



# DynaQ: online learning from imbalanced multi-class streams through dynamic sampling

Farnaz Sadeghi<sup>1</sup> · Herna L. Viktor<sup>1</sup>  · Parsa Vafaie<sup>1</sup>

Accepted: 14 July 2023 / Published online: 29 July 2023  
© The Author(s) 2023

## Abstract

Online supervised learning from fast-evolving data streams, particularly in domains such as health, the environment, and manufacturing, is a crucial research area. However, these domains often experience class imbalance, which can skew class distributions. It is essential for online learning algorithms to analyze large datasets in real-time while accurately modeling rare or infrequent classes that may appear in bursts. While methods have been proposed to handle binary class imbalance, there is a lack of attention to multi-class imbalanced settings with varying degrees of imbalance in evolving streams. In this paper, we present the Dynamic Queues (DynaQ) algorithm for online learning in multi-class imbalanced settings to fill this knowledge gap. Our approach utilizes a batch-based resampling method that creates an instance queue for each class to balance the number of instances. We maintain a queue threshold and remove older samples during training. Additionally, we dynamically oversample minority classes based on one of four rate parameters: recall, F1-score,  $\kappa_m$ , and Euclidean distance. Our learning algorithm consists of an ensemble that uses sliding windows and a soft voting schema while incorporating a drift detection mechanism. Our experimental results demonstrate the superiority of the DynaQ approach over state-of-the-art methods.

**Keywords** Online learning · Multi-class imbalance · Data streams · Ensembles · Concept drift

## 1 Introduction

In dynamic data streaming environments, online learning algorithms consider incoming examples “on arrival” without needing persistent storage and multiple scans, while maintaining a model that reflects the current data. This type of learning has applications in various real-world applications, such as network intrusion detection, spam filtering, fault diagnostics in manufacturing, and e-commerce applications [23]. Learning from such streaming data is challenging, especially in the presence of multiple skewed class distributions, also known as “multi-class imbalance”, where a large number of majority-class instances may lead to the minority classes

being ignored. This problem is aggravated in an online learning setting because a steady arrival of minority instances cannot be guaranteed, and a minority class may become a majority concept and vice versa [21]. In addition, evolving streams are susceptible to concept drifts, which is the phenomenon of unexpected changes in the underlying data distribution [35].

While previous studies have primarily focused on binary imbalanced data or stationary streams, only a limited number of studies have addressed the challenge of learning from evolving streams with multi-class imbalanced data [1]. For instance, [21] proposed modifications to resampling methods to adjust for multi-class imbalanced data. In addition, [14, 15] introduced ensemble-based approaches utilizing random feature subsets and resampling to address imbalances. However, further investigation is needed to explore classifier-agnostic approaches that can effectively utilize alternative evaluation metrics and resampling parameters to address the challenges of imbalanced datasets. Additionally, it is crucial to develop class-based drift detection methods to prevent performance deterioration.

This paper introduces an algorithm for online learning in a multi-class imbalance setting to address these shortcomings.

---

✉ Herna L. Viktor  
hviktor@uottawa.ca  
Farnaz Sadeghi  
fsade079@uottawa.ca  
Parsa Vafaie  
pvafa014@uottawa.ca

<sup>1</sup> School of Electrical Engineering and Computer Science, University of Ottawa, 800 King Edward Road, Ottawa, Ontario K1N 6N5, Canada

Our dynamic queues (DynaQ) approach extends the online-multi-class-queue (OMCQ) [46] approach that maintains separate queues for each class, without any form of sampling. In addition, our DynaQ algorithm uses oversampling with replacement based on a rate parameter associated with the various classes. That is, we oversample the minority classes based on the recall, F1-score,  $\kappa_m$  and Euclidean distance.

Our DynaQ algorithm combines batch-incremental and instance-incremental ensemble learning. Initially, a batch of data with the classes that have been seen so far is presented to the learner, and it subsequently updates the model with new instances as they arrive. Each ensemble member learns from a sliding batch, and the results are combined using soft voting. In addition, we incorporate a class-specific concept drift detection mechanism into DynaQ. Our algorithm can thus dynamically adapt to changes in label arrivals and individual class performances, as monitored by the recalls. Specifically, we do not make any assumptions regarding the frequency of classes, which implies that minority classes may become majority classes, and vice versa. Our experimental results confirm that our DynaQ algorithm is efficient in terms of multi-class concept separation.

The main contributions of this paper are as follows.

*Novel online queue-based ensemble architecture:* DynaQ utilizes a batch-based resampling method that creates an instance queue for each class and incorporates an ensemble approach that utilizes sliding windows and a soft voting schema.

*Self-handling class imbalance:* DynaQ dynamically oversamples minority classes based on a rate parameter associated with the various classes.

*Class-based concept drift approach:* DynaQ incorporating a drift detection mechanism to dynamically adapt to changes in label arrivals and individual class performances.

*Multi-class imbalanced and concept drift stream:* DynaQ highlights the challenges of learning from dynamic data streaming environments while addressing multi-class imbalance and concept drift in online learning.

The paper is organized as follows. Section 2 presents related work, while Section 3 introduces the DynaQ algorithm. Section 4 describes the experimental evaluation, and Section 5 concludes the paper.

## 2 Background and related work

In online learning, a data-generating process provides at each time step  $t$  a sequence of examples  $(x_t, y_t)$  from an unknown probability distribution, where  $x_t$  is a vector consisting of

qualitative or quantitative  $f$  features, and  $y_t \in Y$  is the class label, where  $Y = \{c_1, c_2, \dots, c_s\}$ , and  $S$  is the number of classes. An online classifier is built receiving an example  $x_t$  at time step  $t$ , resulting in a prediction  $\hat{y}_t$ . In a supervised learning setting, the label  $y_t$  is available, and the performance of a learning algorithm is evaluated using a loss function  $f(x_t) = l(y_t, \hat{y}_t)$  to find the best predictor for future data at each step [23]. This paper focuses on online learning from multi-class imbalanced data, where  $S > 2$ , from evolving streams.

### 2.1 Online class imbalance learning

In an online class imbalance learning setting, the main goal is to correctly classify minority examples because the minority class is often of most interest. Existing online learning approaches that address the class imbalance problem may be categorized into data-level, algorithmic and hybrid ensemble-based approaches. Several methods proposed for solving class imbalance problems in data-level techniques provide solutions, including resampling and feature selection [4]. Data-level modifications aim to balance the underlying dataset, making them classifier-agnostic approaches. Resampling is an effective data-level approach that proceeds independently of the learning algorithm; this method has been used for binary classification problems in the data stream setting. The major types of resampling are oversampling (increasing the number of minority-class examples), under-sampling (reducing the number of majority-class examples), and hybrid sampling [23]. Synthetic Minority Over-sampling Technique (SMOTE) [19] is a base idea for many resampling methods. SMOTE is an over-sampling technique that creates new instances of the minority class by interpolating between existing instances that are located close to each other. This method effectively reduces the degree of class imbalance compared to the original majority-to-minority class ratio. Recently, new modifications of the SMOTE method have been introduced that enable it to work compatible with streaming data [5–7]. The technique known as Selection-Based Resampling (SRE) [45], utilizes undersampling to eliminate safe instances from the majority class in an iterative manner, without causing reverse bias towards the minority class. However, undersampling methods may lead to crucial information being overlooked, whereas oversampling may potentially introduce artificial instances that may be deemed unacceptable in real-world domains. Hybrid ensemble-based methods combine resampling and algorithmic approaches to manage the class imbalance [13]. For example, in a study by [51], the authors integrated resampling into ensemble algorithms to define the oversampling online bagging (OOB) and undersampling online bagging (UOB) techniques for binary classification. The work extends bagging ensembles following a class-based ensemble approach to dynamically change

the learning rate by maintaining a base learner for each class and updating the base learners with new data to deal with binary class imbalance. Resampling will be triggered to either increase the chance of training minority-class examples (in OOB) or reduce the chance of training majority-class examples (in UOB).

Online queue-based resampling [38] has also been proposed as a method for binary classification. The main idea of this algorithm is to selectively include a subset of the positive and negative examples in the training set that thus far have appeared in the stream. The examples are selected by maintaining, at any given time  $t$ , separate queues based on class labels received from data of equal lengths  $L \in \mathbb{Z}^+$ ,  $q_n^t = \{(x_i, y_i)\}_{i=1}^L$  and  $q_p^t = \{(x_i, y_i)\}_{i=1}^L$  that contain the negatives as majority examples and the positives as minority examples, respectively. Once the queues are filled, the classifier is incrementally updated after combining the two queues into one training set [38]. The algorithm employs an interleaved test-then-train evaluation [8] in which each example is used to test the model before the example is appended to the queues for training, thus implementing a sliding window method. Our work extends the notion of queues as presented in [38] to the multi-class scenario by incorporating explicit concept drift detection during learning.

## 2.2 Multi-class imbalanced learning

Multi-class classification problems are often considered more challenging than their binary counterparts, because multiple classes can increase the data complexity and aggravate the imbalanced distribution [1]. Current approaches in an online setting are primarily based on binary decomposition techniques, algorithmic-level modifications using misclassification costs, and resampling methods [1, 51]. Binary decomposition algorithms typically combine binarization techniques that transform the original multi-class data into binary subsets [30]. One of the most commonly used binarization strategies is the one-versus-one (OVO) [20] decomposition where it first selects a subset from the original data that only contains the instances for each pair of classes and proceeds to train a binary classifier for each pair. In contrast, in the one-versus-all (OVA) approach [20], a multi-class data set is decomposed into several binary class problems and subsequently train single classifiers for each class by considering a single class versus a combination of all the remaining classes. For instance, [28] uses adaptive online one-class Support Vector Machines to monitor changes in minority classes over time. Further, [17] propose integrating one-class classification with ensembles using over-sampling and instance selection techniques to balance the class distribution of incoming data batches, which are then used to induce classifier ensembles. A disadvantage of binarization

techniques is that the interactions between multiple classes cannot be considered simultaneously.

Algorithmic-level approaches adapt the training process to enhance the classifiers' ability to deal with skewed distributions. These methods are often specific to particular learning models, making them more specialized but less flexible than data-level approaches. One of the most common algorithm modifications for addressing class imbalance combines Hoeffding Trees with the Hellinger splitting criteria for imbalanced domains. In GHVFDT [36], Hoeffding Trees are used to construct a decision tree that can handle data streams, while the Hellinger distance is used as a splitting criterion that is less sensitive to class imbalance. Further, [31] introduced an approach that modifies predictions made by a base classifier to address imbalanced data streams. This algorithm aims to map prior probabilities in the statistics of assigned classes.

A few research studies considered multi-class online learning from evolving streams, focusing on resampling techniques, using resampling and online ensemble learning together [1, 29]. Online ensembles learn each incoming training example separately, and component classifiers are constructed from corresponding instances. These approaches use this method to learn the data stream in one pass. Several approaches extend the well-known online bagging (OB) algorithm, as introduced by [42]. The OB algorithm modifies the original batch-based bagging method that samples with replacement in an online setting by calculating the value of  $k$  based on the Poisson distribution. New instances are classified by a majority voting of the  $N$  base model. Specifically, in a recent study [52], the authors proposed two ensemble learning methods for multi-class online learning. These two algorithms, multi-class oversampling-based online bagging (MOOB) and multi-class under-sampling-based online bagging (MUOB), use resampling to overcome class imbalance with the framework of OB, as introduced above [42]. These algorithms can process multi-classes directly without using class decomposition. However, the performance is based on underlying assumptions, such as that sampling only based on the size of classes is efficient and does not introduce bias. Some approaches employ feature space modification to define what feature space input is used by base classifiers. In [14] diverse feature subspaces of random sizes are used to improve the ensemble's performance. The Kappa Updated Ensemble (KUE) method employ a combination of base classifiers that are updated dynamically based on a  $\kappa$  statistic, which measures the agreement between the base classifiers on a sliding window of recent data. KUE also incorporates an instance weighting scheme that prioritizes recent data over older data. Similarly, [15] introduce the Robust Online Self-Adjusting Ensemble (ROSE), an online ensemble-based algorithm that combines classifiers trained on variable-sized random subsets of features. The ROSE

algorithm incorporates undersampling of the majority classes and employs so-called background classifiers initiated once drift is detected. In essence, ROSE is based on self-adjusting bagging and balances the training sets by monitoring the product of the accuracy and  $\kappa$  statistic. Specifically, each window provides a background classifier with a balanced data set of recent instances at the start of model construction. Background classifiers are dynamically added to the ensemble once their performance surpasses a threshold.

