift to change, while the others remain fixed. That is, these four types of drifts are correlated and usually occur simultaneously at any time in the real-world data stream. We therefore define the concept drift comprising a mixture of different components of drifts as joint concept drift. Take fault diagnosis for example, and consider a situation where the faulty samples are the minority class [7]. If there is a change in the percentage of different sample types, this corresponds to a drift of $p(\mathbf{x})$. If the percentage of a certain type of faulty samples is increasing, this corresponds to the drift of $p(\mathbf{x}|y)$. If the percentage of a certain type of faulty sample is increasing, perhaps due to the improvement of a fault-standard diagnostic, this corresponds to a drift of $p(y|\mathbf{x})$. Furthermore, if the cause of the fault is not fixed, the number of minority

Manuscript received April 30, 2018; revised January 29, 2019 and September 12, 2019; accepted November 2, 2019. Date of publication December 5, 2019; date of current version August 4, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61672444 and Grant 61272366, in part by the Hong Kong Baptist University (HKBU), Research Committee, Initiation Grant—Faculty Niche Research Areas (IG-FNRA) 2018/2019 under Grant RC-FNRA-IG/18-19/SCI/03, in part by the Innovation and Technology Fund of the Innovation and Technology Commission of the Government of the Hong Kong SAR under Project ITS/339/18, in part by the Faculty Research Grant of HKBU under Project FRG2/17-18/082, and in part by the Shenzhen Science, Technology and Innovation Committee (SZSTI) under Grant JCYJ20160531194006833. *(Corresponding author: Yiu-Ming Cheung.)*

Y. Lu is with the Fujian Key Laboratory of Sensing and Computing for Smart City, School of Informatics, Xiamen University, Xiamen 361005, China, and also with the Department of Computer Science, Hong Kong Baptist University, Hong Kong (e-mail: lylylytc@gmail.com).

Y.-M. Cheung is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong (e-mail: ymc@comp.hkbu.edu.hk).

Y. Y. Tang is with the Faculty of Science and Technology, UOW College Hong Kong/Community College of City University, Hong Kong, and also with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau, China (e-mail: yytang@cityu.edu.hk).

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

Digital Object Identifier 10.1109/TNNLS.2019.2951814

faulty samples will increase until they become the majority class, and this corresponds to a drift of $p(y)$. The challenge of investigating the concept drifts in nonstationary environments and developing adaptation algorithms has drawn increasing attention from researchers [8]–[11].

Another typical data mining problem is class imbalance, where one class has much more samples than another class [12], [13]. In this situation, directly applying standard learning algorithms tends to ignore minority class samples because of their subtle influence on the overall accuracy. Class imbalance and concept drift have mutually reinforcing effects, i.e., if these problems co-occur, they will tend to exacerbate each other. In other words, if the classes in the data stream are imbalanced, it will be even more difficult to detect the concept drifts of the minority class and adapt the online learner to them. Conversely, concept drift may alter the status of class imbalance because the class prior is a variable in the concept drift environment.

Thus far, only a few methods have been proposed for managing the problem of concept drift with class imbalance. A comprehensive literature review can be found in [6]. The existing methods can be divided into online and chunk-based approaches.[1] The online approaches [14]–[17] update the prediction model for each incoming sample and use a drift detector to monitor the data stream. Once any concept drift is detected, the existing prediction model is reset and a new model is built for the new concept. For example, oversampling-based online bagging (OOB) and undersampling-based online bagging (UOB) [15] use a time-decayed scheme to obtain the recent imbalance ratio of the data stream, which is used to integrate oversampling and undersampling with online bagging. It can be associated with a drift detector such as Drift Detection Method for Online Class Imbalance (DDM-OCI) [14], which gives a warning when necessary by monitoring the minority class recall and confirms detections using statistical information. However, a major problem of approaches based on the detectors is that they suffer from false alarms, delayed detections, or even missed detections. In contrast, the chunk-based approaches [4], [18]–[21] buffer the data stream until a certain number of samples are accumulated, and classifiers are then built on the data chunk. These approaches usually assume that the occurrence of concept drift is ubiquitous in the data stream and thus continuously update the current model. Usually, the ensemble framework is adopted to create a single classifier for each incoming data chunk and adjust the weights of classifiers to adapt to the new concept [22], [23]. In the meantime, the imbalance issue can be handled by accumulating the minority class samples in the past chunks. For example, dynamic feature group weighting (DFGW-IS) [20] uses importance sampling to draw minority class samples from the past chunks and bootstraps the majority class sample to create a bagging-like ensemble. This strategy works when the minority class is a fixed concept in the data stream. However, in the complex concept drift environment, the $p(y)$ drift leads to the change of the imbalance ratio. Therefore, when the minority class has expanded to become the majority class, the stored past minority class samples cannot be used to refine the current minority class. In addition, it is hard to properly weight the individual classifiers trained on different timestamps while maintaining a limited number of classifiers to prevent the number increasing infinitely.

Another aspect is the size of chunks in the chunk-based method. The chunk-based ensemble methods are usually based on the assumption that the imbalance ratio is fixed in each data chunk. However, the imbalance ratio may also be altered by concept drift, i.e., the prior drift [24], and when this prior drift occurs, it is likely that the incoming size-fixed chunk consists of only the majority class samples. In this case, most of the chunk-based methods that are designed to process imbalanced data streams fail because the classifier cannot be built upon a data set comprising only a single class. Moreover, even if there are two classes in the chunk, a low number of minority class samples may inadequately represent their class distribution; an extreme example of this would be the situation of there being only one minority class sample in the chunk. Thus, for the existing ensemble methods with a size-fixed chunk, using a large chunk size to avoid the imbalance problem will simply delay the reaction to the concept drift. Therefore, the selection of chunk size is crucial for handling imbalanced streaming data with prior drift. However, to the best of our knowledge, no chunk size selection algorithm exists in the literature for use in the chunk-based ensemble methods to process imbalanced data stream.

In this article, we propose an improved chunk-based incremental learning method called adaptive chunk-based dynamic weighted majority (ACDWM) to deal with binary class imbalanced data streams with concept drift. The method utilizes an ensemble framework with dynamic individual classifier weighting. The classifier weights depend on the imbalance-aware error of the classifier tested on the current chunk. A classifier with a smaller error will receive a higher weight in the ensemble because it may have been trained on the chunk with a similar concept as the current chunk. The classifier weights are naturally reduced over time based on the accumulated testing error. In addition, a proper chunk size is adaptively determined to train the individual classifiers, which enables ACDWM to handle the online learning scenarios that display joint concept drift. ACDWM is initialized with a small chunk and adaptively increases the chunk size until the stability of the classifier generated by the enlarged chunk is not significantly different from the one generated by the previous chunk. The stability is determined by the prediction variance of test samples. We propose a classifier called SubUnderSampling that has normally distributed predictions, meaning that the statistical hypothesis tests can be used to compare the prediction variance. There are four advantages of ACDWM as follows.

1) It can remain stable for stationary streams and also rapidly adapt to the new concept of a nonstationary environment.
2) It is wholly incremental, meaning that no previous data need to be stored to assist with prediction, and thus, the method is unaffected by memory problems.
3) It stores far fewer classifiers than the current timestamp, meaning that the outdated classifiers will be discarded so that the ensemble size does not expand infinitely.

[1]Another taxonomy classifies these as active and passive approaches [2]. These two taxonomies are similar because active approaches are almost all conducted online and passive approaches usually adopt a chunk-based learning scenario.

4) The classifier stability is ensured by training on the properly selected chunk size.

The main difference of this article compared with our preliminary work in [21] is that ACDWM can adaptively select the chunk size to ensure the stability of the individual classifier in the ensemble, thus enabling the data stream with a prior class drift to be fitted. In addition, we systematically compare the chunk-based and online methods in the joint concept drift environment. Thus, the main contributions of this article are as follows.

1) It is the first to examine the chunk size problem of ensemble methods processing imbalanced streaming data with joint concept drift.
2) We propose an online learning method ACDWM for processing imbalanced data stream classification with joint concept drift.
3) We propose a chunk size selection algorithm to obtain an appropriate chunk size for online ensemble methods.

The rest of this article is organized as follows. Section II is an overview of related work on processing imbalanced data streams with concept drift. Section III describes the proposed ACDWM in detail. Section IV presents the experiments and discussions. Conclusions are presented in Section V.

II. Related Work

In this section, we briefly review the known online and chunk-based methods for processing imbalanced data streams with concept drift. Online approaches update the model for each incoming sample. recursive least square adaptive cost perceptron (RLSACP) [25] uses perceptron-based classifiers and handles the concept drift using a forgetting factor of the error model, with the imbalance ratio proposed to adapt the error weight. OOB and UOB [14], [15], [26] combine traditional random oversampling and undersampling with online bagging [27], where each individual classifier in bagging is updated K times on the incoming new sample. In OOB and UOB, K is a Poisson random number, which is updated by a decayed factor according to the recent imbalance ratio. There are also some approaches based on change detection [1]. DDM-OCI [14] improves DDM [28] by monitoring the minority class recall instead of the accuracy used in DDM for the class imbalance issue. Linear Four Rates (FLR) [29] traces four rates, i.e., TPR, TNR, positive predicted value (PPV), and negative predicted value (NPV), with statistical tests to detect the drift on both positive and negative classes. Hierarchical FLR (HFLR) [16] further improves FLR by adopting permutation based concept drift detection scheme (PERM) [30] as the upper layer to verify the detection result from FLR. Prequential area under the ROC curve Page–Hinkley (PAUC-PH) [17] extends the traditional metric area under the ROC curve (AUC) to the online calculable version and uses the PH-test [31] to detect drift. Class-Based ensemble for Class Evolution (CBCE) [32] focuses the class evolution problem for multi-class classification, using a one-versus-rest strategy to create one classifier for each class and conduct undersampling for the majority class.

The chunk-based method for an imbalanced data stream was first proposed in [18]. The proposed method accumulates the minority class samples in the current chunk with those in all past chunks while conducting undersampling on the majority class samples to balance the classes. A bagging-like ensemble framework is then used for classification. However, it suffers from limitations in memory for storing the past data and lacks the ability to rapidly adapt to the new concept. selectively recursive approach (SERA) [33] and recursive ensemble approach (REA) [19] are improvements, as they select only parts of the minority class samples from the past chunks based on the similarity to the minority class samples in the current chunk. heuristic updatable weighted random subspaces with instance propagation (HUWRS.IP) [5] calculates the Hellinger distance between the samples in the current and past chunks to detect the concept drift. The distance is used as the weight of the individual classifier built on different feature subspaces in the ensemble. To handle the imbalance problem, HUWRS.IP creates a Naïve Bayes classifier on the current chunk to select the similar minority class samples from the past chunks. DFGW-IS [20] further improves HUWRS.IP by incorporating the importance sampling technique to collect similar positive class samples from the past chunks. Gradual Resampling Ensemble (GRE) [34] selects the minority class samples from the past chunks by utilizing a clustering technique. Thus, only the minority samples that have minimal overlap with the majority samples in the current chunk are selected to construct the current minority training set. In summary, these methods are based on the assumption that the minority class does not change and the past information can therefore be utilized. However, from a practical viewpoint, if $p(y)$ changes over time, the past minority class may not be the same class as the current minority class. In light of this, some methods do not use historical data. Learn++.CDS and Learn++.NIE [4] create one individual classifier for each chunk and use the ensemble to predict the incoming data. The individual classifiers are weighted according to a time-decay function and their performance on the current chunk. ensemble of subset online sequential extreme learning machine (ESOS-ELM) [35] uses an ensemble of extreme learning machine, where the weight matrices trained on each chunk are stored for the ensemble. When the concept drift is detected by a change detector, the ensemble model will be reinitialized. Our previously developed method Dynamic Weighted Majority for Imbalance Learning (DWMIL) [21] keeps a limited number of classifiers in the ensemble, and the classifier weights are then decayed based on their discriminative ability on the current chunk and the timestamp of their creation.