The OMCQ framework maintains multiple queues for each class in the multi-class learning setting [46]. This algorithm learns directly from the original data without resampling and incorporates a drift detection mechanism that can adapt to class sizes. While previous results were promising, the OMCQ approach may be extended to use dynamic sampling, as will be discussed in Section 3. The authors in [47] introduced a new method for dealing with multi-class imbalanced data called improved online ensembles (IOE) for semi-supervised learning. In this technique, instances from the minority and majority classes are sampled based on the classifier's performance, measured in terms of the recall of each class. Classes with lower-than-average recalls are oversampled, while classes with higher recalls are undersampled. The sampling in the IOE algorithm is controlled by setting the rate parameter of the Poisson distribution based on the recall score of classes, which controls the number of times each instance is used for training each base learner in the ensemble learning model. At each time step, the ensemble model is employed so that the  $N$  individual classifiers are trained by oversampled or undersampled instances.

In cases where ensembles have limited access to labels, a set of algorithms is also available [48, 55]. One such approach is CALMID [32], which is a robust framework that deals with limited label access, concept drift, and class imbalance by dynamically inducing new base classifiers and weighting the most relevant instances. The method uses a variable threshold uncertainty strategy based on an asymmetric margin threshold matrix to address the problem of a given class being a majority to a given subset of classes while also being a minority to others. A novel sample weight formula is designed to consider the class imbalance ratio of the sample's category and the prediction difficulty.

Next, we introduce our DynaQ framework.

### 3 DynaQ framework

Our DynaQ framework maintains a queue for each of our multiple classes. Initially, all queues will be empty. As instances arrive, they are added to the appropriate queue per their true label. Our sampling and training processes

commence when the first queue has been filled. Figure 1 illustrates how our contributions fit together and operate in one iteration of an interleaved test-then-train loop. For each arriving instance, minority classes are oversampled to balance the different classes. As noted in Section 2, the sampling process is based on rate parameters of the classes, where classes with rates lower than the average are oversampled, while classes with higher rates use the original training instances. That is, our algorithm oversamples minority instances, while majority classes are not undersampled.

The online learning phase of DynaQ thus incorporates four processes: evaluation, dynamic class balancing, ensemble learning, and concept drift detection. Our algorithm creates a queue space in the class balancing module to separate the instances from each class as they arrive within the stream. Subsequently, if the queue for a class is not full or the parameter rate for that class is below a threshold, the queue is updated with an oversampled instance; otherwise, we use the instance only once by inserting it in the related queue. The concept drift detector captures changes in the data distributions by adapting the idea of the drift detection method (DDM) [50] and subsequently updating the instances in the queues. The online ensemble component updates a single learner per iteration of the interleaved test-then-train loop in a cyclical manner, meaning that each classifier in the ensemble is trained only once every  $N$  loop iterations;  $N$  is the number of classifiers in an ensemble. That is, a different learner from the ensemble is trained at the next loop iteration, with the sliding batch including the new instance and overlapping by the previous one.

Specifically, the evaluator is used to predict the class label of arriving instances, using soft voting, and to update the evaluation metrics. The concept drift detector captures changes in the data distributions by adapting the DDM algorithm [50] idea and subsequently updating the instances in the queues. In the class balancing module, our algorithm creates a queue space to separate the instances from each class as they arrive within the stream. Subsequently, if the queue for a class is not full or the rate parameter for that class is below a threshold, the queue is updated with the oversampled instance. Otherwise, as indicated above, we use the instance only once by inserting it in the related queue. During online learning, a single learner is trained during a single iteration of the interleaved test-then-train loop in a cyclical manner. A soft voting-based ensemble of classifiers is used during training to incrementally update the model using sliding batches. This implies that each classifier in the ensemble is trained only once every  $N$  loop iteration, where  $N$  is the number of classifiers in the ensemble. A different learner from the ensemble is trained at the next loop iteration, where the sliding batch includes one new instance. Next, we present our DynaQ algorithm's details, as shown in Algorithm 1.

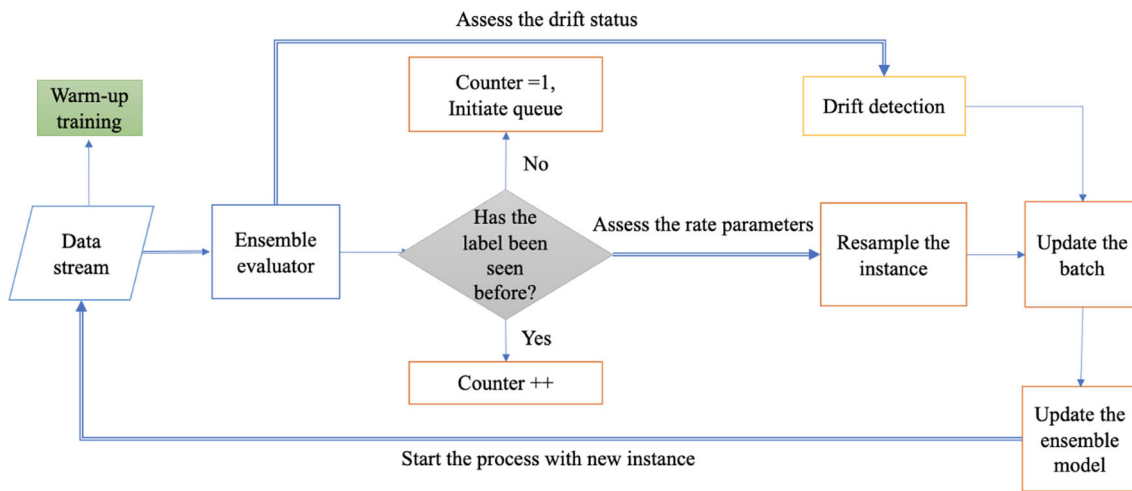


Fig. 1 High-level overview of DynaQ methodology

### 3.1 Online queue construction

In an offline supervised learning setting, it follows that a data set  $D$  is available with input signals  $x_i$  and output  $y_i$ . The task is to infer a model  $M \approx p(y | x)$  from such data. In contrast, online learning refers to approaches when the full data set  $D$  is unavailable during learning. Here, examples arrive over time, and the task is to infer a reliable model  $M_t$  at time step  $t$  based on the newly arriving example  $(x_t, y_t)$  and the previous model  $M_{t-1}$ .

Incremental online learning is a sub-area of online learning that is additionally bounded by memory resources and the capability of continuous learning with limited data compared to offline learning. In the literature, approaches to incremental learning can generally be categorized as batch incremental and instance incremental [44]. As the name suggests, batch learning methods employ batches of data to form hypotheses about the data. At every time step  $t$ , this form of training collects the  $k$  newest instances to form a batch. When a batch of data  $D_t$  is filled, a model  $m_t$  is learned [34]. This process continues, batch by batch. In our work, following [38], we consider a sequence of streaming data  $f = \{(x_1, y_1), \dots, (x_t, y_t)\} \in R^n \times \{1, \dots, S\}$ , where  $f$  is the data dimension, and  $S$  is the total number of classes. The key idea is to keep a fixed number of examples (queue

size denoted by  $L$ ) for each class in a stream to combine the training set. In other words, each arriving sample  $(x_t, y_t)$  at any given time  $t$  will be stored in a separate queue of equal length  $q_{C_s}^t = L$ , where  $c_s$  is the class label received with the data. Together, the queues form a sliding batch  $B_t$ . This method considers a given stream of data  $x_1, x_2, \dots, x_t$  and learns from a sequence of batches  $b_1, b_2, \dots, b_t$ , where the batches are updated as instances arrive and update the model.

Figure 2 illustrates how  $Queue_L$  works when  $q = 3$ . The upper part shows the examples that arrive at each time step; for example,  $z^0$  and  $z^4$  arrive at  $t = 0$  and  $t = 4$ , respectively. Assume that the data stream contains three classes  $Y = \{c_1, c_2, c_3\}$  and that all instances have their own queues. The queues are of equal length  $L \in \mathbb{Z}^+$ ,  $q_{C_1}^t = L$ ,  $q_{C_2}^t = L$ , and  $q_{C_3}^t = L$  and contain the samples of class  $c_1$ , class  $c_2$ , and class  $c_3$ , respectively. After instances have been separated based on their labels, the arriving samples for class  $c_s$  are placed at the front of the  $queue_{c_s}$ . When the queues fill, we combine the full queues in the batch by adding to the training set and commence online learning. Here, the sliding batch explicitly employs a forgetting mechanism, where the oldest instance will be removed from the head of the related queue.

Fig. 2 Example of  $Queue_3$  resampling (adapted from [38])

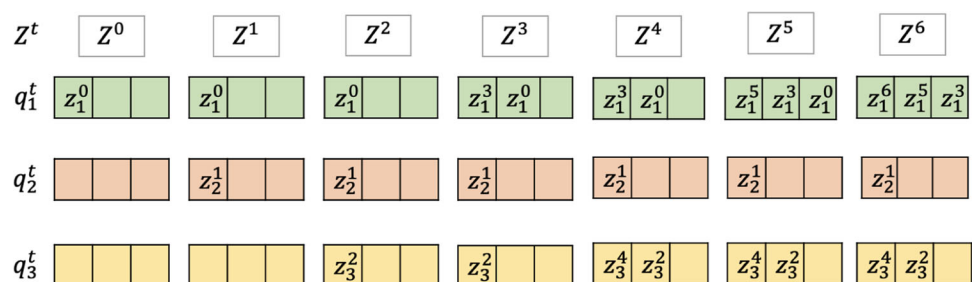


Figure 3 shows that  $B_t$  is a batch of data, including the full queues of current classes, defined as the number of instances used to update the model. We construct an initial model with warm-up instances from the data stream as an initial training step. Whenever the first queue is full, we proceed to initiate our rebalancing process until we have created a full queue for each class. We include those queues in our batch. Next, the model construction process is initiated, and this batch-incremental process continues until the end of the stream. For each arriving instance  $(x_t, y_t)$  at time  $t$ , the oldest sample from the queue to which  $y_t$  belongs is removed, and the recent sample is added to batch  $B_t$ . Whenever the first queues are full, they are used as our first batch, and we then proceed to update the model. Meanwhile, at each point in time, only one learner is updated with the batch that includes the new resampled instance in a circular order. The learner will use batch  $B_t$  to update its model; the training process utilizes a balanced set consisting of the most recent data. The algorithm waits until it has enough instances from the classes, including the current minority classes, before updating its model. It follows that the sizes of the individual queues are highly domain-dependent; the size of the queue is set by inspection.

### 3.2 Queue-based sampling

As noted above, our DynaQ methodology employs queue-based sampling of original instances of each class to dynamically construct models against all classes, using an ensemble against a sliding batch. Next, we will explain the process of sampling each instance in the queues. Recall that minority classes suffer from not having enough data to present against majority classes. Following [47], we oversample each class’s recent instances based on the class’s rate parameter while maintaining the majority instances without any form of undersampling. This is done dynamically during learning as the stream evolves. In this way, the learner has access to newer samples and concepts, and we can balance the number of instances for all classes. In addition, the oversampling of minority classes implies that the associated queues will fill faster. We employ a sliding batch of  $S$  queues, where  $S$  is the number of current classes, and  $N$  learners are updated one at a time in a periodic order with each arriving instance. The oversampling rate of our DynaQ algorithm is implemented considering four different metrics, as discussed below.

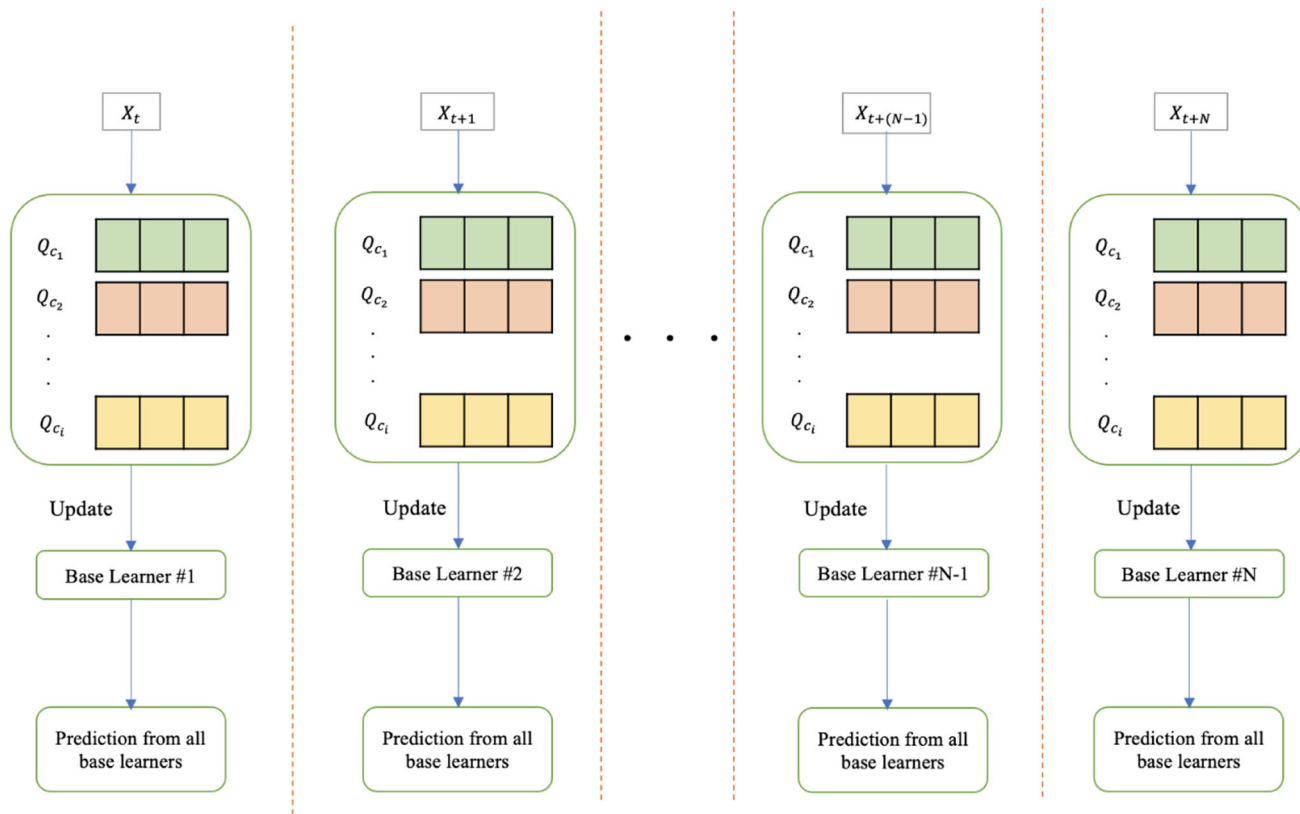


Fig. 3 Illustration of batch-instance incremental learning process

### 3.2.1 Recall-based Sampling

In our first variant, the rate parameter is set based on the recalls of the classes [47]. That is, we employ the recall value to maintain a balance in terms of the samples from the majority and minority classes that we maintain, where we define:

$$recall = \frac{TP}{TP + FN} \tag{1}$$

and where true positive ( $TP$ ) refers to the instances from the actual class that are correctly classified, while false negative ( $FN$ ) denotes those instances with incorrect predictions. As such, recall measures the ratio of correctly classified instances from the minority class (true positive rate). When employing the recall scores for the multi-class data stream, we may assess how many of the examples from each class are correctly classified. A model with high recall on each class successfully predicts true labels in the data [52]. If the absolute difference between the recall score for a class and the mean recall is higher than the pre-defined threshold, then the examples for this class will be oversampled using recalls, as follows.

$$k = Poisson(r_{avg\_excluding\_c}/r_c) \tag{2}$$

Where,  $r_{avg\_excluding\_c}$  is the average recall for the classes excluding class  $c$  and  $r_c$  is the recall for class  $c$ , which is calculated based on prequential evaluation, where each instance will be used to test the model before it is used for training [56], and from this the recall can be incrementally updated. If the recall for a class is lower than the average recall, then the class will be oversampled. We set the max-value of  $k$  equal to the defined queue size, which is determined through inspection.

### 3.2.2 F1-Score-based Sampling

The F-measure [21] refers to the harmonic mean of two metrics, recall, and precision. The F-measure may be weighted depending on the value assigned to  $\alpha$ . We used a balanced value, referred to as the F1-score, by setting  $\alpha = 1$ , which implies that precision and recall are assumed to carry equal weights in the metric.

$$F - measure = \frac{(1 + \alpha^2 \times recall \times precision)}{\alpha^2 \times recall + precision} \tag{3}$$

In our second sampling approach, we utilize the F1-score of the classes as a measure to control the number of oversampled instances. That is, if the absolute difference between the F1-score for a class and the mean F1-score is higher than

the pre-defined threshold, then the samples for this class will be oversampled using the F1-score.

$$k = Poisson(F1 - score_{avg\_excluding\_c}/F1 - score_c) \tag{4}$$

Here, the F1-score refers to the score of each class calculated prequentially and  $F1 - score_{avg\_excluding\_c}$  denotes the average F1-score of all classes excluding class  $c$ . If the F1-score of one class is lower than the average, it will be oversampled  $k$  times.

### 3.2.3 $\kappa_m$ -based Sampling

In the third case, we utilize the  $\kappa_m$  measure to oversample. Bifet and Morales proposed the  $\kappa_m$  statistic for online learning in [11]; where they confirmed that this measure has advantages over accuracy and the original  $\kappa$  statistic [8]. The main motivation for using the  $\kappa_m$  statistic is when data streams are evolving, and classes are imbalanced, where we have:

$$\kappa_m = \frac{p_0 - p_m}{1 - p_m} \tag{5}$$

In (5), quantity  $p_0$  refers to the current algorithm  $X$ 's prequential accuracy, while  $p_m$  is the prequential accuracy of a majority-class classifier, a baseline learner that predicts the label that occurred most frequently up to now [8]. If classifier  $X$  is always correct, we conclude  $\kappa_m = 1$ . If its predictions are correct as often as those of a majority-class classifier, then  $\kappa_m = 0$ . Since the  $\kappa_m$  metric measures the performance agreement between the majority class classifier and classifier  $X$ , we cannot calculate it for each class. However, it is a measure that sensitively detects changes in the class distribution while automatically compensating for such changes. It may thus be used to recognize where classifier  $X$  is underperforming the baseline majority class learner. In this way, we will be able to assess when classifier  $X$  would benefit from oversampling the minority classes. Specifically, the ratio of the mean of the  $\kappa_m$  value divided by the last calculated  $\kappa_m$  value is considered as a rate parameter. If the result surpassed a threshold, the class will be oversampled  $M$  times following:

$$k = Poisson(MajorClassSize/NewInstanceClassSize) \tag{6}$$

In this equation,  $MajorClassSize$  refers to the number of instances in the class with the most instances seen so far, and we update each class size by the class label of each arriving instance.

### 3.2.4 Euclidian distance-based sampling

The last oversampling version monitors the ratio of the majority class to the class label of newly arrived instances. This parameter as per equation 6, defines the number of instances we need to oversample based on a distance criterion [39]. In this case, if  $k$  is greater than a defined threshold, we resample the  $k$  instances that are **most similar** to the current one. The Euclidean distance similarity score is used to determine the  $k$  most similar instances as:

$$EuclideanDistance(x_c, q_c) = \sqrt{(x_c - q_c)^2} \quad (7)$$

Here,  $x_c$  refers to the most recently arrived instance with the class label  $c$ , and  $q_c$  is a vector of instances inside the queue of class  $c$ . After we sort the results based on distances, the top  $k$  instances will be oversampled and inserted into the  $q_c$ . In this way, the  $q_c$  contains instances that are most similar to the  $x_c$  while disregarding the less similar ones.

### 3.3 Ensemble learning

Recall that we utilize a sliding ensemble approach, where the base learners update their models against different batches, corresponding to sliding batches, following [22]. In our algorithm, each of the base classifiers is trained independently [22]. We extended OB [42] so that, instead of training each instance  $k$  times from a Poisson distribution, we employ  $k$  to oversample the relevant queues. Figure 4 illustrates how we use sliding batches to update base learners. That is, for each instance in the data stream, one base learner in  $N$  is updated by the sliding batch. If we consider  $N$  learners as  $N = \{p_1, p_2, \dots, p_{N-1}, p_N\}$ , where  $p_1$  refers to the first

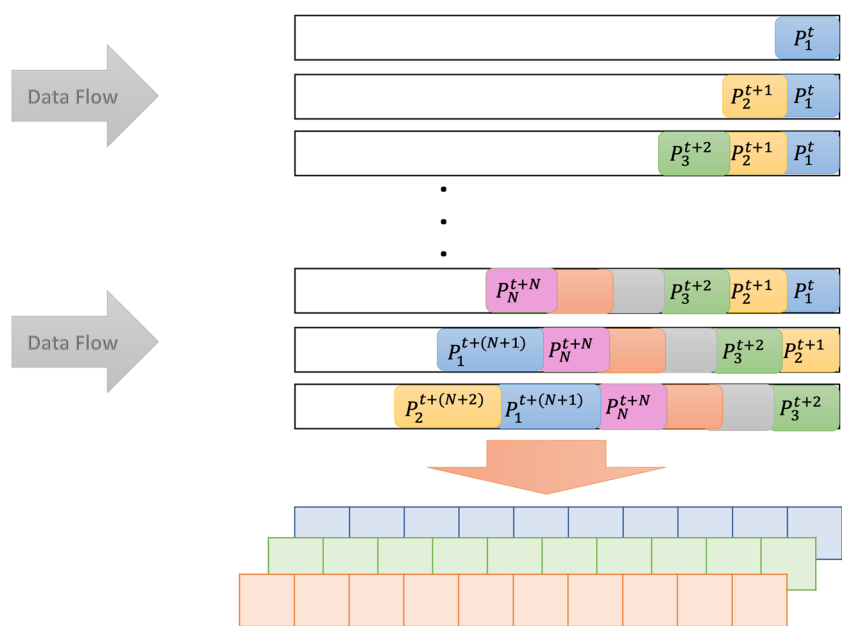
learner,  $p_1$  updates the batch at time  $t$ ,  $p_2$  updates with the batch at time  $t + 1$  and the  $p_N$  updates with the batch at  $t + N$ . Subsequently, the rotation restarts from  $p_1$ . Note that the prediction at each time step is made over all the base learners. We employed a soft voting process to determine the ensemble prediction [27]. For each arriving instance, soft voting requires each of the learners in the ensemble to produce a confidence score (within the range  $[0, 1]$ ) for their prediction for each class value or to output the probabilities that an instance belongs to a given class label. Consequently, a simple, soft voting classifier without weighting factors, given an instance  $x_t$ , calculates the average probability for each class label over the predictions of all classifiers and determines the most probable class  $C$  as in (8) [27]:

$$\hat{y}_t = \arg \min_C \sum_{j=1}^{|N|} P_j(= C || x_t), y_t \in Y, t \in \{1, 2, \dots, t'\} \quad (8)$$

The class with the largest average probability is exported as the winner through this process, where  $y_t \in Y$ , and the notation of  $N$  depicts the number of the combined classifiers.

Our ensemble is designed so that the training set of the  $N$  individual classifiers proceeds out of step, using a sliding batch for instance selection. As shown in Fig. 4, at every time step, we append a new updated batch to the ensemble and train a single classifier  $p_n$  on that batch. For the next  $N - 1$  iterations, we train the remaining  $N - 1$  classifiers, and so on. From the ensemble point of view, it looks like we are using a sliding window to train with the differences of  $N$  time steps. However, from the point of view of each base classifier in the ensemble, we are employing sliding batches to train. Figure 4 illustrates this process for  $N$  classifiers in the ensemble where

Fig. 4 Ensemble learning of sliding batch [22]





the difference for each is a time step, and each classifier will learn based on a batch of the same color. Starting with  $p_1$  with the blue batch, the next classifier is  $p_2$  with the orange batch of time  $t + 1$ . The next rotation will start after  $N + 1$  steps, as shown with the next blue batch training  $p_1$  at time  $t + (N + 1)$ . Intuitively, this method ensures diversity in terms of the instances used by the individual classifiers when casting their votes.