All the above-listed chunk-based methods use size-fixed chunks in the data stream and are based on the assumption that the samples of both classes exist in each chunk. However, if prior drift occurs, a very high imbalance ratio may result, which can develop into a situation where the chunk consists of only one class. As far as we know, a chunk size selection algorithm has yet to be developed for online ensemble methods with such an imbalanced data stream. In the literature, a related topic is adaptive window for drift detection. DDM [28] monitors the error rate and sets two levels for drift detection; when the warning level is triggered, all subsequent incoming data are stored. When the drift level is triggered, the stored samples are used to train a new model, where the window size depends

on the steps between two levels. ADWIN [36] is based on the theoretical analysis of Hoeffding bound to determine if two parts of a data sequence are from the same distribution. PERM [30] compares the testing error of the data in the latest window with the validation error by random permutation, and the window size is increased if there is no significant difference between the two errors. These methods are designed for drift detection, and the criterion that determines the window size depends on the point when the concept drift happens. However, when the data stream is imbalanced, these methods cannot be used to determine whether there is sufficient data (especially the minority class data) in the current chunk to build a stable classifier.

III. Proposed Method

A. General Framework

An online algorithm that can address concept drift should rapidly adapt to the new concept if the changes in the data stream occur, and yet remain stable in a stationary environment. DWM [37] uses a weighted ensemble to predict streaming data with concept drift. Each classifier is associated with a weight when it is created. This weight is periodically updated based on the performance of the classifier on the data stream. For every p timestamps, the weight is decreased by a factor β if the individual classifier in the ensemble misclassifies the incoming sample in the data stream. DWM also periodically removes and creates classifiers to deal with the concept drift. If the weight is lower than a threshold, the corresponding classifier will be removed, and if the ensemble classifier misclassifies an incoming sample, a new classifier will be created. Therefore, as an ensemble method, the effectiveness of DWM is due to the dynamic control of the classifier weights. Specifically, the weight adjustment scheme considers two aspects. One is the time factor: The outdated classifiers have lower weights when they are farther from the current timestamp. The other one is the concept drift factor: if the concept drift occurs, the speed of weight reduction will increase, such that the classifier built on the old concept will receive a lower weighting. The Learn++ framework [38] incorporates a similar consideration, in which the time factor is integrated with the error to calculate the weight of each individual classifier. The pseudocode of DWM is shown in Algorithm 1.[2]

Algorithm 1 DWM

Input: Data stream t: $\mathcal{D} = \{\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$, $i = 1, \ldots, N$, number of classes c, threshold for deleting individual classifiers θ, factor for decreasing weights β, period between classifier removal, creation, and weight update p.

$m \leftarrow 1$;
$w_m \leftarrow 1$;
$H_m \leftarrow CreateClassifier$;
$\mathcal{H} \leftarrow \{H_m\}$;
for $i \leftarrow 1$ to N **do**
  for $j \leftarrow 1$ to m **do**
    if $H_j(\mathbf{x}_i) \neq y_i$ and i mod $p = 0$ **then**
      $w_j \leftarrow \beta w_j$;
    end if
  end for
  Predict $\mathbf{x}_i$ by the ensemble classifier: $\bar{y}_i \leftarrow sign(\sum_{j=1}^{m} w_j H_j(\mathbf{x}_i))$;
  if i mod $p = 0$ **then**
    Normalize classifier weights: $\mathbf{w} \leftarrow \mathbf{w}/\sum_j w_j$;
    Remove classifiers with weights less than θ: $\mathcal{H} \leftarrow \mathcal{H} \backslash \{H_j | w_j < \theta\}$;
    if $\bar{y}_i \neq y_i$ **then**
      $m \leftarrow m + 1$;
      $H_m \leftarrow CreateClassifier$;
      $\mathcal{H} \leftarrow \mathcal{H} \cup H_m$;
      $w_m \leftarrow 1$;
    end if
  end if
  for $j \leftarrow 1$ to m **do**
    $H_j \leftarrow UpdateClassifier(H_j, \mathbf{x}_i, y_i)$;
  end for
end for

Output: Ensemble classifier set $\mathcal{H}$, weight of individual classifiers $\mathbf{w}$.

Although DWM is effective on data streams with concept drift, it tends to adjust the classifier weights improperly if the data stream is imbalanced. The reasons for this are as follows. First, DWM processes one data at a time and updates the classifiers on every p timestamps. If the data stream is imbalanced, the occurrence frequency of the minority class samples will be very low. Thus, updating a long sequence of majority class samples will likely bias the model toward the majority class. Second, the weight update depends on the classification results of each individual classifier in the ensemble for every p timestamps. A classifier built on an imbalanced data stream usually has high accuracy on the majority class and low accuracy on the minority class. Thus, the weight update frequency will be low because the individual classifier makes correct predictions on the majority class samples in most of the cases, as these samples appear more frequently. Consequently, the weight update is barely influenced by the classification result for the minority class. Finally, a new classifier is created by the DWM according to the prediction performance of a single sample. If the data stream is imbalanced, it is highly probable that the sample belongs to the majority class rather than the minority class, and there is a low probability that a majority class sample will be misclassified. Thus, there is a low probability that new classifiers will be created on an imbalanced data stream, meaning that it cannot efficiently adapt to a concept drift, especially when the drift occurs in the minority class data.

The proposed ACDWM incorporates the advantages of DWM in that the individual classifiers are dynamically adjusted to address concept drift with class imbalance. The outdated classifiers are gradually allocated lower weights and are removed when these weights are beneath a certain threshold. The weight is reduced according to the performance of the classifiers on the current concept, with the difference being that ACDWM adopts a chunk-by-chunk online learning fashion such that a classifier is created for each data chunk.

[2]The DWM in this article is for binary classification to match the task in this article. The DWM in the original article is for multiclass classification [37].

The class imbalance problem is then solved within the chunk by sampling techniques, and imbalance-aware metrics are used to adjust the classifier weights. In addition, ACDWM can adaptively select a proper chunk size to create a new classifier, thereby preventing the data in the size-fixed chunk comprising only one class. This adaptive chunk size selection can also ensure that the classifier built on the chunk is stable enough to assist with ensemble prediction. The details of ACDWM are described in Sections III-B and III-C.

B. Chunk Training

At timestamp $t-1$, ACDWM maintains m classifiers in the set $\mathcal{H}^{(t-1)} = \{H_1^{(t-1)}, \ldots, H_m^{(t-1)}\}$ trained on the data chunks from timestamp 1 to $t-1$. When ACDWM receives a new data chunk $\mathcal{D}^{(t)}$ at timestamp t, it learns a new classifier H on the current data chunk and merges H with $\mathcal{H}^{(t-1)}$ to form $\mathcal{H}^{(t)}$. The classifiers trained at each chunk are associated with weights $\mathbf{w}^{(t)} = [w_1^{(t)}, \ldots, w_m^{(t)}]^T$, which represent the importance of the classifier in the ensemble. When the new classifier H is created, its initial weight is set at 1 and m is increased by 1. In order to adapt to a new concept and make the previously learned classifiers less influential in the ensemble, the weight $w_j^{(t)}$ for classifier $H_j^{(t)}$ is reduced on each timestamp after it is created

$$w_j^{(t)} = (1 - \epsilon_j^{(t)}) w_j^{(t-1)} \tag{1}$$

where $j = 1, \ldots, m-1$ and $\epsilon_j^{(t)}$ is the testing error of $H_j^{(t)}$ on the current data chunk $\mathcal{D}^{(t)}$. The error $\epsilon_j^{(t)}$ can be calculated by any error function, such as the F1 score or geometric mean (G-mean). Thus, the weights of the classifiers trained on the past chunks are reduced based on their performance on the current data chunk. As this weight reduction is accumulated over time, the weight $w_j^{(t)}$ is actually equal to

$$w_j^{(t)} = \prod_{\tau=l+1}^{t} \left(1 - \epsilon_j^{(\tau)}\right) \tag{2}$$

where l is the timestamp when $H_j^{(t)}$ is created. As $1-\epsilon_j^{(\tau)} \leq 1$, the classifier weight decreases over time according to the error on each chunk after it is created. Then, the classifiers with a weight less than the threshold θ are removed and the counter m is also reduced according to the number of classifiers left. Thus, ACDWM only retains a limited number of classifiers in the ensemble and does not suffer from memory problems if the data stream is extremely long.

There are two factors that make the weight of a classifier lower than θ, thus flagging it for removal. One is that the classifier is trained on a very early timestamp that makes the production in (2) small. The other is that the concept has changed in recent chunks and the testing error of the classifier on those chunks is large. Thus, this kind of classifier is less likely to provide positive effects for the prediction on the current and following chunks. Finally, the model predicts the incoming data $\mathbf{x}$ in $\mathcal{D}^{(t+1)}$ by the ensemble of $\mathcal{H}^{(t)}$ associated with the classifier weights $\mathbf{w}^{(t)}$

$$\text{sign}\left(\sum_{j=1}^{m} w_j^{(t)} H_j^{(t)}(\mathbf{x})\right). \tag{3}$$

Algorithm 2 ACDWM_Train

Input: Data chunk at timestamp t: $\mathcal{D}^{(t)} = \{\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$, $i = 1, \ldots, N$, threshold for deleting individual classifiers θ, individual classifier set $\mathcal{H}^{(t-1)} = \{H_1^{(t-1)}, \ldots, H_m^{(t-1)}\}$, weight of individual classifiers $\mathbf{w}^{(t-1)}$, minimal ensemble size T_0.