---

**Algorithm 1** DynaQ.

---

```

1: Define rate_parameter(Recall, F1score,  $\kappa_m$ , EuclideanDistance)
2: while stream.has_more_instances() at each time step t do
3:    $x_t^i, y_t^i = \text{get.next\_instance}()$ ;
4:    $y_t^i\text{\_predict} = \text{Ensemble.Predict}(x_t^i)$ ; #Test-then-train evaluation
5:    $W = \text{Update\_}W_{C_i}$ ;
6:    $\text{Drift\_list}_{C_i} = []$ ;
7:    $\alpha_{C_i}^t = \text{TestForDrift}(x_t^i, y_t^i)$ ; #Test for concept drifts
8:   if  $y^t$  in previously seen class  $C_i$  then #Add to queue
9:     Increment  $\text{Counter}_{C_i}$ ;
10:     $Q_{C_i}^t = Q_{C_i}^{(t-1)}.append(i)$ ;
11:    Update  $(\alpha_{C_i}^t)$ ;
12:   else
13:      $\text{Counter}_{C_i} = 1$ ;
14:     Initialize  $Q_{C_i}^t, Q_{C_i}^t.append(i)$ ;
15:     Update  $(\alpha_{C_i}^t)$ ;
16:   end if
17:   if rate_parameter_ratio > threshold then
18:      $L[C_i] = \text{rate\_parameter\_oversampling}$ ;
19:   end if
20:    $k = \text{Poisson}(L[C_i])$ ;
21:   if  $k > \text{queue\_size}$  then,  $k = \text{queue\_size}$ ;
22:   end if
23:   for  $k$  times do
24:      $Q_{C_i}^t.append(x_t^i)$ ;
25:   end for
26:   if  $(\alpha_{C_i}^t \geq w_{C_i}^t)$  is TRUE then #Warning, Start Drift list
27:     Add instance  $i$  to  $\text{Drift\_list}_{(C_i)}$ ;
28:   end if
29:   if  $(\alpha_{C_i}^t \geq d_{C_i}^t)$  is TRUE then #Drift detected
30:     Update  $Q_{C_i} = \text{Drift\_list}_{C_i}$ ;
31:   end if
32:    $Q_{C_i} = Q_{C_i}(x_h, y_h)$ ;
33:    $B_t = \bigcup_{i=2}^n Q_{C_i}$ ;
34:    $\text{Base\_Learner}_p = t\%N$ ; #Creates a rotatory updates of base learners
35:    $\text{Classifier}(\text{Base\_Learner}_p).\text{Incremental.Update}(B_t)$ ;
36: end while
37: Return  $G\_Mean, F\_Measure, \kappa, \text{Model}$ 

```

---

### 3.4 Concept drift detection

Recall that we also included a class-based concept drift detector to handle evolving streams. To this end, we adapted the idea of the DDM drift detection algorithm in our framework [50]. The main task of a drift detector is to prompt the learner to update the model after drift occurs. The number of misclassified instances corresponding to each class

is used as a drift indicator based on the results so far. Following [50], we employ two counters for each class, where  $w_i$  denotes a warning level, and  $d_i$  denotes the drift detection threshold. That is, we continuously update  $w_i$  and  $d_i$ , and if the number of misclassified instances reaches  $d_i$ , then a drift is detected. Subsequently, a new model is induced using the examples stored between  $w_i$  and  $d_i$ . Practically, this process aids in removing outdated samples and updates the queue with new instances. Our drift detector process is initiated once an instance is misclassified, then continues until it reaches the specified proportion of the queue (denoted by  $L/n$ ). The rationale behind this approach is to find a trade-off between the ability of the learner to adapt faster while not testing for drift too often to limit the overhead associated with detection. We aim to maintain only the optimal “small subset” of data necessary to accurately flag for drift. Intuitively, if  $L = 1$ , then the process corresponds to testing for drift as every instance arrives; that is,  $n = 1$ . Figure 5 shows our results against the Gas Sensor [49] and LED data stream [10], two of the repositories we used in our experiments where  $n$  was set to 2 by inspection. The reader will notice that, as expected, the drift detection threshold has a considerable influence on the predictive performance. In this setting, once a misclassification occurs, we signal a warning for potential drift and start to collect all instances from this point in time into the drift detector queues. Next, we test for drift when we reach  $(L/n)$  instances and proceed accordingly. We either detect a drift (reset the learner) or continue monitoring. If no drift has been detected, but the warning level remains, we proceed to collect and test with the next  $(L/n)$  instances. This process continues until the set of examples is equal to our queue size  $L$ . When a drift is detected, the learner is reset, and a new model is learned using a training set consisting of all the examples in the drift detection queues maintained since the warning was triggered. It follows that the values of  $n$  and  $L$  are domain-dependent and should be carefully selected to ensure the accuracy and efficiency of the drift detector. As shown in Fig. 5, the response to concept drift in  $\alpha = (1, L)$  is later than the other two, which decreased the model performance. However, as should be expected, the reaction to the threshold values is different for both data streams. The Gas Sensor data experience more fluctuation and decrement during the stream; the LED stream is more tolerant of late drift detection for  $\alpha = (1, L)$ . In summary, results of  $\alpha = (1, 1/2L)$  outperform the other thresholds for both streams.

In summary, in our DynaQ algorithm, the rate parameter is calculated when a new instance arrives. Subsequently, for minority classes, the new instance is resampled  $k$  times and appended to the appropriate queue. As explained above, we use each instance to test the model, then insert the instance into the appropriate queue. The set of queues form a batch and each batch is used by a single learner to update the model.

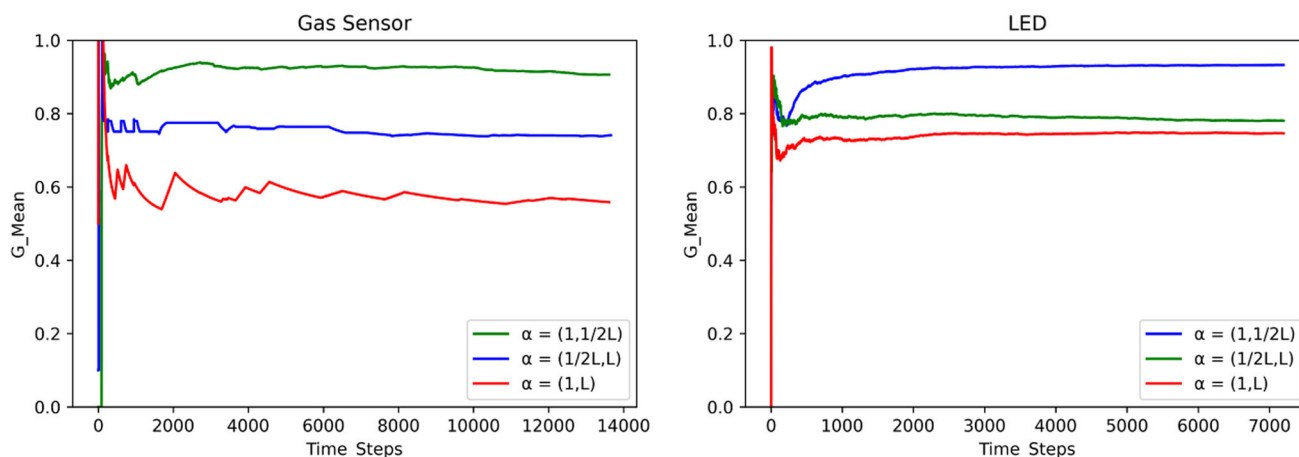


Fig. 5 G-mean value results with different concept drift thresholds against the Gas Sensor and LED streams

Ensemble learning involves sliding windows, and soft voting. Next, we discuss our experimental evaluation.

### 4 Experimental evaluation

All experiments were conducted on a MacBook Pro with a Dual-Core Intel Core i5 processor, CPU @ 3.1 GHz processor, 8.0 GB RAM on the Mac Catalina Operating System (OS) and the Alliance Canada Cloud with 10 Core CPUs [3]. Our code was implemented using the Scikit-Learn [43] and Scikit-Multiflow [40] packages in Python version 3.8.2. Our competitor methods run on the MOA [10] library and we used MOA to generate synthetic streams. The framework’s implementation and all the code for the experiments will be made available on GitHub upon publication. The no-change and majority-class classifiers were used as our baselines. The no-change classifier assumes that the class label of instance  $x_i$  would be the same as the last-seen instance  $x_{i-1}$ , while the majority-class learner assigns the class seen most often so far to a new instance [40]. Additionally, we employed three baseline classifiers, namely, Hoeffding Adaptive Tree (HAT) [9], Hoeffding Tree (HT) [18], and the self-adjusting memory (SAM) model for the K-nearest neighbour (KNN), denoted by SAMKNN [33], during our ensemble learning. HTs are incremental decision trees for data stream classification that use Hoeffding’s bound to commence online learning. HAT is an extension of HT that adaptively learns from data streams that change over time without needing a fixed-size sliding window. SAMKNN is an online implementation of KNN, and we set  $k = 7$  by inspection. Following [47], we set the number of base learners in our ensemble to 10.

The estimation technique we use is prequential evaluation, which consists of executing a loop infinitely, where the ensemble first predicts labels for new data (without its label), then updates its model by that data with the correct label [56].

The performance measures we used are the F-measure, geometric mean (G-mean), and  $\kappa_m$  statistic. As mentioned, the F-measure [21] is macro-averaged over the sum of F1-scores over all classes, which assigns equal weights to the existing classes. Additionally, we employed the G-mean [50] value which is the geometric mean of the recall rates of majority and minority classes in the imbalanced data set. The calculation method is as shown in (9).

$$G - mean = \sqrt{\frac{TP}{TP + TN} \times \frac{TN}{FP + TN}} \tag{9}$$

The G-mean value is higher only when the classification accuracies of the majority sample and the minority sample are high; therefore, the G-mean value can accurately the classification effect of unbalanced data sets. In addition, we utilize the previously introduced  $\kappa_m$  metric, to address the effect of the performance agreement between the majority class classifier and classifier X [11].

We also compare DynaQ with six state-of-the-art online multi-class learning methods, namely, OMCQ, IOE, ROSE, KUE, MOOB and MUOB. We use inspection and grid-searching for hyper-parameter tuning to determine the optimal parameters for all methods. However, if a particular algorithm specifies certain parameters, we respect that and use them accordingly. As we introduced in Section 2.2, ROSE and KUE are driven by the  $\kappa$  metric. While KUE is a chunk-based general-purpose ensemble for drifting data streams [14], ROSE is an online ensemble that works with imbalanced data streams with dynamic imbalance ratio and concept drift, offering several features designed specifically to deal with these challenges [15]. All ensembles are evaluated using HTs as base learners with the same parameter settings of 10 base classifiers. These two methods employ a sliding window size of 1000 instances. Recall that OMCQ uses a queue-based sampling strategy to keep each class separated in the queues

to maintain a balanced training set. The method employs a class-based DDM with a queue-based recovery process. Following [46], the queue sizes were set by inspection. As noted in Section 2.2, Vafaie et al. (2019) introduced an IOE algorithm for handling multi-class imbalanced data. The approach samples with replacement and incorporates DDM-OCI [50]. DDM-OCI tracks the recall rates on the minority classes to actively locate concept drifts for imbalanced data streams. A significant drop in the recall suggests a drift in this class. After drift is detected, the model will be reset and trained based on the data received between the drift warning and drift detection. Following [47], we set the forgetting factor in recall rates to be 0.9 and the threshold of the absolute difference between the class recall and average recall equal to 0.05. Additionally, as mentioned in Section 2.2, [52] introduced the MOOB and MUOB algorithms that oversample or undersample classes based on the probabilities of instances belonging to a class. That is, in MOOB, oversampling is used to increase the possibility of learning minority-class examples based on the occurrence probability of examples belonging to each class. Meanwhile, in MUOB, undersampling is used to reduce the chance of learning majority-class examples. We incorporated the DDM-OCI [50] method with MOOB and MUOB in case of concept drift to conduct a fair comparison. We conducted four sets of experiments to assess our DynaQ algorithm. First, we studied the impact of the four different rate parameters used in order to conduct minority class oversampling. Second, we considered the impact of queue size on learning. Third, we explored the performance of DynaQ when utilizing different base classifiers in our ensemble. Fourth, we considered the impact of concept drift detection on DynaQ. Finally, we contrast our DynaQ with the state of the art.

#### 4.1 Data streams

Our experimental study was based on the following multi-class data sets depicted in Table 1: historical weather data

obtained from Open Data Canada [24], the Shuttle data set from the KEEL repository [2], the LED data stream [10], the radial basis function (RBF) stream [10], the Gas Sensor stream [49], the Human Activity Recognition (HAR) stream [16], the Coverttype data stream [12] and the Intel Berkley Research Lab Sensor data stream [37]. The Weather repository contains data from probes located across Canada to detect adverse weather with natural drifts. The Shuttle stream considers three classes and is used to predict when an auto-landing would be preferable to the manual control landing of a spacecraft. The LED data set comprises seven Boolean attributes and ten labels; the goal was to predict the digit displayed on a seven-segment LED display, where each attribute has a 10% noise level. We used a version of LED available through Scikit-Multiflow that includes gradual concept drifts in the stream by simply changing the attribute positions. The RBF generates a fixed number of random centroids, where each center has a random position, a single standard division (SD), a class label, and a weight. The generated RBF data sets have ten numerical attributes and 50 centers with four classes, and a change speed of 0.89 was chosen for the gradual drift in the data. To assess the impact of concept drifts on imbalanced streams, we created two RBF streams with comparable imbalance ratios but distinct concept drift patterns. The Gas Sensor stream contains 13,610 measurements from 16 chemical sensors utilized in simulations for drift compensation in a discrimination task of six gases at various concentration levels. The HAR data set contains uncalibrated accelerometer data from 15 participants performing seven activities. We combined the activity of three participants to create drift in the stream. Coverttype is a benchmark data set for evaluating stream classifiers that originates from the UCI repository and contains cartographic attributes for predicting forest cover type. This data set represents a forest cover type for  $30 \times 30$ m cells, where each cover type is represented by one of the seven classes. Concept drift may appear in this domain due to weather and climate change. The Intel lab data is collected from 54 sensors deployed in the

**Table 1** Data streams and their properties

Data set	Size	Number of class	Class imbalance ratio	Drift type
Weather	29,375	4	1:2.5:1.5:13	Abrupt
Shuttle	2167	3	1:5:13	No drift
LED	7205	4	1:1.5:2.7:5.7	Gradual and Noise
RBF1	50,000	4	1:1:1:2	Gradual
RBF2	40,000	5	1:2:4:10:20	Gradual
RBF3	40,000	5	1:2:4:10:20	Abrupt
Gas Sensor	13,610	4	1:1:1:4.4	Abrupt
HAT	35,300	4	1:1:1.7:6.3	Gradual
Coverttype	42,000	7	1:1:1:1:1:5:13	Abrupt
Intel Sensor	50000	15	1,1,1,1,1,2,2,3,3,3,3,6,6,6,10,10	Abrupt

Intel Berkeley Research lab between February 28th and April 5th, 2004. Mica2Dot sensors with weatherboards collected timestamped topology information and humidity, temperature, light, and voltage values once every 31 seconds. Data was collected using the TinyDB in-network query processing system, built on the TinyOS platform. We used the data instances from 15 sensors to produce an imbalanced data stream.

## 4.2 Experimental results

In our first set of experiments, we investigate using the four different DynaQ oversampling metrics introduced in Section 3.2, as depicted in Table 2. In this set of experiments, we employ the HT classifier and report the G-mean, F-measure, and  $\kappa_m$  evaluation metrics. Table 2, shows that the DynaQ variant using recall produced the highest values for four of seven data streams. When considering the individual metrics, the reader will notice that the results for DynaQ with F1-score are highest for the Weather and RBF streams. In the case of the CoverType data set, the variant of DynaQ- $\kappa_m$  using G-mean and  $\kappa_m$  metrics yields the highest results.

Future, we consider the statistical significance of the results using the Nemenyi post-hoc test with  $\alpha = 0.05$ , as depicted in Fig. 6. Figure 6 illustrates that oversampling using recall and F1-score, results in a similar performance, with the recall-based sampling ranking first. This result indicates that paying close attention to the true positive rates clearly benefits learning. As a result, we utilize the DynaQ variant with recall-based sampling in our subsequent experiments.

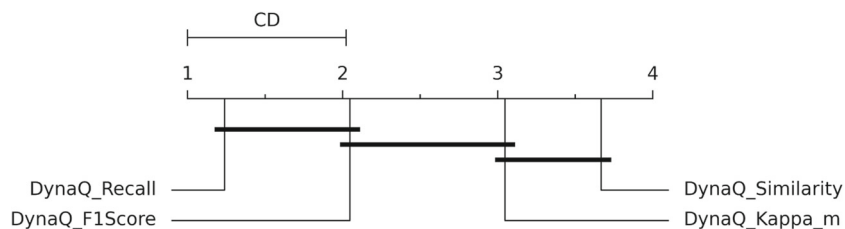
Second, we investigated the effect of queue size on our DynaQ algorithm to assess how the size of  $L$  affects the performance of queue-based learning. Figure 7 depicts the proposed method's behavior on different queue sizes  $L \in \{1, 10, 20, 30, 50\}$ . As expected, the figure shows that the smaller the queue length, the faster the learning speed, although the results might differ with longer queues. For most data sets, this number is highly domain-dependent and should be set according to the characteristics of each data stream. In online learning, there is an obvious interplay between accuracy and learning time. Our results indicate that, for our experiments, a queue size of 10 resulted in a good trade-off between accuracy and speed for the Shuttle, Weather, and Gas Sensor data sets. A queue size of 20 produced good results

**Table 2** Evaluation of different versions of DynaQ against data streams

Data	Metric	Hoefding Tree			
		DynaQ_Recall	DynaQ_F1Score	DynaQ- $\kappa_m$	DynaQ_Similarity
Shuttle	G-mean	<b>0.971</b>	0.958	0.871	0.828
	F-measure	<b>0.952</b>	0.916	0.849	0.805
	$\kappa_m$	<b>0.837</b>	0.742	0.783	0.582
Weather	G-mean	0.862	<b>0.863</b>	0.750	0.711
	F-measure	0.851	<b>0.861</b>	0.726	0.732
	$\kappa_m$	0.821	0.763	0.734	0.505
LED	G-mean	<b>0.943</b>	0.892	0.788	0.781
	F-measure	<b>0.901</b>	0.869	0.830	0.727
	$\kappa_m$	<b>0.693</b>	0.660	0.639	0.555
RBF	G-mean	0.878	<b>0.894</b>	0.863	0.801
	F-measure	0.890	<b>0.901</b>	0.848	0.790
	$\kappa_m$	0.675	<b>0.698</b>	0.641	0.650
Gas Sensor	G-mean	<b>0.883</b>	0.855	0.775	0.780
	F-measure	<b>0.833</b>	0.828	0.782	0.699
	$\kappa_m$	<b>0.822</b>	0.794	0.706	0.681
HAR	G-mean	<b>0.904</b>	0.894	0.875	0.725
	F-measure	<b>0.801</b>	0.798	0.708	0.763
	$\kappa_m$	<b>0.701</b>	0.678	0.602	0.635
Covertime	G-mean	0.915	0.908	<b>0.923</b>	0.827
	F-measure	<b>0.858</b>	0.844	0.850	0.768
	$\kappa_m$	0.711	0.648	<b>0.751</b>	0.638

Bold entries are used to indicate the best results

**Fig. 6** Nemenyi graph ranking HT base classifier performance for various sampling methods



for RBF and LED, but Covertypes and HAR worked better with a queue size of 50. We subsequently report the results of a queue size of 50 against all data sets. It is noticeable that our extensive experimentation shows that the queue size does not depend on the size of the data set.

Next, we focus on the second set of experiments, where we utilize various base learners in our DynaQ algorithm. Figure 8 depicts the G-mean results for our data sets when assessing the performance of the DynaQ technique compared to the two baseline algorithms. The results clearly show the benefit of our DynaQ algorithm compared to the majority-class and no-change learners, which could not learn the concepts within our multi-class imbalanced streams. The HT ensemble reaches higher performance (in Shuttle by 1%, in LED by up to 4%, in Covertypes by 6%, and in HAR by 7%) than the HAT or SAMKNN ensembles. However, for the Weather, RBF, and Gas Sensor data sets, the HAT ensemble presents up to 4% better results than ensembles based on HT or SAMKNN. Recall that HT and HAT are both incremental tree learners. However, HAT is more adaptable with streaming data because it uses an adaptive sliding window (ADWIN) algorithm as a drift detector error estimator and requires no parameters related to change control. We conclude that those data sets that work slightly better with HAT ensembles may include gradual or abrupt concept drift while streaming. Readers should notice that fast drift detection and the associated recovery process prevented a significant performance drop. Our results also indicate that the baseline no-change and majority-class learners produced low values among all data sets.

In the next set of experiments, we investigate the effect of the queue-based concept drift detection method on the DynaQ learning process. As expected, Fig. 9 indicates that the G-mean is higher when we include the DDM mechanisms during learning. This result suggests that because the queue-based sampling keeps the recent concepts of the stream, the use of drift detection clearly benefits the learning process. The fluctuation and recovery periods are different for various types of concept drifts. In the related visualization graphs, the reader will notice that the weather data experience abrupt drift while LED data show gradual drift. With drift detection, the performances against these two streams stay relatively high. Still, with drift detection, the learner can recover

and reach back to higher values after the drift happens. In this case, employing a DDM clearly prevents performance degradation. From Fig. 9, we find that RBF and HAR, susceptible to gradual drifts, do not suffer a drastic reduction in their total predictive accuracies. However, there are clear fluctuations related to gradual concept drifts handled by the DDM. The Gas Sensor and Covertypes data sets encounter sudden performance decrement and variation among the stream caused by abrupt drifts. However, when utilizing DDM, these changes recovered quickly without harming performance.

Finally, we present our results when contrasting different online multi-class approaches. Specifically, Tables 3, 4, 5, and 6 presents the results of our comparative study contrasting the DynaQ, ROSE, KUE, OMCQ, IOE, MOOB, and MUOB algorithms. Table 3 includes results on all methods based on HT as the base classifier. The remainder of the tables present results on base-classifier agnostic methods with three component classifiers (HT, HAT, and SAMKNN) and three evaluation metrics (G-mean, F-measure and  $Kappa_m$ ). Table 3 shows that DynaQ outperforms all approaches over different measures. Although DynaQ and ROSE produce competitive results, DynaQ yields better results in 25 out of 30 cases than ROSE. However, ROSE performs slightly better on the Weather, RBF3, and Covertypes streams. This result may be attributed to the abrupt drifts that these three data streams experience, where the background classifiers utilized by the ROSE algorithm facilitate the learning process. Table 4 shows that the DynaQ algorithm produced the highest values in terms of G-mean for 18 out of the 21 cases. The results also indicate that the three base learners produce comparable results in terms of G-mean, while no single base learner consistently outperforms the other two in all settings. However, the reader will notice that there are often substantial differences in G-mean, e.g., for LED and Covertypes, when contrasting the queue-based algorithms with IOE, MOOB, and MUOB.

A similar observation holds for the F-measure depicted in Table 5, where DynaQ produced the highest results for all data streams and base learners. The design of DynaQ implies that when there is no wait time for each queue to commence training the model, and the queues are resampled with the most recent data based on recall rates. Thus, the algorithm allows for better generalization on the incoming stream.