1: $m \leftarrow |\mathcal{H}^{(t-1)}|$;
2: **for** $i \leftarrow 1$ to N **do**
3: Predict $\mathbf{x}_i$ by the ensemble classifier: $\bar{y}_i \leftarrow sign(\sum_{j=1}^{m} w_j^{(t-1)} H_j^{(t-1)}(\mathbf{x}_i))$;
4: **end for**
5: **for** $j \leftarrow 1$ to m **do**
6: Calculate the error $\epsilon_j^{(t)}$ for classifier $H_j^{(t-1)}$ on $\mathcal{D}^{(t)}$;
7: Update weight of individual classifiers: $w_j^{(t)} \leftarrow (1 - \epsilon_j^{(t)}) w_j^{(t-1)}$;
8: **end for**
9: Remove classifiers with weights less than θ: $\mathcal{H}^{(t)} \leftarrow \mathcal{H}^{(t-1)} \backslash \{H_j^{(t-1)} | w_j^{(t)} < \theta\}$;
10: $m \leftarrow |\mathcal{H}^{(t)}|$;
11: Create new individual classifier: $H \leftarrow UnderBagging(\mathcal{D}^{(t)}, T_0)$;
12: $m \leftarrow m + 1$;
13: $\mathcal{H}^{(t)} \leftarrow \mathcal{H}^{(t)} \cup H$;
14: $w_m^{(t)} \leftarrow 1$;

Output: Ensemble classifier set $\mathcal{H}^{(t)}$, weight of individual classifiers $\mathbf{w}^{(t)}$, number of individual classifiers m, prediction $\bar{\mathbf{y}}$.

Algorithm 3 UnderBagging

Input: Data $\mathcal{D} = \{\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$, $i = 1, \ldots, N$, the number of positive samples N_p, the number of negative samples N_n, minimal ensemble size T_0.

1: $N_s \leftarrow \min(N_n, N_p)$;
2: $T = \max(T_0, \lceil \max(N_n, N_p)/N_s \rceil)$;
3: **for** $t \leftarrow 1$ to T **do**
4: $\mathcal{D}_p \leftarrow$ Bootstrap N_s positive samples;
5: $\mathcal{D}_n \leftarrow$ Bootstrap N_s negative samples;
6: $h_t \leftarrow BaseLearner(\{\mathcal{D}_p, \mathcal{D}_n\})$;
7: **end for**

Output: Classifier $H(\mathbf{x}) = sign(\sum_{t=1}^{T} h_t(\mathbf{x}))$;

The chunk training process of ACDWM is shown in Algorithm 2. The prediction of the data in the current chunk by the ensemble classifier is shown in lines 2–4. Then, the error of each individual classifier in the ensemble is calculated and used to update its weight in lines 5–8. After the weight update, the classifiers with a weight lower than θ are removed in lines 9–10. Finally, a new classifier is created in the ensemble and its weight is initialized in lines 10–14. In ACDWM, we use UnderBagging as the learner to train imbalanced data, as shown in Algorithm 3. In each bagging iteration,

Algorithm 4 SUB_Train

Input: Training set $\mathcal{D}$, pool size Q, subsampling size k.

$\mathcal{D}_p \leftarrow$ Positive samples in $\mathcal{D}$;
$\mathcal{D}_n \leftarrow$ Negative samples in $\mathcal{D}$;
for $q \leftarrow 1$ to Q **do**
  if $|\mathcal{D}_p| < |\mathcal{D}_n|$ **then**
    $\mathcal{D}'_p \leftarrow$ Take all samples from $\mathcal{D}_p$;
    $\mathcal{D}'_n \leftarrow$ Take k samples from $\mathcal{D}_n$;
  else
    $\mathcal{D}'_p \leftarrow$ Take k samples from $\mathcal{D}_p$;
    $\mathcal{D}'_n \leftarrow$ Take all samples from $\mathcal{D}_n$;
  end if
  $h_q \leftarrow BaseLearner(\{\mathcal{D}'_p, \mathcal{D}'_n\})$;
end for

Output: Classifier pool $\mathcal{H} = \{h_1, \ldots, h_Q\}$.

Algorithm 5 SUB_Variance

Input: Classifier pool $\mathcal{H} = \{h_1, \ldots, h_Q\}$, testing set $\mathcal{S}_t = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_t}\}$, ensemble size T, pool size Q, number of simulations P.

for $q \leftarrow 1$ to Q **do**
  for $i \leftarrow 1$ to $|\mathcal{S}_t|$ **do**
    $\mathbf{o}_{qi} \leftarrow sign(h_q(\mathbf{x}_i))$;
  end for
end for
for $i \leftarrow 1$ to $|\mathcal{S}_t|$ **do**
  for $p \leftarrow 1$ to P **do**
    $\mathcal{I} \leftarrow$ Randomly select T classifiers;
    $r_p \leftarrow sign(\sum_{t \in \mathcal{I}} \mathbf{o}_{ti})$;
  end for
  $v_i = Var(\mathbf{r})$;
end for

Output: The prediction variance vector $\mathbf{v}$.

we conduct undersampling on the majority class to make the training data balanced. The ensemble size T of UnderBagging is set by $\lceil \max(N_n, N_p)/N_s \rceil$, which means that T increases as does the extent of imbalance. If the imbalance ratio is higher, the ensemble size of UnderBagging is larger, to ensure that each majority class sample has a certain probability of being involved in the training process. We also set a minimal ensemble size T_0 to ensure that the ensemble size has a minimal value if the training data are relatively balanced.

Compared with DWM, the ACDWM technique of processing the imbalanced data stream chunk-by-chunk is more stable, as the problem of class imbalance can be solved within the chunk. That is, instead of creating a new classifier based on the classification result on a single classifier, ACDWM creates a new classifier for each chunk to learn the new concept in time. Besides, in DWM, the weights are reduced by a fixed parameter β and reduced again after normalization. Instead, ACDWM reduces the weight based on the performance without any normalization. Thus, a classifier in ACDWM can last longer, and thus contribute to the prediction for longer, if the current concept is like that when the classifier is created. Both the Learn++ framework [4], [38] and ACDWM create classifiers for each chunk and use the testing error on the current chunk to adjust the weights. However, the Learn++ framework uses a time-decay function σ to reduce the weights of the classifiers trained on the past chunks. This σ depends on two free hyperparameters a and b, where different values of these will produce diverse results. In ACDWM, the weight reduction depends only on the performance of the classifiers without free parameters. Furthermore, in the Learn++ framework, the weights depend not only on the current chunk but also depend on all the chunks from when the classifier was created up until the current chunk. Under these circumstances, bias may be produced. Specifically, if one classifier shows good prediction ability on its created chunk, it will continuously receive higher weights in the following several chunks. If the concept changes, the classifier with high weight but trained on the old concept will hinder the prediction on the new concept. In addition, the Learn++ framework keeps all the classifiers in the ensemble over time. This increasing number of classifiers will aggravate the computational burden if the algorithm runs on an extremely long or lifelong data stream because it needs to evaluate all stored classifiers on the current chunk to reassign their weights. In contrast, ACDWM removes the outdated and ineffective classifiers, thus greatly increasing the computational efficiency.

C. Adaptive Chunk Size Selection

When the data stream is imbalanced, the classifier trained from the imbalanced data chunk may be unstable because the limited minority class data do not represent the true distribution. If an unstable classifier is adopted in the ensemble, the overall performance of the ensemble classifier cannot be guaranteed even if weight adjustment techniques are utilized. Under these circumstances, a straightforward solution is to increase the chunk size until the number of samples in the chunk is enough to produce a classifier that affords stable predictions. Thus, one can incrementally create new classifiers and compare their validation error to determine a proper chunk size. However, as the chunk size increases, the validation error may continuously decrease simply because the training set becomes larger. Therefore, instead of measuring a validation error, we measure the stability by the variance of predictions of the chunks of different sizes and then select the appropriate chunk size for training. To this end, the proposed adaptive chunk size selection module in ACDWM adaptively compares the stability of the classifier trained on the current chunk and that trained on the enlarged chunk that includes a few more samples from the data stream. If the classifier obtained from the latter enlarged chunk is much more stable, ACDWM replaces the current chunk with the enlarged chunk and continues the comparison. When there is no significant stability increase, ACDWM terminates and the current chunk is used for training individual classifiers in the ensemble.

The stability of a classifier is measured by prediction variance. We propose a special algorithm call subunderbagging (SUB) to calculate the prediction variance on some testing data, as shown in Algorithms 4 and 5. This algorithm conducts undersampling on the majority class and obtains a relatively balanced training set to build classifiers. This process repeats Q times to create a classifier pool. For each testing sample $\mathbf{x}$, T classifiers randomly selected from the pool are

combined to obtain the ensemble prediction. We get P predictions for each test sample $\mathbf{x}$, and the different combination of the T classifiers and the variance of P predictions can be calculated. It can be shown by Theorem 1 that the predictions generated by SUB are normally distributed [39].

Theorem 1 [39]: Let Z_i be the iid sample drawn from distribution F_Z and let $U_{N,k_N,T}$ be an incomplete, infinite-order U-statistic with kernel h_{k_N}. Let $\theta_{k_N} = \mathbb{E}h_{k_N}(Z_1, \ldots, Z_{k_N})$, such that $\mathbb{E}h^2_{k_N}(Z_1, \ldots, Z_{k_N}) \leq C < \infty$ for all n and some constant C, and let $\lim \frac{N}{T} = \alpha$. Then, as long as $\lim \frac{k_N}{N} = 0$ and $\lim \operatorname{cov}(h_{k_N}(Z_1, \ldots, Z_{k_N}), h_{k_N}(Z_1, Z'_2, \ldots, Z'_{k_N})) \neq 0$, $U_{N,k_N,T}$ is asymptotic normal.

As in [39], when a modified bagging algorithm takes k_N samples in each round with the ensemble size T instead of bootstrapping all samples, the ensemble prediction

$$p(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} L_{\mathbf{x},k_N}\left((X_{i_1}, y_{i_1}), \ldots, (X_{i_{k_N}}, y_{i_{k_N}})\right) \quad (4)$$

is incomplete and infinite-order U-statistic [40]. $L_{\mathbf{x},k_N}((X_{i_1}, y_{i_1}), \ldots, (X_{i_{k_N}}, y_{i_{k_N}}))$ be the prediction of $\mathbf{x}$ by the tree-based classifier trained on k_N randomly selected samples. According to Theorem 1, the prediction $p(\mathbf{x})$ is asymptotic normal if the conditions are satisfied. For SUB, the classifiers are generated by taking k_N samples from the majority class with the size N and all minority class samples are engaged in training. Thus, the prediction of SUB can be written as

$$p'(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} L_{\mathbf{x},k_N}\left(\left(X^-_{i_1}, y^-_{i_1}\right), \ldots, \left(X^-_{i_{k_N}}, y^-_{i_{k_N}}\right), \left(X^+_1, y^+_1\right), \ldots, \left(X^+_M, y^+_M\right)\right) \quad (5)$$

where (X^-, y^-) is the majority class sample and (X^+, y^+) is the minority class sample, and M is the size of the minority class. It is also an incomplete and infinite-order U-statistic because the minority class samples are fixed and the average of the function $L_{\mathbf{x},k_N}$ by selecting k_N from N in the majority class satisfies the definition of incomplete and infinite-order U-statistic [39]. Therefore, $p'(\mathbf{x})$ is also a normal distribution random variable and its variance of P i.i.d. samples can be investigated with statistical tools.