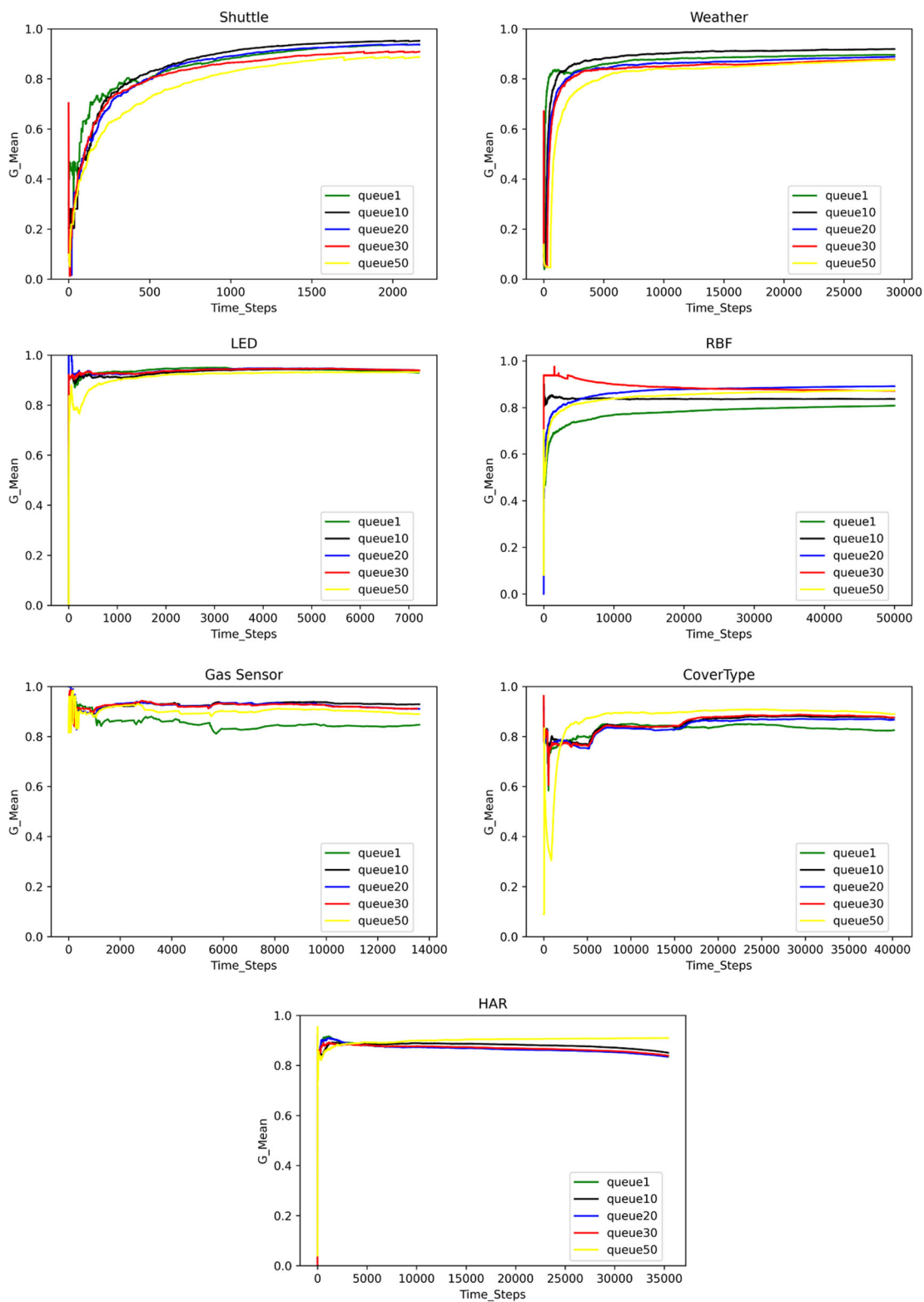
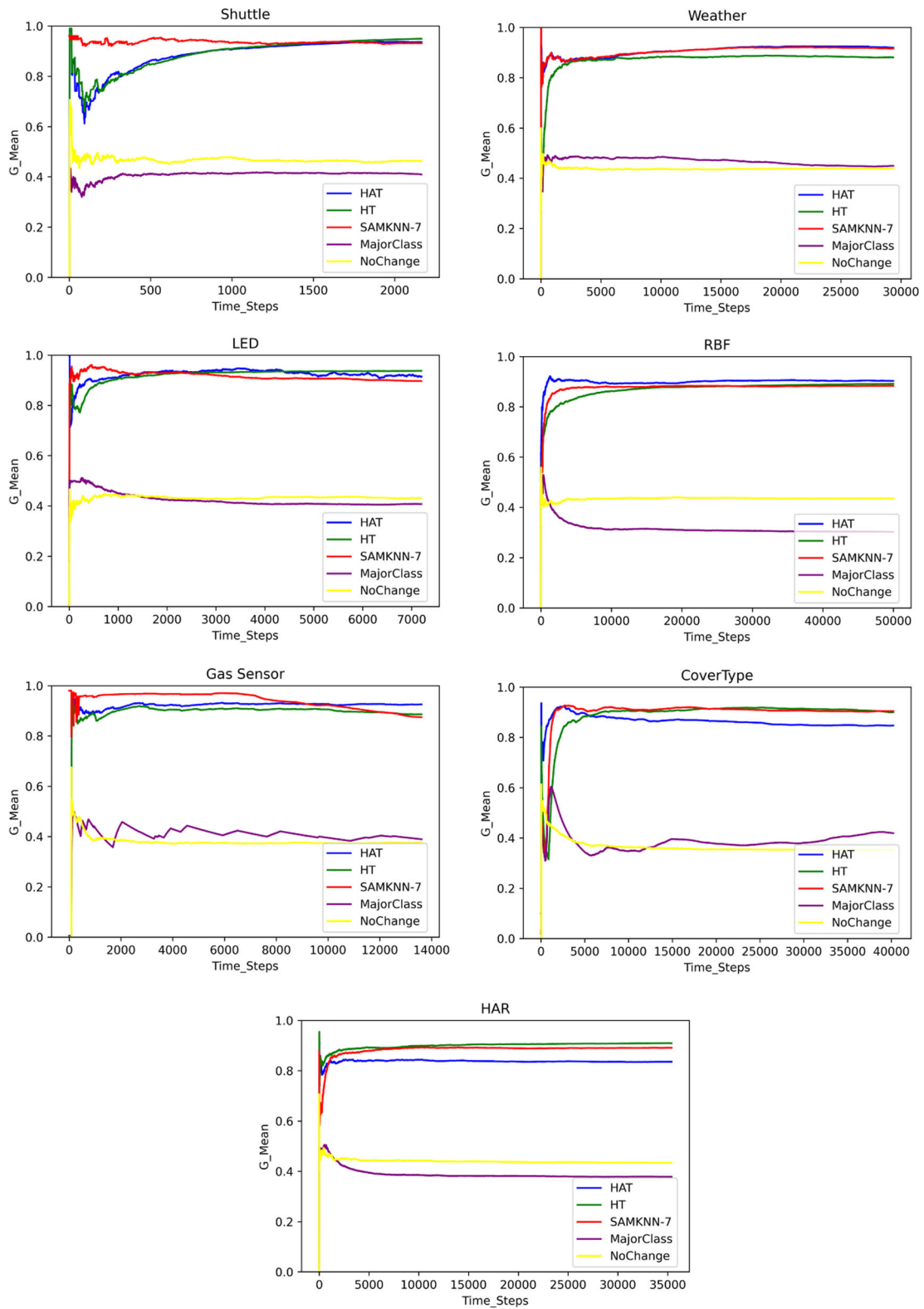


Fig. 7 Performance comparison against different queue sizes on DynaQ



**Fig. 8** Results of DynaQ algorithm for different base learner when compared with baselines

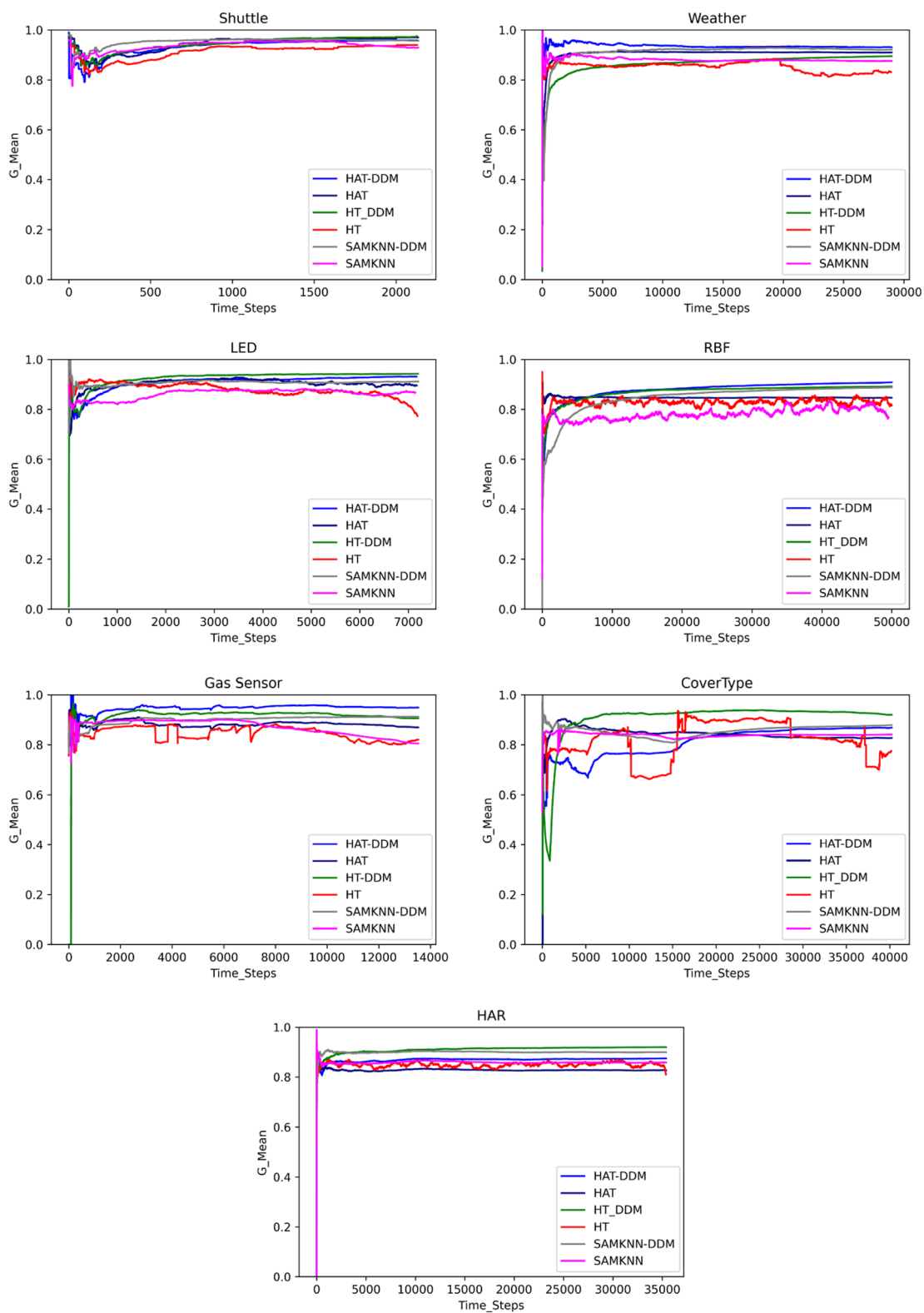


Fig. 9 Results with and without concept drift detection for DynaQ ensembles



**Table 3** HT results against data streams

Data	Metric	Hoefding-Tree						
		DynaQ	OMCQ	IOE	MOOB	MUOB	ROSE	KUE
Shuttle	G-mean	<b>0.971</b>	0.951	0.901	0.838	0.800	0.955	0.895
	F-measure	<b>0.952</b>	0.942	0.849	0.815	0.764	0.802	0.782
	$\kappa_m$	<b>0.837</b>	0.742	0.683	0.483	0.472	0.822	0.813
Weather	G-mean	0.862	0.857	0.840	0.791	0.787	<b>0.871</b>	0.843
	F-measure	<b>0.851</b>	0.831	0.766	0.752	0.749	0.796	0.770
	$\kappa_m$	0.821	0.713	0.714	0.605	0.601	<b>0.824</b>	0.777
LED	G-mean	<b>0.943</b>	0.912	0.782	0.780	0.777	0.940	0.893
	F-measure	<b>0.901</b>	0.888	0.890	0.747	0.710	0.867	0.832
	$\kappa_m$	<b>0.693</b>	0.560	0.659	0.455	0.495	0.664	0.649
RBF	G-mean	<b>0.878</b>	0.844	0.843	0.701	0.617	0.823	0.811
	F-measure	<b>0.890</b>	0.819	0.828	0.590	0.476	0.729	0.713
	$\kappa_m$	<b>0.675</b>	0.615	0.601	0.450	0.325	0.624	0.630
RBF2	G-mean	<b>0.825</b>	0.784	0.762	0.692	0.688	0.812	0.800
	F-measure	<b>0.818</b>	0.769	0.715	0.630	0.621	0.799	0.782
	$\kappa_m$	<b>0.638</b>	0.598	0.544	0.568	0.487	0.602	0.652
RBF3	G-mean	<b>0.811</b>	0.782	0.723	0.800	0.719	0.767	0.738
	F-measure	<b>0.794</b>	0.724	0.751	0.732	0.689	0.751	0.722
	$\kappa_m$	0.608	0.583	0.512	0.534	0.492	<b>0.615</b>	0.565
Gas Sensor	G-mean	<b>0.883</b>	0.860	0.815	0.740	0.550	0.710	0.694
	F-measure	<b>0.833</b>	0.833	0.760	0.587	0.414	0.683	0.603
	$\kappa_m$	<b>0.822</b>	0.780	0.786	0.511	0.495	0.792	0.778
HAR	G-mean	<b>0.904</b>	0.834	0.872	0.825	0.830	0.730	0.714
	F-measure	<b>0.801</b>	0.781	0.778	0.743	0.652	0.716	0.668
	$\kappa_m$	<b>0.701</b>	0.641	0.666	0.655	0.612	0.671	0.663
Coverttype	G-mean	0.915	0.888	0.793	0.627	0.582	<b>0.919</b>	0.784
	F-measure	<b>0.858</b>	0.844	0.667	0.661	0.387	0.816	0.751
	$\kappa_m$	0.711	0.618	0.651	0.438	0.291	<b>0.726</b>	0.715
Intel Sensor	G-mean	0.723	0.64	0.58	0.66	0.51	<b>0.730</b>	0.624
	F-measure	<b>0.591</b>	0.508	0.543	0.509	0.484	0.523	0.518
	$\kappa_m$	<b>0.433</b>	0.421	0.409	0.375	0.333	0.419	0.420

Bold entries are used to indicate the best results

Recall that OMCQ maintains queues that only include original instances. In the case of highly imbalanced streams, the minority queues may still contain instances representing the concept prior to drift, thus leading to a degradation in performance. The IOE technique also provides competitive results with DynaQ, which suggests the benefit of balancing performance based on the recall parameter. However, the results suggest that the undersampling of majority instances do not benefit learning. The superior results of DynaQ are especially evident for the LED, RBF, Gas Sensor and Coverttype data streams, which contained severe abrupt or gradual concept

drifts. The MOOB and MUOB algorithms struggled to obtain high values against such evolving streams.

Regarding the  $\kappa_m$  statistical results presented in Table 6, the DynaQ and IOE algorithms yielded comparable values, with DynaQ in first place and IOE in second place. In the case of the LED and RBF data sets, employing SAMKNN results in  $\kappa_m$  values up to 10% higher than HT and HAT. The reader will notice that recall balancing causes improvement on synthetic streams using SAMKNN while the performance decreases drastically in the HAR and Weather data sets. For the Coverttype, Gas Sensor and Weather data sets, even

**Table 4** G-mean results against data streams

Data	Classifier	G-mean				
		DynaQ	OMCQ	IOE	MOOB	MUOB
Shuttle	HAT	<b>0.964</b>	0.950	0.957	0.867	0.826
	HT-Tree	<b>0.971</b>	0.951	0.901	0.838	0.800
	SAMKNN-7	0.963	0.968	<b>0.970</b>	0.896	0.845
Weather	HAT	<b>0.905</b>	0.854	0.825	0.785	0.790
	HT-Tree	<b>0.862</b>	0.857	0.840	0.791	0.787
	SAMKNN-7	<b>0.890</b>	0.880	0.876	0.839	0.771
LED	HAT	<b>0.935</b>	0.921	0.898	0.784	0.789
	HT-Tree	<b>0.943</b>	0.912	0.782	0.780	0.777
	SAMKNN-7	0.908	<b>0.911</b>	0.856	0.767	0.792
RBF	HAT	<b>0.886</b>	0.882	0.806	0.620	0.645
	HT-Tree	<b>0.878</b>	0.844	0.843	0.701	0.617
	SAMKNN-7	<b>0.871</b>	0.861	0.823	0.777	0.643
Gas Sensor	HAT	<b>0.920</b>	0.866	0.901	0.619	0.640
	HT-Tree	<b>0.883</b>	0.860	0.815	0.740	0.550
	SAMKNN-7	<b>0.880</b>	0.874	0.824	0.764	0.757
HAR	HAT	0.835	<b>0.839</b>	0.827	0.806	0.796
	HT-Tree	<b>0.904</b>	0.834	0.872	0.825	0.830
	SAMKNN-7	<b>0.887</b>	0.883	0.867	0.839	0.767
Coverttype	HAT	<b>0.857</b>	0.844	0.848	0.835	0.772
	HT-Tree	<b>0.915</b>	0.888	0.793	0.627	0.582
	SAMKNN-7	<b>0.917</b>	0.909	0.868	0.818	0.800

Bold entries are used to indicate the best results

though they experience abrupt drifts that may change the behaviour of major classes, our results are up to 8% higher than other methods. In DynaQ, we do not make any assumptions about separating majority and minority classes. This aids DynaQ to reach promising  $\kappa_m$  performance. In our overall analyses, DynaQ using HT as a base learner performs better than, or similar to HAT, as a base learner. Also, the comparison between MOOB and MUOB shows that oversampling benefits the method more than undersampling. This observation further reinforces our design choice of using an oversampling approach.

Next, we present the results of the Nemenyi posthoc test [26] shown in Figs. 10, 11, 12, and 13, where  $\beta$  is set to 0.05. This test highlights the contrasts in the algorithms against all data sets, where a lower rank means a better predictive performance (G-mean, F-measure, and  $kappa_m$ ). In Fig. 10 DynaQ is ranked first, followed by ROSE, OMCQ, and IOE. While ROSE benefits from combining backup ensembles, DynaQ provides a stable backup in case of drifts by allocating small-sized queues to each class. For this reason, DynaQ delivers consistent performance and high ranks across all data streams.

Figures 11 and 12 show a critical difference between our DynaQ algorithm and the IOE, MOOB, and MUOB techniques, for the F-measure and G-mean metrics. The reader will notice no significant statistical differences between the DynaQ and OMCQ methods. The DynaQ and OMCQ methods benefit from their underlying queue-based learning processes. However, our DynaQ method ranks first and OMCQ second. This ranking indicates the strength of combining queue-based learning with minority-class oversampling and online ensemble learning.

Figure 13 indicates that DynaQ and IOE present similar  $\kappa_m$  values, with DynaQ ranked first, while outperforming OMCQ, MOOB, and MUOB. The DynaQ and IOE algorithms both utilize recall rates that aid the learners in handling the change in class labels caused by an evolving and skewed stream. The main difference between these approaches is that DynaQ employs oversampling, while IOE combines oversampling and undersampling. Since DynaQ is ranked first, one may conclude that undersampling is unnecessary in most settings. The results further indicate the value of balancing recall rates to improve performance in a

**Table 5** F-measure results against data streams

Data	Classifier	F-measure				
		DynaQ	OMCQ	IOE	MOOB	MUOB
Shuttle	HAT	<b>0.944</b>	0.945	0.936	0.833	0.796
	HT-Tree	<b>0.952</b>	0.942	0.849	0.815	0.764
	SAMKNN-7	<b>0.967</b>	0.964	0.947	0.879	0.723
Weather	HAT	<b>0.906</b>	0.838	0.785	0.733	0.750
	HT-Tree	<b>0.851</b>	0.831	0.766	0.752	0.749
	SAMKNN-7	<b>0.874</b>	0.868	0.745	0.786	0.694
LED	HAT	<b>0.926</b>	0.902	0.905	0.756	0.761
	HT-Tree	<b>0.901</b>	0.888	0.890	0.747	0.710
	SAMKNN-7	<b>0.910</b>	0.893	0.862	0.699	0.754
RBF	HAT	<b>0.877</b>	0.854	0.830	0.478	0.494
	HT-Tree	<b>0.890</b>	0.819	0.828	0.590	0.476
	SAMKNN-7	<b>0.858</b>	0.841	0.836	0.721	0.547
Gas Sensor	HAT	<b>0.901</b>	0.826	0.836	0.640	0.544
	HT-Tree	<b>0.833</b>	<b>0.833</b>	0.760	0.587	0.414
	SAMKNN-7	<b>0.889</b>	0.865	0.818	0.710	0.737
HAR	HAT	<b>0.778</b>	0.745	0.771	0.674	0.640
	HT-Tree	<b>0.801</b>	0.781	0.778	0.743	0.652
	SAMKNN-7	<b>0.819</b>	0.811	0.772	0.704	0.655
Coverttype	HAT	<b>0.761</b>	0.728	0.730	0.715	0.661
	HT-Tree	<b>0.858</b>	0.844	0.667	0.661	0.387
	SAMKNN-7	<b>0.863</b>	0.846	0.782	0.682	0.641

Bold entries are used to indicate the best results

multi-class imbalanced setting when focusing on the  $kappa_m$  metric.

Our experimental evaluations indicate the strength of combining a queue-based approach with dynamic minority class oversampling, concept drift detection, and online ensemble learning.

## 5 Conclusion

The paper addressed the challenge of online learning from evolving multi-class imbalanced data streams susceptible to concept drifts. The DynaQ algorithm combined class-based queues, dynamic oversampling of minority classes, online ensembles based on sliding windows, and class-based concept drift detection. An advantage of the DynaQ method is that it operates independently of a base classifier, thus providing a general framework for dealing with evolving multi-class streams. Our experimental results showed the benefits of our

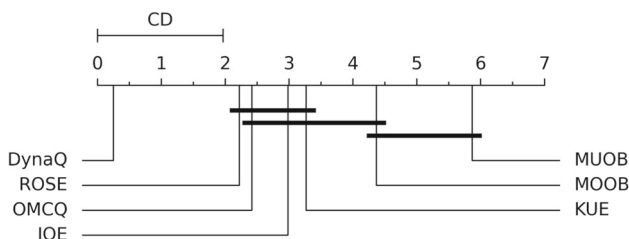
approach, and we determined that the DynaQ method constructs accurate models.

Our DynaQ algorithm is highly suitable for scenarios where minority classes may become majority classes, and vice versa. Future work will specifically address highly skewed distributions, where minority instances may arrive in bursts. We will analyze the effect of the number of classes on the performance of each class and our method. We will also research other resampling methods, such as extending SCUT-DS [41], an approach for stationary streams that combines SMOTE-based oversampling with cluster-based undersampling. Another interesting idea is to extend the SOUP bagging ensemble-based algorithm that uses the notion of safe levels to resample data in an offline setting [25]. Moreover, we are interested in examining the impact of our proposed approach in the domain of privacy [53, 54]. We intend to extend DynaQ to incorporate privacy concerns and conduct experiments using time-series and text datasets to evaluate its effectiveness.

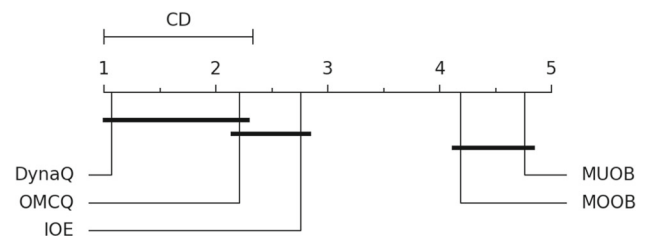
**Table 6**  $\kappa_m$  results against data streams

$\kappa_m$ Data	Classifier	DynaQ	OMCQ	IOE	MOOB	MUOB
Shuttle	HAT	<b>0.825</b>	0.715	0.738	0.430	0.391
	HT-Tree	<b>0.837</b>	0.742	0.683	0.483	0.472
	SAMKNN-7	<b>0.819</b>	0.713	0.760	0.660	0.531
Weather	HAT	<b>0.784</b>	0.780	0.746	0.715	0.714
	HT-Tree	<b>0.821</b>	0.713	0.714	0.605	0.601
	SAMKNN-7	<b>0.642</b>	0.613	0.642	0.340	0.321
LED	HAT	<b>0.662</b>	0.616	0.638	0.450	0.454
	HT-Tree	<b>0.693</b>	0.560	0.659	0.455	0.495
	SAMKNN-7	<b>0.789</b>	0.726	<b>0.788</b>	0.392	0.445
RBF	HAT	<b>0.681</b>	0.618	0.612	0.460	0.338
	HT-Tree	<b>0.675</b>	0.615	0.601	0.450	0.325
	SAMKNN-7	<b>0.733</b>	0.710	<b>0.732</b>	0.525	0.482
Gas Sensor	HAT	<b>0.826</b>	0.756	0.760	0.516	0.500
	HT-Tree	<b>0.822</b>	0.780	0.786	0.511	0.495
	SAMKNN-7	<b>0.816</b>	0.754	0.795	0.679	0.654
HAR	HAT	<b>0.685</b>	0.618	0.654	0.581	0.427
	HT-Tree	<b>0.701</b>	0.641	0.666	0.655	0.612
	SAMKNN-7	<b>0.566</b>	0.514	0.510	0.483	0.464
Coverttype	HAT	<b>0.684</b>	0.611	0.627	0.385	0.310
	HT-Tree	<b>0.711</b>	0.618	0.651	0.438	0.291
	SAMKNN-7	<b>0.718</b>	0.637	0.648	0.580	0.339