ACDWM calculates the prediction variances of $\mathbf{x}$ on both the current chunk and the enlarged chunk, denoted as v and v', respectively. As the prediction of SUB is normally distributed, we can adopt a one-tailed F-test of equality of variances to determine if v is significantly larger than v' [41]. The test statistic of the F-test is calculated by $F = \sigma^2_A/\sigma^2_B$, where $A = (A_1, \ldots, A_a)$ and $B = (B_1, \ldots, B_b)$ are drawn i.i.d. from two normal distributions, and σ^2_A and σ^2_B are their variances, respectively. The test statistic F has an F-distribution with $a-1$ and $b-1$ degrees of freedom if the null hypothesis of equality of variances is accepted. Then, the p-value of F can be calculated and used to judge whether σ^2_A is obviously larger than σ^2_B. In our case, $\sigma^2_A = v$, $\sigma^2_B = v'$, and $a = b = P$. To avoid the prediction variance of a single test sample $\mathbf{x}$ leading to biased results, a set of n_t test samples is used to verify the classifier stability. As the stability is used to confirm whether there are enough minority class samples, all test samples are taken from the minority class. Thus, a set of p-values can be obtained by conducting an F-test. As each p-value is independent of another, we use Fisher's method [42] to combine multiple p-values, and the final test statistic $\chi^2_{2n_t} = -2\sum_{i=1}^{n_t}\ln(p_i)$ can be obtained, where p_i is the p-value of the F-test on the test sample $\mathbf{x}_i$ and $\chi^2_{2n_t}$ is a chi-squared distribution with n_t degrees of freedom. Therefore, we can obtain the final p-value from $\chi^2_{2n_t}$. If the p-value is smaller than a given confidence level Δ, this means that the enlarged chunk can produce a more stable classifier.

Algorithm 6 ACDWM

Input: Number of simulations P, significance level Δ, window size d, ensemble size T, forest pool size Q, subsampling size k, testing size n_t, minimal ensemble size T_0.

1: $t \leftarrow 1$, $l \leftarrow t$;
2: $\mathcal{S}_t \leftarrow$ Randomly select n_t minority class samples from previous stream;
3: $\mathcal{H} \leftarrow \emptyset$, $\mathbf{w} = \emptyset$;
4: **while** not end of stream **do**
5: $enough \leftarrow 0$;
6: $\mathcal{S} \leftarrow \{x_t, \ldots, x_{l+d}\}$;
7: $\mathcal{H}_p \leftarrow$ SUB_train($\mathcal{S}$, Q, k);
8: $\mathbf{v} \leftarrow$ SUB_variance($\mathcal{H}$, $\mathcal{S}_t$, T, Q, P);
9: **while** $enough = 0$ **do**
10: $l \leftarrow l + d$;
11: $\mathcal{S}' \leftarrow \{x_t, \ldots, x_{l+d}\}$;
12: $\mathcal{H}_p \leftarrow$ SUB_train($\mathcal{S}'$, Q, k);
13: $\mathbf{v}' \leftarrow$ SUB_variance($\mathcal{H}_p$, $\mathcal{S}_t$, T, Q, P);
14: **for** $i \leftarrow 1$ to n_t **do**
15: $F \leftarrow v_i/v'_i$;
16: $p_i \leftarrow$ The p-value of F by F-test;
17: **end for**
18: $K \leftarrow -2\sum_i \log(p_i)$;
19: $p_K \leftarrow$ The p-value of K by Fisher test;
20: **if** $p_K < \Delta$ **then**
21: $\mathbf{v} \leftarrow \mathbf{v}'$;
22: $\mathcal{S} = \mathcal{S}'$;
23: **else**
24: $\mathcal{H}$, $\mathbf{w} \leftarrow$ACDWM_train($\mathcal{S}$, $\mathcal{H}$, $\mathbf{w}$);
25: $\mathcal{S}_t \leftarrow$ Randomly select n_t minority class samples from $\mathcal{S}'$;
26: $enough \leftarrow 1$;
27: $t \leftarrow l$;
28: **end if**
29: **end while**
30: **end while**

Output: Ensemble classifier $H(\mathbf{x}) = sign(\sum_{t=1}^{|\mathcal{H}|} w_t\mathcal{H}_t(\mathbf{x}))$;

ACDWM is summarized in Algorithm 6. When d samples arrive from the data stream to form a data chunk, they are collected in $\mathcal{S}$ as shown in line 6. $\mathcal{S}$ is used to obtain the trained classifier pool $\mathcal{H}_p$ by SUB_train, and then, the prediction variance $\mathbf{v}$ can be calculated to represent the stability of classifier built on $\mathcal{S}$. After that, we continuously increase the chunk size in the loop in lines 9–29. $\mathcal{S}'$ is the chunk enlarged by d samples, and $\mathbf{v}'$ is the corresponding prediction variance. The statistical test is conducted in lines 14–19. If the statistic p_K is smaller than the significance level Δ in line 20, it means

that the prediction variance of the classifiers trained on the enlarged chunk is much smaller, and thus, the loop continues. Otherwise, we use the chunk $\mathcal{S}$ to train a new classifier by ACDWM_train in line 24. Finally, the output of Algorithm 6 is the weighted ensemble classifier constructed by $\mathcal{H}$ and $\mathbf{w}$.

The computational cost of the chunk size selection part in ACDWM has two main parts: SUB_train and SUB_variance. SUB_train trains Q classifiers by $BaseLearner$ on training set $\mathcal{D}$. Its computational cost is $O(QL_{tr}(|\mathcal{D}|))$, where $L_{tr}(|\mathcal{D}|)$ is the cost of training $|\mathcal{D}|$ samples by $BaseLearner$. SUB_variance randomly selects T classifiers from Q repeated by P times with different permutations for each testing sample. It can be performed by first predicting each testing sample with all Q classifiers in lines 1–5 of Algorithm 5 and then randomly selecting T from Q classifiers by P times in lines 6–10 of Algorithm 5. The computational cost is $O(QL_{te}(n_t)+PQTn_t)$, where $L_{te}(n_t)$ refers to the testing cost for n_t testing samples by a trained classifier. The second term is the addition operation performed $PQTn_t$ times. Therefore, the total computational complexity for one iteration of checking is $O(QL_{tr}(|\mathcal{D}|)+QL_{te}(n_t)+PQTn_t)$. Most of computational cost is therefore associated with the first two terms, i.e., the training and testing of $BaseLearner$.

Remarks: The remarks are stated as follows.

1) The subsampling size k_N is set slightly larger than N as suggested in [39]. Therefore, we set the subsampling size $k = \sqrt{|\mathcal{D}|}$ in SUB. As the training data $\mathcal{D}$ are supposed to be imbalanced, k will be slightly larger than the square root of the majority class size. As k is small compared with $|\mathcal{D}|$, the computational burden for SUB to generate Q classifiers is not great. In addition, the training process of SUB can be easily parallelized.
2) ACDWM is not specifically designed for drift detection because the chunk-based ensemble methods *per se* are able to manage concept drift by adjusting the classifier weights. However, when the concept drift occurs, ACDWM can also stop increasing the size of the current chunk because the enlarged chunk contains the samples from different concepts that will probably increase the prediction variance of the generated classifier. To explain further, as the one-tailed F-test of equality between υ and υ' is only rejected when $\upsilon > \upsilon'$ with confidence level Δ, it is not rejected if υ' is larger. Therefore, when υ' is larger, the classifier created by the current chunk will be regarded as stable, and thus, ACDWM stops increasing the size of the current chunk.
3) ACDWM can produce a high-quality data chunk for training stable classifiers on imbalanced data streams with concept drifts, but this comes at the cost of an increased computational burden. Therefore, a tradeoff should be considered when handling high-speed data stream processing.

IV. Experiments

In this section, we empirically compare the proposed method ACDWM with other state-of-the-art methods for imbalanced streaming data with concept drift. We also show the effectiveness of the adaptive chunk selection module in ACDWM and how it manages concept drifts.

A. Experiment Settings

We select five chunk-based and four online methods to compare with ACDWM. The compared chunk-based methods are as follows.

1) *Uncorrelated Bagging (UB) [18]:* All the positive samples in previous chunks are stored and combined with the positive samples in the current chunk. The negative samples are drawn based on the sampling ratio r, which is set at 0.5.
2) *REA [19]:* The minority class samples are stored but only those that are in the k-nearest neighbors of the minority class samples in the current chunk that are used.
3) *Learn++.NIE [4]:* Each chunk is built with a classifier and these are weighted according to their performance and a time-decay function.
4) *DFGW-IS [20]:* The ensemble is composed of classifiers built on data with feature subspace. The classifiers are created by importance sampling and weighted according to the validation error and the distribution distance.
5) *DWMIL [21]:* The fixed-size-chunk version of ACDWM.

The chunk size of all chunk-based methods is fixed at 1000. The compared online methods are as follows.

1) *OOB [15]:* Oversampling ratio is used as λ in the Poisson distribution of online bagging.
2) *DDM-OCI [14]:* The minority class recall is monitored to detect concept drift.
3) *HLFR [16]:* Four rates are monitored for drift detection, and the permutation test is used to confirm the detection.
4) *PAUC-PH [17]:* Prequential AUC is calculated with the data stream, and a PH test on PAUC is used to detect concept drift.

Aside from OOB, these online methods monitor the data stream with a drift detector, for which OOB is used as the base online learner. All the parameters of the compared methods are set at the values suggested in the original literature.

For the parameters of ACDWM, the threshold to remove the outdated classifier θ is set at 0.001. As discussed in [37], the value of θ has a negligible influence on the accuracy and affects only the number of stored classifiers. G-mean error $\epsilon_{gm} = 1 - (\text{TPR} \cdot \text{TNR})^{1/2}$ is chosen as the error function used in Learn++.NIE and ACDWM, where TPR is the true positive rate and TNR is the true negative rate. Any other error functions, such as the F1-score, can also be used as the error function. For the chunk size selection module in ACDWM, we set the ensemble size $T = 100$, the forest pool size $Q = 1000$, the number of simulations $P = 250$, the significance level $\Delta = 0.05$, the testing size $n_t = 10$, and the window size $d = 100$. Aside from the window size d, the parameters are better if larger, as this generates meaningful variance for statistical testing. For each enlarged chunk, we produce Q trees and select T from them P times for calculating the variance, where T is the ensemble size and is arbitrarily set as a large number. Q should be set as several times of T to ensure that each repeated ensemble has different combinations. The value of P determines the quality of the estimated variance and $P = 250$ is the value adopted in [39] to show the histogram. The window size d is set at 100, which is regarded as a small chunk size. The testing size n_t is also

used to ensure the reliability of the statistical test of variance. A small number is enough for Fisher's method to use for a combination of the p-values for each testing sample.