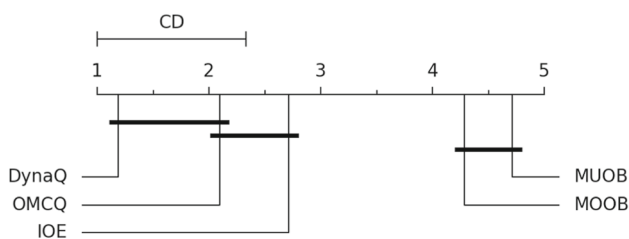
Bold entries are used to indicate the best results



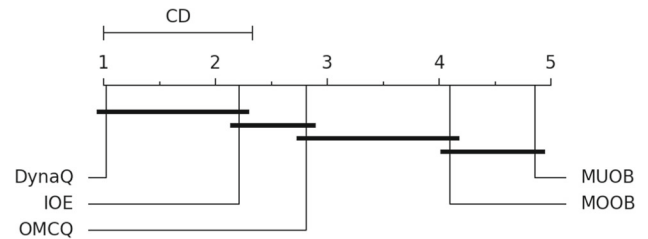
**Fig. 10** Nemenyi graph ranking algorithms based on HT base classifier



**Fig. 12** Nemenyi graph ranking F-measure results among various algorithms



**Fig. 11** Nemenyi graph ranking G-mean results among various algorithms



**Fig. 13** Nemenyi graph ranking  $Kappa_m$  results among various algorithms

**Author Contributions** The three authors contributed equally to the paper.

**Funding** The authors would like to thank the National Science and Engineering Research Council (NSERC) of Canada for its financial support through the Discovery Grants Program.

**Availability of data and material** All data used in our experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes.

**Code Availability** All source code required for conducting experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes.

## Declarations

**Conflicts of interest** Not applicable.

**Ethics approval** Not applicable.

**Consent to participate** Not applicable.

**Consent for publication** Not applicable.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Aguiar G, Krawczyk B, Cano A (2022) A survey on learning from imbalanced data streams: taxonomy, challenges, empirical study, and reproducible experimental framework. [arXiv:2204.03719](https://arxiv.org/abs/2204.03719)
2. Alcalá-Fdez J, Fernández A, Luengo J et al (2011) Keel data mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J Mult-Valued Log Soft Comput* 17(2–3):255–287
3. Alliance Canada Compute (2022) Available Resources. last access on November 2022 <https://alliancecan.ca>
4. Aminian E, Ribeiro RP, Gama J (2021) Chebyshev approaches for imbalanced data streams regression models. *Data Min Knowl Discov* 35:2389–2466
5. Bernardo A, Della Valle E (2021) Smote-ob: Combining smote and online bagging for continuous rebalancing of evolving data streams. In: 2021 IEEE International Conference on Big Data (Big Data). IEEE, p 5033–5042
6. Bernardo A, Della Valle E (2021) Vfc-smote: very fast continuous synthetic minority oversampling for evolving data streams. *Data Min Knowl Discov* 35(6):2679–2713
7. Bernardo A, Della Valle E (2022) An extensive study of c-smote, a continuous synthetic minority oversampling technique for evolving data streams. *Expert Syst Appl* 196:116630
8. Bifet A, Frank E (2010) Sentiment knowledge discovery in Twitter streaming data. In *International Conference on Discovery Science*, p 1–15
9. Bifet A, Gavaldá R (2009) Adaptive parameter-free learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*. p 249–260
10. Bifet A, Holmes G, Pfahringer B, et al (2010) MOA: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the First Workshop on Applications of Pattern Analysis*, p 44–50
11. Bifet A, de Francisci Morales G, Read J, et al (2015) Efficient online evaluation of big data stream classifiers. In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, p 59–68
12. Blackard J (1998) UCI Machine Learning Repository. <https://doi.org/10.24432/C50K5N>
13. Bobowska B, Klikowski J, Woźniak M (2020) Imbalanced data stream classification using hybrid data preprocessing. In: *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II*, Springer, pp 402–413
14. Cano A, Krawczyk B (2020) Kappa updated ensemble for drifting data stream mining. *Mach Learn* 109:175–218
15. Cano A, Krawczyk B (2022) Rose: Robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams. *Mach Learn* 111(7):2561–2599
16. Casale P, Pujol O, Radeva P (2012) Using information on class interrelations to improve classification of multi-class imbalanced data. *Pers Ubiquitous Comput* 16(5):563–580
17. Czarnowski I (2022) Weighted ensemble with one-class classification and over-sampling and instance selection (wecoi): An approach for learning from imbalanced data streams. *J Comput Sci* 61:101614
18. Domingos P, Hulten G (2000) Mining high-speed data streams. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p 71–80
19. Fernández A, García S, Herrera F et al (2018) Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *J Artif Intell Res* 61:863–905
20. Fernández A, López V, Galar M et al (2013) Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowl Based Syst* 42: 97–110
21. Fernández A, García S, Galar M, et al (2018) Learning from imbalanced data stream. In: *Learning from Imbalanced Data Sets*, p 279–303
22. Floyd S, Viktor H (2019) Soft voting windowing ensembles for learning from partially labelled streams. *International Workshop on New Frontiers in Mining Complex Patterns*. Springer, Cham, pp 85–99
23. Gomes HM, Read J, Bifet A et al (2019) Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explore Newsllett* 21(2):6–22
24. Government C (2022) Historic climate data from environment and climate change canada. [https://climate.weather.gc.ca/historical\\_data/search\\_historic\\_data\\_e.html](https://climate.weather.gc.ca/historical_data/search_historic_data_e.html)
25. Janicka M, Lango M, Stefanowski J (2019) Using information on class interrelations to improve classification of multi-class imbalanced data: a new re-sampling algorithm. *Int J Appl Math Comput Sci* 29(4):769–781
26. Japkowicz N, Shah M (2011) *Evaluating learning algorithms: A classification perspective*. Cambridge University Press

27. Karlos S, Kostopoulos G, Kotsiantis S (2020) A soft-voting ensemble based co-training scheme using static selection for binary classification problems. *Algorithms* 13(1):26
28. Klikowski J, Woźniak M (2020) Employing one-class svm classifier ensemble for imbalanced data stream classification. In: *Computational Science—ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV 20*, Springer, p 117–127
29. Krawczyk B, Minku L, Gama J et al (2017) Ensemble learning for data stream analysis: a survey. *Inform Fusion* 37:132–156
30. Krawczyk B, Galar M, Woźniak M et al (2018) Dynamic ensemble selection for multi-class classification with one-class classifiers. *Pattern Recognit* 83:34–51
31. Ksieniewicz P (2021) The prior probability in the batch classification of imbalanced data streams. *Neurocomputing* 452:309–316
32. Liu W, Zhang H, Ding Z et al (2021) A comprehensive active learning method for multiclass imbalanced data streams with concept drift. *Knowl Based Syst* 215:106778
33. Losing V, Hammer B, Wersing H (2017) Self-adjusting memory: How to deal with diverse drift types. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)* pp 4899–4903
34. Losing V, Hammer B, Wersing H (2018) Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing* 275:1261–1274
35. Lu J, Liu A, Dong F et al (2018) Learning under concept drift: A review. *IEEE Trans Knowl Data Eng* 31(12):2346–2363
36. Lyon RJ, Brooke J, Knowles JD, et al (2014) Hellinger distance trees for imbalanced streams. In: *2014 22nd International Conference on Pattern Recognition, IEEE*, p 1969–1974
37. Madden S (2004) Intel berkeley research lab. last access May 2023 <http://db.csail.mit.edu/labdata/labdata.html>.
38. Malialis K, Panayiotou C, Polycarpou M (2018) Queue-based resampling for online class imbalance learning. In: *International Conference on Artificial Neural Networks*, p 498–507
39. Marie M, Deza D (2018) *Encyclopedia of distances*. Springer
40. Montiel J, Read J, Bifet A et al (2018) Scikit-multiflow: A multi-output streaming framework. *J Mach Learn Res* 19(72):1–5
41. Olaitan O, Viktor H (2018) SCUT-DS: Learning from Multi-class imbalanced Canadian weather data. In: *International Symposium on Methodologies for Intelligent Systems*, p 291–301
42. Oza N, Russell S (2001) Online bagging and boosting. In: *International Workshop on Artificial Intelligence and Statistics (PMLR)*, p 229–236
43. Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: Machine learning in Python. *J Mach Learn Res* 12:2825–2830
44. Read B, Bifet A, B. P, et al (2012) Batch-incremental versus instance-incremental learning in dynamic and evolving data. In: *International Symposium on Intelligent Data Analysis*, p 313–323
45. Ren S, Zhu W, Liao B et al (2019) Selection-based resampling ensemble algorithm for nonstationary imbalanced stream data learning. *Knowl Based Syst* 163:705–722
46. Sadeghi F, Viktor H (2021) Online-mc-queue: Learning from imbalanced multi-class streams. *Third International Workshop on Learning with Imbalanced Domains: Theory and Applications (LIDTA)*. *Proc Mach Learn Res* 154:21–34
47. Vafaie P, Viktor H, Michalowski W (2019) Multi-class imbalanced semi-supervised learning from streams through online ensembles. *International Conference on Data Mining Workshops (ICDMW)* pp 867–874
48. Vafaie P, Viktor H, Michalowski W (2020) Multi-class imbalanced semi-supervised learning from streams through online ensembles. In: *2020 International Conference on Data Mining Workshops (ICDMW), IEEE*, p 867–874
49. Vergara A, Vembu S, Ayhan T et al (2012) Chemical gas sensor drift compensation using classifier ensembles. *Sens Actuators B Chem* 166:320–329
50. Wang S, Minku L, Ghezzi D, et al (2013) Concept drift detection for online class imbalance learning. In: *International Joint Conference on Neural Networks (IJCNN '13)*, p 1–10
51. Wang S, Minku L, Yao X (2014) Resampling-based ensemble methods for online class imbalance learning. *IEEE Trans Knowl Data Eng* 27(5):1356–1368
52. Wang S, Minku L, Yao X (2016) Dealing with multiple classes in online class imbalance learning. *Int Jt Conf Artif Intell* 2118–2124
53. Wu Z, Shen S, Lian X et al (2020) A dummy-based user privacy protection approach for text information retrieval. *Knowl Based Syst* 195:105679
54. Wu Z, Lu C, Zhao Y et al (2021) The protection of user preference privacy in personalized information retrieval: challenges and overviews. *Libri* 71(3):227–237
55. Zhang H, Liu W, Liu Q (2020) Reinforcement online active learning ensemble for drifting imbalanced data streams. *IEEE Trans Knowl Data Eng* 34(8):3971–3983
56. Žliobaitė I, Bifet A, Read PBJ, et al (2015) Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Mach Learn* 455–482

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Farnaz Sadeghi** received her Master's in Computer Science, focusing on complex network science and online social network analysis, at Tehran University in 2018. Presently, she is pursuing her Ph.D. at the School of Electrical Engineering and Computer Science at the University of Ottawa. Her research primarily revolves around online learning from evolving streams, multi-class imbalance, fairness-aware learning, and the interpretability of machine learning models.



**Herna Viktor** obtained a Ph.D. in Computer Science from Stellenbosch University in 1999. She is the Director and a Full Professor at the School of Electrical Engineering and Computer Science (EECS) at the University of Ottawa. Her research interests include machine learning algorithms for sequential and temporal data, bias-free and discrimination-aware learning, and class imbalance. Professor Viktor has over twenty years of experience designing, implementing, and applying machine

learning solutions in healthcare, cybersecurity, finance, and biomedicine. Her work has received numerous recognitions and awards.



**Parsa Vafaie** holds a Bachelor's in Computer Engineering from the University of Tehran (2019) and a Master's in Computer Science from the University of Ottawa (2021). His current work includes detecting financial criminal activities using Machine Learning. His research interests include learning with missing labels and data imbalance and applications of machine learning in finance and health care.