Python 3.6 is used as the tool to implement all methods on all experiments.[3] classification and regression tree (CART) [43] is used as *BaseLearner* of the individual classifier for all methods, where the default parameters provided by *scikit-learn* learning library in Python are used [44]. All experimental results are the averages of ten runs. The test-then-train strategy is adopted to evaluate the performance of the methods on each chunk. As the chunk size of ACDWM is not fixed and online methods are also compared, it is not straightforward to compare the accuracy per chunk as shown in [4]. Therefore, we calculate the prequential minority class recall $\text{Rec}_i = TP_i/P_i$ if $P_i < N_i$, $\text{Rec}_i = TN_i/N_i$ if $P_i > N_i$ and the prequential G-mean $G\text{-}mean_i = (TP_i/P_i \cdot TN_i/N_i)^{1/2}$, where TP_i and TN_i are the true positives and true negatives from the beginning to the ith time step, and P_i and N_i are the accumulated numbers of positive and negative class samples [15].

B. Data Sets

In the experiments, six synthetic and two real streaming data sets are used to evaluate the performance of the proposed method and other methods as follows.

1) *Moving Gaussian [21]:* This data set consists of two Gaussian distributed classes with identity covariance and two dimensions. The initial coordinates of the mean of the two classes are [5,0] and [7,0]. They gradually move to [−5,0] and [−3,0] between the beginning and halfway through the stream and then move back to the initial coordinates.
2) *Drifting Gaussian [4]:* This data set is a linear combination of three Gaussian components where one is the minority class. The mean and the variance of the Gaussian components are varying throughout time.
3) *SEA [45]:* This data set contains three attributes ranging from 0 to 10. Only the first two attributes are related to the class that is determined by $attr_1 + attr_2 \leq \alpha$. The third attribute is treated as noise. The control parameter α is set at 15 for the first and the last third of the chunks and 5 for the second third of the chunks.
4) *Hyper Plane [46]:* In this data set, the gradually changed concepts are calculated by $f(\mathbf{x}) = \sum_{i=1}^{d-1} a_i \cdot ((x_i + x_{i+1})/x_i)$, where the dimension $d = 10$ and a_i is used to control the decision hyperplane.
5) *Spiral [4]:* This data set comprises four spirals rotating with a size-fixed 2-D window. The position of the spirals is used to predict the class.
6) *Checkerboard [4]:* This is a nonlinear XOR classification problem. The data set is produced by selecting from a size-fixed window in the rotating checkerboard.
7) *Electricity [37]:* This data set contains the changes in electricity price according to the time and demand in New South Wales, Australia. The class label is determined by the change of price over the last 24 h.
8) *Weather [4]:* This data set contains the weather information over 50 years in Bellevue and Nebraska, USA. The task is to predict whether a day is rainy or not.

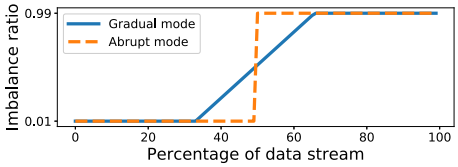

Fig. 1. Two prior drift modes used in the experiments.

TABLE I
INFORMATION OF SIX STREAMING DATA SETS

Dataset	#features	#samples		
		original	abrupt	gradual
Drifting Gaussian	2	100,000	49,693	51,893
Moving Gaussian	2	100,000	49,995	56,392
SEA	3	100,000	48,780	47,118
Hyper Plane	10	100,000	49,741	55,817
Spiral	2	100,000	49,995	56,392
Checkerboard	2	100,000	50,155	56,567
Electricity	7	27,549	13,924	15,519
Weather	8	18,159	9,318	9,931

The synthetic data sets are generated by the concept drift data generator.[4] These data sets contain the different kinds of real drift and virtual drift, but their prior distribution does not change a large amount across the stream. To increase their complexity to be close to the definition of joint concept drift, we incorporate the prior drift into the data stream by manually adjusting the imbalance ratio on all data sets with undersampling. In the experiments, the imbalance ratio is changed by two prior drift modes [15] as follows.

1) *Abrupt Drift:* The imbalance ratio is initially set at 0.01. After half of the data stream, the imbalance ratio suddenly changes to 0.99, i.e., the majority class becomes the minority class with an imbalance ratio of 0.01. The prequential measures are reset at the position of the abrupt drift.
2) *Gradual Drift:* The imbalance ratio is initially set at 0.01. After one-third of the data stream, the imbalance ratio starts to gradually increase until it reaches 0.99 at the two-thirds of the data stream. The prequential measures are reset at the starting and ending positions of the gradual drift.

The imbalance ratio here refers to the percentage of positive class samples. To control the imbalance ratio, undersampling is performed on every 1000 samples in the original data stream. The majority class is undersampled if the original imbalance ratio on this chunk is smaller than the assigned imbalance ratio, and the minority class is undersampled if the assigned imbalance ratio is smaller than the original imbalance ratio on this chunk. As the original imbalance ratio of each data set is different, the drift position after undersampling is also different. The drift modes are shown in Fig. 1. The information of the six data sets is summarized in Table I.

C. Experimental Results

1) General Comparison: The prequential G-mean at each time step in abrupt drift mode is shown in Fig. 2. The reset

[3]The code is available at https://github.com/jasonyanglu/ACDWM

[4]http://users.rowan.edu/ polikar/research/NIE_data/

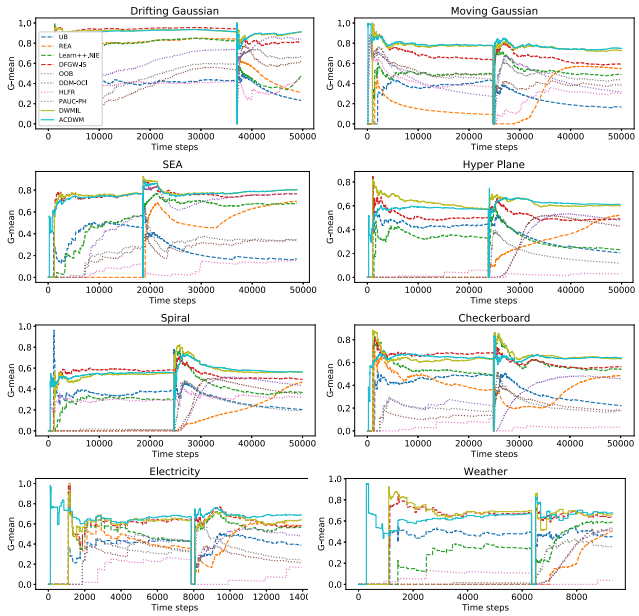

Fig. 2. Prequential G-mean performance of each time step in abrupt drift mode.

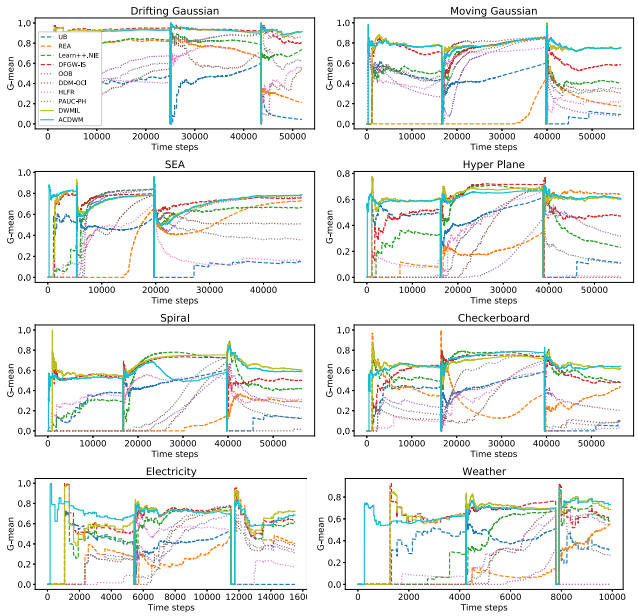

Fig. 3. Prequential G-mean performance of each time step in gradual drift mode.

of the prequential measure after the abrupt drift is shown as vertical lines in Figs. 2–5. Generally, ACDWM produces stable G-mean results for most of the data sets before and after the abrupt drift. In most of these cases, ACDWM has better or comparable G-mean performance with DWMIL. However, it can be observed on the data set Hyper Plane that DWMIL performs better than ACDWM before the abrupt drift, while after the drift, ACDWM has more stable and better G-mean performance than DWMIL. This indicates the ability of ACDWM to manage prior concept drift. A similar phenomenon can be observed when compared with DFGW-IS. In the data sets Drifting Gaussian, Spiral, Checkerboard, and WeatherDFGW-IS performs better than ADCWM before the drift but worse after the drift. This indicates that the methods such as DFGW-IS that save the past minority class samples suffer from prior drift because the concepts of the minority and majority classes are switched.

Compared with the performance of DWMIL on the data set Electricity, ACDWM demonstrates high performance from the beginning and is superior to DWMIL to the end of the data stream. ACDWM rapidly determines the proper chunk size using its adaptive chunk size selection module, while DWMIL has to buffer the data until reaching the preset chunk size, which results in that ACDWM learns better than DWMIL. The results of Learn++.NIE, UB, and REA for these data sets are not stable. In addition, the online methods rarely produce good results from highly imbalanced data streams with concept drift. OOB tends to outperform other drift detection-based online methods because false detection may lead to frequent model resetting, which worsens the performance of these methods.

The prequential G-mean on each time step in gradual drift mode is shown in Fig. 3. In most of the data sets, ACDWM and DWMIL produce the best results. However, compared with the experiments in abrupt drift mode, ACDWM does not perform significantly differently to DWMIL in gradual drift mode. The possible reason for this is that when the class prior is gradually changed, the adaptive chunk size selection module in ACDWM is of limited use for classification, as during the gradual drift, the classes become relatively balanced, and thus, the selection of the optimal chunk size to collect sufficient minority class samples is of little utility. Moreover, the adaptive chunk size selection module tends to select small chunks in this situation, which performs less well than large chunks when the data in the chunk are balanced. In summary, ACDWM outperforms its competitors in most of the cases, and it performs better in abrupt drift mode than in gradual drift mode.

The prequential minority class recall at each time step in abrupt drift mode is shown in Fig. 4. Overall, ACDWM and DWMIL perform well both before and after the abrupt drift. ACDWM shows more stable minority class recall than does DWMIL with the data sets Moving Gaussian, Hyper Plane, and Electricity after the abrupt drift. ACDWM outperforms DWMIL for the data sets Spiral and Checkerboard before the abrupt drift and has comparable performance after the abrupt drift. REA, OOB, and DDM-OCI show high minority class recall after the abrupt drift with most of the data sets. This is because they have low minority class recall before the abrupt drift, which is caused by predicting most of the majority class samples to the minority class samples. Therefore, they obtain high minority class recall after the abrupt drift when the majority class becomes the minority class. However, several time steps after the abrupt drift, their minority class recall performances drop greatly because they do not learn the minority class well.

The prequential minority class recall at each time step in abrupt drift mode is shown in Fig. 5. ACDWM and DWMIL show stable performance in all the three stages of the gradual drift, indicating that these two methods learn well when the class prior is gradually changing. REA and DFGW-IS are two typical examples of poor adaption to data sets with gradual

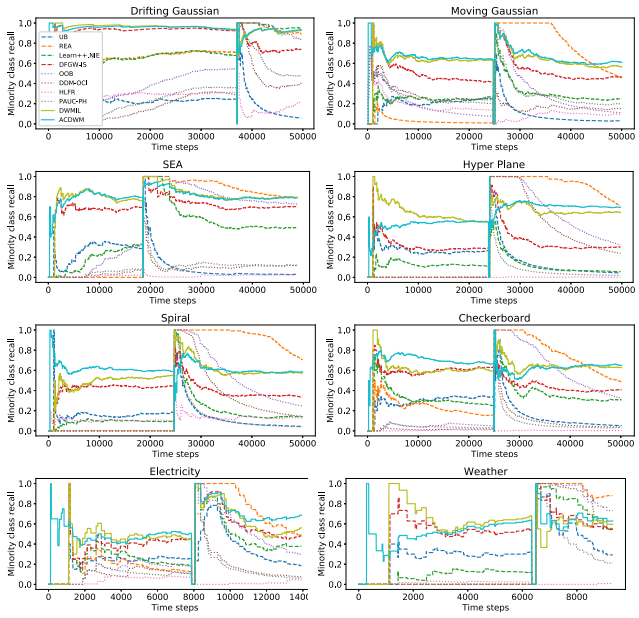

Fig. 4. Prequential minority class recall performance of each time step in abrupt drift mode.

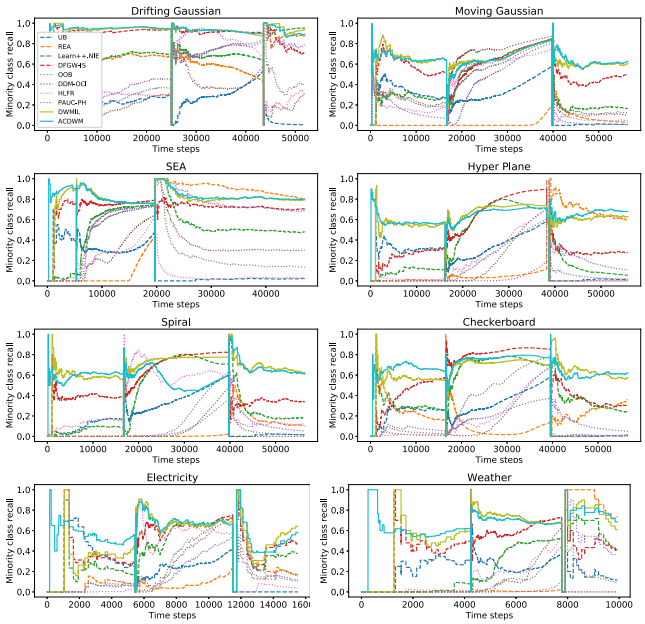

Fig. 5. Prequential minority class recall performance of each time step in gradual drift mode.

drift. Thus, REA shows high minority class recall on the data sets SEA, Hyper Plane, and Weather after the ending position of the gradual drift, but low performance before it, for the same reason as the abrupt drift case. DFGW-IS performs well before the ending position of the gradual drift on most of the data sets but drops significantly after the reset. This is because it is generally biased toward the minority class. Thus, it struggles to predict the minority class after the ending position of the gradual drift. In summary, it can be observed from Figs. 2–5 that ACDWM and DWMIL generate more stable G-mean and minority class recall than other methods in both prior drift modes. ACDWM shows better performance than DWMIL in abrupt drift mode and has comparable performance in gradual drift mode on both G-mean and minority class recall.

The numerical results are shown in Tables II–V. The results are obtained by the final-step prequential performance. The average rank is calculated and shown at the bottom of each table. The one-tailed Nemenyi test [47] is used to determine if the ranks of two methods are significantly different, with the boldface indicating the best result. The bracket in the last row shows the p-value of the Nemenyi test between its rank and the first rank, and the underline indicates the statistical significance at a confidence level of 0.05.

The results of prequential G-mean are shown in Tables II and III. For the data sets in abrupt drift mode, ACDWM performs best on all data sets, with DWMIL ranking second. According to the average rank, ACDWM significantly outperforms all the other methods except DWMIL. For the results in gradual drift mode shown in Table III, DWMIL achieves the best results in four examples, while ACDWM does so in three examples and REA in one example. However, the average rank shows that DWMIL and ACDWM have the same rank. In contrast to the results for abrupt drift mode, the performance of ACDWM is comparable with that of DWMIL in gradual drift mode, which means that the adaptive chunk size selection module is more effective in abrupt drift mode than that in gradual drift mode; in the latter, ACDWM seems to degenerate back to DWMIL.

We also show the prequential minority class recall in Tables IV and V. For the data sets in abrupt drift mode, ACDWM and REA are both ranked the first in three examples. The good minority class recall of REA is due to its accumulation of minority class samples and the use of nearest neighbors to select from the stored set. Although REA and ACDWM both win three times, ACDWM gets higher average rank. For the results in gradual drift mode shown in Table V, ACDWM achieves the best results in three examples and also has the best average rank. To sum up, ACDWM performs better in G-mean than in minority class recall. One possible reason for this is that the error function used in ACDWM is G-mean instead of minority class recall. However, ACDWM still ranks well in minority class recall.

2) Effectiveness of Adaptive Chunk Size Selection: In this section, we demonstrate the effectiveness of our adaptive chunk size selection module in ACDWM. Thus far, ACDWM is the first chunk size selection method for ensemble methods processing imbalanced streaming data. There is no similar method with the same purpose that is comparable to ACDWM. We therefore compare ACDWM with the fixed-size-chunk methods and several adaptive window methods. After the chunk size is determined, the samples in the chunk will be trained and predicted by DWMIL, which is the fixed-size-chunk version of ACDWM. Fixed Chunk 100 (FC100) and Fixed Chunk 1000 (FC1000) adopt the strategy by fixing the chunk size at 100 and 1000. When the chunk only consists of one class, the current chunk is merged with the next chunk until there are two classes present in the merged chunk. Fixed minority (FM) buffers the incoming data until a fixed number of minority class samples, which is set at 20, is achieved. ADWIN [36] and PERM [30] are the concept drift detection methods. All the parameters are used with the suggested values in the literature. A maximum chunk size is

TABLE II

PERFORMANCE OF FINAL STEP G-MEAN OF DATA SETS IN ABRUPT DRIFT MODE

Dataset	UB	REA	Learn++.NIE	DFGW-IS	OOB	DDM-OCI	HLFR	PAUC-PH	DWMIL	ACDWM
Drifting Gaussian	0.2324	0.3069	0.4761	0.8086	0.8384	0.6204	0.4453	0.6663	0.9094	**0.9134**
Moving Gaussian	0.1661	0.5452	0.4891	0.5913	0.3161	0.3826	0.3011	0.4350	0.7259	**0.7479**
SEA	0.1693	0.6985	0.6784	0.7684	0.7717	0.3413	0.1536	0.3449	0.8031	**0.8041**
Hyper Plane	0.2059	0.5222	0.2392	0.4904	0.4797	0.4299	0.0285	0.1200	0.5996	**0.6113**
Spiral	0.2041	0.4678	0.3669	0.4927	0.4351	0.3493	0.3179	0.1925	0.5628	**0.5634**
Checkerboard	0.2219	0.4854	0.5409	0.5626	0.4577	0.1797	0.0337	0.1766	0.6331	**0.6396**
Electricity	0.3877	0.5850	0.5676	0.5806	0.4804	0.2150	0.1662	0.2393	0.6394	**0.6888**
Weather	0.4527	0.5113	0.5904	0.6392	0.5452	0.5276	0.0377	0.3602	0.6609	**0.6781**
Average rank	8.5	5	5.5	3.5	5.13	7.13	9.375	7.88	2	1
	(0.0000)	(0.0041)	(0.0015)	(0.0493)	(0.0032)	(0.0000)	(0.0000)	(0.0000)	(0.2544)	

TABLE III

PERFORMANCE OF THE FINAL STEP G-MEAN OF DATA SETS IN GRADUAL DRIFT MODE

Dataset	UB	REA	Learn++.NIE	DFGW-IS	OOB	DDM-OCI	HLFR	PAUC-PH	DWMIL	ACDWM
Drifting Gaussian	0.0450	0.2100	0.7373	0.8002	0.8675	0.5564	0.5580	0.6331	**0.9159**	0.9137
Moving Gaussian	0.0932	0.1739	0.4055	0.5871	0.0798	0.3082	0.2328	0.3570	**0.7558**	0.7535
SEA	0.1493	0.7314	0.6682	0.7590	0.7679	0.5103	0.1612	0.3555	0.7829	**0.7847**
Hyper Plane	0.1125	**0.6366**	0.2304	0.4712	0.3162	0.0000	0.0079	0.1060	0.6001	0.6076
Spiral	0.1217	0.2889	0.4159	0.4992	0.2240	0.0156	0.3093	0.1277	**0.6031**	0.5870
Checkerboard	0.0993	0.4405	0.4798	0.4808	0.2015	0.0890	0.0297	0.1057	0.6165	**0.6412**
Electricity	0.0000	0.4040	0.5939	0.6192	0.3388	0.3818	0.1741	0.3142	**0.7229**	0.6862
Weather	0.3176	0.5566	0.5818	0.6002	0.5522	0.0000	0.0000	0.2676	0.6869	**0.7307**
Average rank	8.75	5.5	4.63	3.38	5.88	8.13	8.25	7.25	1.63	1.63
	(0.0000)	(0.0052)	(0.0238)	(0.1238)	(0.0025)	(0.0000)	(0.0000)	(0.0001)	(0.5000)	

TABLE IV

PERFORMANCE OF THE FINAL-STEP MINORITY CLASS RECALL OF DATA SETS IN ABRUPT DRIFT MODE

Dataset	UB	REA	Learn++.NIE	DFGW-IS	OOB	DDM-OCI	HLFR	PAUC-PH	DWMIL	ACDWM
Drifting Gaussian	0.0580	0.8900	**0.9490**	0.7410	0.8290	0.4050	0.2160	0.4835	0.9270	0.9330
Moving Gaussian	0.0292	0.4600	0.2460	0.4628	0.1092	0.1488	0.0924	0.1972	0.5644	**0.6097**
SEA	0.0301	0.7899	0.4949	0.7051	0.7290	0.1181	0.0254	0.1210	**0.7957**	0.7895
Hyper Plane	0.0445	**0.7101**	0.0601	0.3017	0.3239	0.2378	0.0021	0.0193	0.6445	0.6945
Spiral	0.0440	**0.7080**	0.1404	0.3368	0.2468	0.1440	0.1248	0.0408	0.5740	0.5860
Checkerboard	0.0517	0.4858	0.3073	0.4065	0.3241	0.0379	0.0030	0.0328	0.6272	**0.6481**
Electricity	0.1855	0.4891	0.3800	0.4800	0.2964	0.0745	0.0436	0.0600	0.5600	**0.6866**
Weather	0.2920	**0.8800**	0.5520	0.5440	0.7320	0.5560	0.0080	0.2120	0.5960	0.6280
Average rank	8.13	2.38	5.38	4.75	4.88	7	10	8.13	2.5	1.88
	(0.0000)	(0.3706)	(0.0104)	(0.0290)	(0.0238)	(0.0004)	(0.0000)	(0.0000)	(0.3411)	

TABLE V

PERFORMANCE OF THE FINAL-STEP MINORITY CLASS RECALL OF DATA SETS IN GRADUAL DRIFT MODE

Dataset	UB	REA	Learn++.NIE	DFGW-IS	OOB	DDM-OCI	HLFR	PAUC-PH	DWMIL	ACDWM
Drifting Gaussian	0.0061	0.8788	**0.9530**	0.7030	0.7894	0.3258	0.3348	0.4076	0.9364	0.8924
Moving Gaussian	0.0091	0.0303	0.1667	0.5030	0.0067	0.1018	0.0576	0.1303	0.6067	**0.6284**
SEA	0.0228	**0.8108**	0.4822	0.7081	0.6884	0.3023	0.0286	0.1321	0.7931	0.8004
Hyper Plane	0.0128	0.5904	0.0558	0.2750	0.1051	0.0000	0.0006	0.0179	0.6276	**0.6800**
Spiral	0.0152	0.1030	0.1794	0.3376	0.0527	0.0012	0.1188	0.0176	**0.6212**	0.6107
Checkerboard	0.0103	0.3590	0.2417	0.3000	0.0500	0.0103	0.0032	0.0154	0.5744	**0.6221**
Electricity	0.0000	0.1667	0.3778	0.4694	0.1167	0.1611	0.0389	0.1028	**0.6444**	0.5833
Weather	0.1158	**0.7368**	0.4158	0.4053	0.3632	0.0000	0.0000	0.0947	0.6105	0.6842
Average rank	8.88	3.88	4.25	4	6.38	8.25	8.38	7.25	2	1.75
	(0.0000)	(0.0802)	(0.0493)	(0.0686)	(0.0011)	(0.0000)	(0.0000)	(0.0001)	(0.4344)	

set at 1000 to avoid the chunk continuously growing if no drift is detected.

Tables VI and VII show the result of a Wilcoxon signed-rank test [48] between ACDWM and DWMIL with different chunk size selection methods on the eight data sets, in abrupt and gradual drift modes. The test measures the difference between the performances of two methods in processing multiple data sets. The sum of ranks of each method is calculated by $R^+ = \sum_{d_i>0} \text{rank}(|d_i|) + (1/2)\sum_{d_i=0} \text{rank}(|d_i|)$ and $R^- = \sum_{d_i<0} \text{rank}(|d_i|) + (1/2)\sum_{d_i=0} \text{rank}(|d_i|)$, where R^+ is the rank sum of ACDWM and R^- is the rank sum of the compared method. By investigating the smaller value between R^+ and R^-, we can obtain the p-value that indicates the significance level of the superiority, namely the extent to which the adaptive chunk size selection improves the performance of the size-fixed methods and other sliding window methods on the imbalanced data stream with concept drift.

The test results of data sets in abrupt drift mode are shown in Table VI. The underline indicates the statistical significance at a confidence level of 0.05. All R^+ are larger than R^- in Table VI, which indicates that the adaptive chunk selection module in ACDWM is better than the other methods for determining the chunk size for imbalanced streaming data with concept drift. In particular, ACDWM shows significant improvements on FC1000 and PERM in G-mean, and ADWIN and PERM in positive class recall because it performs

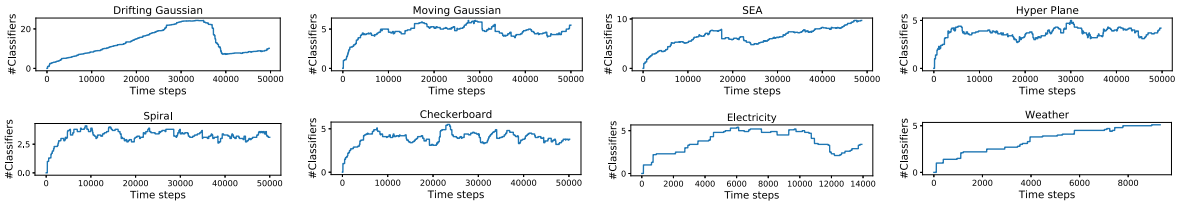

Fig. 6. Average number of classifiers kept in the ensemble of ACDWM in abrupt drift mode.

TABLE VI

G-MEAN RESULT OF WILCOXON SIGNED-RANK TEST FOR COMPARING ACDWM WITH OTHER CHUNK SIZE SELECTION METHODS ON DATA SETS IN ABRUPT DRIFT MODE

	G-mean			Positive class recall		
	R^+	R^-	p-value	R^+	R^-	p-value
FC100	28	8	0.0807	28	8	0.0807
FC1000	36	0	0.0000	29	7	0.0617
FM	23	13	0.2419	27	9	0.1038
ADWIN	24	12	0.2004	36	0	0.0000
PERM	36	0	0.0000	36	0	0.0000

TABLE VII

G-MEAN RESULT OF WILCOXON SIGNED-RANK TEST FOR COMPARING ACDWM WITH OTHER CHUNK SIZE SELECTION METHODS ON DATA SETS IN GRADUAL DRIFT MODE

	G-mean			Positive class recall		
	R^+	R^-	p-value	R^+	R^-	p-value
FC100	29	7	0.0617	21	15	0.3372
FC1000	17	19	0.5557	22	14	0.2877
FM	21	15	0.3372	20	16	0.3897
ADWIN	19	17	0.4443	27	9	0.1038
PERM	24	12	0.2004	26	10	0.1313

best at processing all data sets in abrupt drift mode. The test results of data sets in gradual drift mode are shown in Table VII. ACDWM shows ranks higher than all methods except FC1000 in G-mean, where it shows a comparable result. It is worth noting that none of the test results in Table VII are significant (p-value less than 0.05), and it can be observed that the p-values in Table VII are generally larger than that in Table VI. Therefore, it is again seen that ACDWM performs better in abrupt drift mode than in gradual drift mode. This is because ACDWM is designed for processing imbalanced data streams and can thus rapidly adapt to the abrupt drift when the minority class and the majority class switch roles. For gradual drift, the data stream tends to be relatively balanced for some time, and the arbitrary chunk size for balanced data stream is more likely to produce stable results, meaning that the advantages of ACDWM are not obvious in this situation.

From this experiment, it can be verified that the methods with a size-fixed chunk, i.e., FC100 and FC1000, are in some cases able to produce better results because the best chunk-size may exist for each data set within some time interval; some data sets fit small chunk and some other fit large chunk. However, this is unknown unless repeatedly trying the chunk size as a tuning parameter. This explanation can also be applied to the results of the ADWIN and PERM drift detection methods because they occasionally produce good results, e.g., if the drifts of a data set are successfully detected and the chunk is able to produce stable classifiers. However, in some other cases, it is likely that the classifier created on the current chunk is not stable when the drift is detected. Therefore, although ACDWM does not produce the best results in every case, the results deriving from its chunk size selection mechanism are promising for most situations.

3) Effectiveness of Drift Reaction: To show how the weight decay mechanism works in ACDWM, we plotted the average number of classifiers kept in the ensemble over ten runs in Fig. 6. The reaction to the prior drift can be observed in the data sets Drifting Gaussian and SEA. For Drifting Gaussian, the number of classifiers continuously increases until around time step 36 000, which is the abrupt drift point as shown in Fig. 2. After the drift, the number of classifiers decrease to approximately 6 and then increases again. For the data set SEA, the decrease begins approximately at time step 18 000, which is also when the abrupt prior drift occurs. The decrease is also steep, which means that several classifiers are removed at the same time; after this, the number of classifiers increases again. From these observations, it is seen that ACDWM does not react to every prior drift case. If the prior drift does not influence the performance of the classifier, ACDWM will remain unchanged. For the data sets Moving Gaussian, Hyper Plane, Spiral, and Checkerboard, it can be found that the number of classifiers is not stable. This is because the posterior drift occurs continuously in these data sets. Thus, the number of classifiers kept in the ACDWM is adjusted frequently to adapt to the changes. This experiment shows that the weight adjustment mechanism of ACDWM can also cope well with joint concept drift.

4) Computational Time Analysis: The running times of ACDWM with various parameters are shown in Fig. 7, compared with the other chunk size selection methods for the data set Drifting Gaussian in abrupt drift mode. When tuning one parameter, other parameters are set at the values in Section IV-A. It can be observed that the running time of FC1000 is the lowest because it selects the size-fixed chunk for training. The computational costs of ADWIN and PERM are much higher: ADWIN needs to check every split on a sequence for every incoming data, while PERM is like ACDWM in that it needs to build a set of classifiers on samples with different permutations. However, each classifier built by PERM is an ensemble of trees that cannot be reused because the training data change at each permutation. In contrast, ACDWM only needs to train a pool of individual trees and randomly combines them for its prediction variance test. The running time of ACDWM linearly increases as Q increases and stabilizes when the other parameters increased. This observation is consistent with the computational analysis in Section III-C. Training Q classifiers is most expensive, while inference with random permutation has a minimal influence on the computational cost.

5) Parameter Sensitivity: The experiments of parameter sensitivity with the data sets in abrupt drift mode are shown

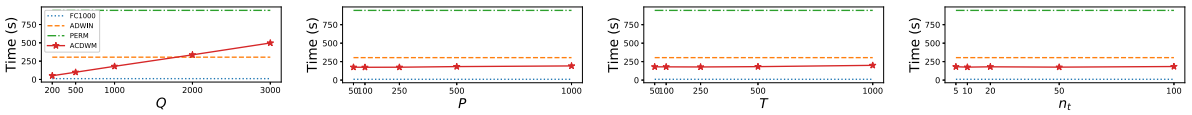

Fig. 7. Running time of ACDWM with different parameters compared with other chunk size selection methods on the data set Drifting Gaussian in abrupt drift mode.

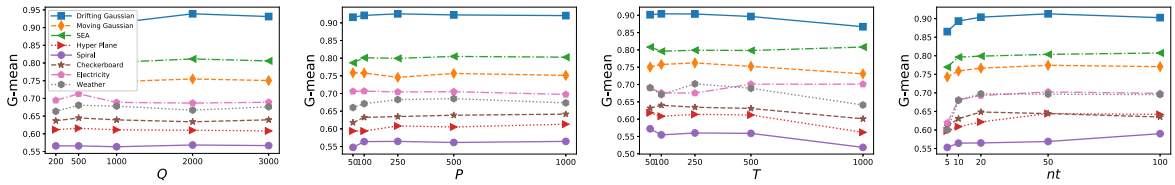

Fig. 8. Performance of ACDWM on final-step G-mean with different values of parameters in abrupt drift mode.

in Fig. 8. Each parameter is tuned, while the others are set at the values in Section IV-A. The results are the average of ten repeated runs. It can be seen that the forest pool size Q does not improve the G-mean performance as it increases because Q only determines how many trees are generated in total. As the number of simulations P increases from 50 to 100 and the testing size n_t increases from 5 to 20, the G-mean performance slightly increases and then stabilizes. This is because the small P and n_t values may lead to unreliable statistical test results for selecting proper chunk size. For the ensemble size T, it can be observed that G-mean decreases as T increases from 500 to 1000. When T is set as 1000, which is equal to the forest pool size Q, each ensemble takes all individual classifiers, and thus, P times simulations have identical results. In this case, the selection of chunk size does not enhance the creation of stable classifiers. Therefore, it is appropriate to set T at a small fraction of Q. In summary, ACDWM is not sensitive to the selection of the parameters, and the suggested parameters in Section IV-A can generally produce good results.

V. Conclusion

Concept drift and class imbalance are two inevitable problems with learning from data streams, which must be dealt with for data to be practically useful. In this article, we propose and develop the use of ACDWM to solve the problem of learning from imbalanced data streams with concept drift. ACDWM creates an individual classifier for each chunk and weights these according to their performance on the current chunk. Thus, a classifier trained recently or on a similar concept as the current chunk will receive a high weighting in the ensemble to assist with prediction. In the meantime, ACDWM is able to select the proper chunk size, which removes the problems inherent to the use of fixed chunk sizes. ACDWM adaptively compares the stability of the classifier trained on the current chunk with that of the enlarged chunk until there is no significant stability increase. The stability is measured by the variance of normally distributed predictions produced by a modified bagging algorithm. Experiments on the data sets with joint concept drift have shown that ACDWM outperforms its counterparts and the adaptive chunk size selection module is effective.

References

[1] J. Gama, I. V. Z. E. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, p. 44, Apr. 2014.

[2] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, Nov. 2015.

[3] C. Chen, Y. Wang, J. Zhang, Y. Xiang, W. Zhou, and G. Min, "Statistical features-based real-time detection of drifted twitter spam," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 914–925, Apr. 2017.

[4] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2283–2301, Oct. 2013.

[5] T. R. Hoens and N. V. Chawla, "Learning in non-stationary environments with class imbalance," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. 2012, pp. 168–176.

[6] S. Wang, L. L. Minku, and X. Yao, "A systematic study of online class imbalance learning with concept drift," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4802–4821, Oct. 2018.

[7] 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. 04, 2013, Art. no. 1340001.

[8] J. L. Lobo, I. Laña, J. Del Ser, M. N. Bilbao, and N. Kasabov, "Evolving spiking neural networks for online learning over drifting data streams," *Neural Netw.*, vol. 108, pp. 1–19, Dec. 2018.

[9] P. R. L. Almeida, L. S. Oliveira, A. S. Britto, and R. Sabourin, "Adapting dynamic classifier selection for concept drift," *Expert Syst. Appl.*, vol. 104, pp. 67–85, Aug. 2018.

[10] H. M. Gomes *et al.*, "Adaptive random forests for evolving data stream classification," *Mach. Learn.*, vol. 106, nos. 9–10, pp. 1469–1495, 2017.

[11] S. Mohamad, A. Bouchachia, and M. Sayed-Mouchaweh, "A bi-criteria active learning algorithm for dynamic data streams," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 74–86, Jan. 2018.

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

[13] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM Comput. Surv.*, vol. 49, no. 2, p. 31, Nov. 2016.

[14] S. Wang, L. L. Minku, D. Ghezzi, D. Caltabiano, P. Tino, and X. Yao, "Concept drift detection for online class imbalance learning," in *Proc. Int. Joint Conf. Neural Netw.*, Dallas, TX, USA, Aug. 2013, pp. 1–10.

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

[16] S. Yu and Z. Abraham, "Concept drift detection with hierarchical hypothesis testing," in *Proc. SIAM Int. Conf. Data Mining*, Houston, TX, USA, 2017, pp. 768–776.

[17] D. Brzezinski and J. Stefanowski, "Prequential AUC for classifier evaluation and drift detection in evolving data streams," in *Proc. 3rd Int. Conf. New Frontiers Mining Complex Patterns*. Nancy, France: Springer, 2014, pp. 87–101.

[18] J. Gao, W. Fan, J. Han, and P. S. Yu, "A general framework for mining concept-drifting data streams with skewed distributions," in *Proc. SIAM Int. Conf. Data Mining*, Minneapolis, MN, USA, 2007, pp. 3–14. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611972771.1

[19] S. Chen and H. He, "Towards incremental learning of nonstationary imbalanced data stream: A multiple selectively recursive approach," *Evolving Syst.*, vol. 2, no. 1, pp. 35–50, 2011.
[20] K. Wu, A. Edwards, W. Fan, J. Gao, and K. Zhang, "Classifying imbalanced data streams via dynamic feature group weighting with importance sampling," in *Proc. SIAM Int. Conf. Data Mining*, Philadelphia, PA, USA, 2014, pp. 722–730.
[21] Y. Lu, Y.-M. Cheung, and Y. Y. Tang, "Dynamic weighted majority for incremental learning of imbalanced data streams with concept drift," in *Proc. 26th Int. Joint Conf. Artif. Intell.* Melbourne, VIC, Australia: AAAI Press, Aug. 2017, pp. 2393–2399.
[22] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, Sep. 2017.
[23] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Comput. Surv.*, vol. 50, no. 2, p. 23, 2017.
[24] S. Wang, L. L. Minku, and X. Yao, "A learning framework for online class imbalance learning," in *Proc. IEEE Symp. Comput. Intell. Ensemble Learn. (CIEL)*, Apr. 2013, pp. 36–45.
[25] A. Ghazikhani, R. Monsefi, and H. S. Yazdi, "Recursive least square perceptron model for non-stationary and imbalanced data stream classification," *Evolving Syst.*, vol. 4, no. 2, pp. 119–131, 2013.
[26] S. Wang, L. L. Minku, and X. Yao, "Dealing with multiple classes in online class imbalance learning," in *Proc. 25th Int. Joint Conf. Artif. Intell.* New York, NY, USA. IJCAI/AAAI Press, 2016, pp. 2118–2124.
[27] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, 2001, pp. 359–364.
[28] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proc. InBrazilian Symp. Artif. Intell.* São Luís, Brazil: Springer, 2004, pp. 286–295.
[29] H. Wang and Z. Abraham, "Concept drift detection for streaming data," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Killarney, Ireland, Jul. 2015, pp. 1–9.
[30] M. Harel, K. Crammer, R. El-Yaniv, and S. Mannor, "Concept drift detection through resampling," in *Proc. 31st Int. Conf. Mach. Learn.* Beijing, China, 2014, pp. 1009–1017.
[31] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Mach. Learn.*, vol. 90, no. 3, pp. 317–346, 2013.
[32] Y. Sun, K. Tang, L. L. Minku, S. Wang, and X. Yao, "Online ensemble learning of data streams with gradually evolved classes," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 6, pp. 1532–1545, Jun. 2016.
[33] S. Chen and H. He, "SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining," in *Proc. Int. Joint Conf. Neural Netw.*, Atlanta, GA, USA, 2009, pp. 522–529.
[34] S. Ren, B. Liao, W. Zhu, Z. Li, W. Liu, and K. Li, "The gradual resampling ensemble for mining imbalanced data streams with concept drift," *Neurocomputing*, vol. 286, pp. 150–166, Apr. 2018.
[35] B. Mirza, Z. Lin, and N. Liu, "Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift," *Neurocomputing*, vol. 149, pp. 316–329, Feb. 2015.
[36] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. SIAM Int. Conf. Data Mining*, Minneapolis, MN, USA, 2007, pp. 443–448.
[37] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *J. Mach. Learn. Res.*, vol. 8, pp. 2755–2790, Dec. 2007.
[38] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.
[39] L. Mentch and G. Hooker, "Quantifying uncertainty in random forests via confidence intervals and hypothesis tests," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 841–881, 2016.
[40] E. W. Frees, "Infinite order U-statistics," *Scand. J. Statist.*, vol. 16, no. 1, pp. 29–45, 1989.
[41] G. W. Snedecor and W. G. Cochran, *Statistical Methods*, 8th ed. Ames, IA, USA: Iowa State Univ. Press, 1989.
[42] R. A. Fisher, "Statistical methods for research workers," in *Breakthroughs in Statistics*. Springer, 1992, pp. 66–70.
[43] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.
[44] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[45] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, 2001, pp. 377–382.
[46] X. Zhu. (2010). *Stream Data Mining Repository*. [Online]. Available: http://www.cse.fau.edu/~xqzhu/stream.html
[47] P. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Princeton Univ., Princeton, NJ, USA, 1963.
[48] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.

Yang Lu (S'13–M'19) received the B.Sc. and M.Sc. degrees in software engineering from the University of Macau, Macau, China, in 2012 and 2014, respectively, and the Ph.D. degree in computer science from Hong Kong Baptist University, Hong Kong, in 2019.

He is currently an Assistant Professor with the Department of Computer Science, School of Informatics, Xiamen University, Xiamen, China. He is also a Research Assistant with the Department of Computer Science, Hong Kong Baptist University. His current research interests include imbalanced data learning, clustering, ensemble learning, and online learning.

Yiu-Ming Cheung (SM'06–F'18) received the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

He is currently a Full Professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. His current research interests include machine learning, pattern recognition, visual computing, and optimization.

Dr. Cheung is a fellow of IET, British Computer Society (BCS), and Royal Society of Arts (RSA) and a Distinguished Fellow of International Engineering and Technology Institute (IETI). He is the Founding Chair of the Computational Intelligence Chapter of the IEEE Hong Kong Section and the Chair of the Technical Committee on Intelligent Informatics of the IEEE Computer Society. He serves as an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON CYBERNETICS, *Pattern Recognition*, *Knowledge and Information Systems*, and *Neurocomputing*, to name a few.

Yuan Yan Tang (S'88–M'88–SM'96–F'04–LF'16) received the B.Sc. degree in electrical and computer engineering from Chongqing University, Chongqing, China, the M.Eng. degree in electrical engineering from the Beijing Institute of Posts and Telecommunications, Beijing, China, and the Ph.D. degree in computer science from Concordia University, Montreal, QC, Canada.

He is currently a Chair Professor with the Faculty of Science and Technology, UOW College Hong Kong/Community College of City University, Hong Kong, and also an Emeritus Chair Professor with the Faculty of Science and Technology, University of Macau, Macau, China. He is also a Professor, an Adjunct Professor, and an Honorary Professor with Chongqing University, Concordia University, and Hong Kong Baptist University, Hong Kong, respectively. He has authored or coauthored over 400 academic articles, over 25 monographs, books, and book chapters. His current research interests include wavelets, pattern recognition, and image processing.

Dr. Tang is an International Association for Pattern Recognition (IAPR) Fellow. He was the General Chair, the Program Chair, and a committee member of many international conferences. He is the Founder and the Chair of the Pattern Recognition Committee of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS. He is the Founder and the General Chair of the series International Conferences on Wavelets Analysis and Pattern Recognition. He is the Founder and the Chair of the Macau Branch of the IAPR. He is the Founder and the Editor-in-Chief of the *International Journal of Wavelets, Multiresolution, and Information Processing* and an Associate Editor for several international journals.