Online-MC-Queue: Learning from Imbalanced Multi-Class Streams

Farnaz Sadeghi FSADE079@UOTTAWA.CA and **Herna L. Viktor** HVIKTOR@UOTTAWA.CA
School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada

Editors: Nuno Moniz, Paula Branco, Luís Torgo, Nathalie Japkowicz, Michał Woźniak and Shuo Wang.

Abstract

Online supervised learning from fast-evolving data streams has application in many areas. The development of techniques with highly skewed class distributions (or 'class imbalance') is an important area of research in domains such as manufacturing, the environment, and health. Solutions should not only be able to analyse large repositories in near real-time but also be capable of providing accurate models to describe rare classes that may appear infrequently or in bursts, while continuously accommodating new instances. Although online learning methods have been proposed to handle binary class imbalance, solutions suitable for multi-class streams with varying degrees of imbalance in evolving streams have received limited attention. In order to address this knowledge gap, this paper introduces the Online-MC-Queue (OMCQ) algorithm for online learning in multi-class imbalanced settings. Our approach utilises a queue-based resampling method that dynamically creates an instance queue for each class. The number of instances is balanced by maintaining a queue threshold and removing older samples during training. In addition, new and rare classes are dynamically added to the training process as they appear. Our experimental results confirm a noticeable improvement in minority-class detection and in classification performance. A comparative evaluation shows that the OMCQ algorithm outperforms the state-of-the-art.

Keywords: Online learning . multi-class imbalance . data streams

1. Introduction

Algorithms for online learning from data streams use techniques that process each incoming example "on arrival" without the need for storage and multiple scans while maintaining a model that reflects the current data. This type of learning contributes to various real-world applications, such as spam filtering, fault detection in manufacturing, and medical diagnosis (Gomes et al., 2019). Learning from such streaming data is challenging, especially in the presence of multiple skewed class distributions, also known as "multi-class imbalance". In this scenario, a large number of majority-class examples may lead to the minority classes being ignored. This problem is aggravated in an online setting because a steady arrival of minority instances cannot be guaranteed, and a minority class may become a majority concept and vice versa (Fernandez et al., 2018). In addition, evolving streams are susceptible to concept drifts, which is the phenomenon of unexpected changes in the underlying data distribution (Lu et al., 2018).

Only a limited number of studies have addressed the combined problem of learning from such evolving streams that contain multi-class imbalance. Recent studies (Fernandez et al.,

2018) have mainly focused on binary imbalanced data, propose modifications of resampling methods, or worked only with stationary streams. Also, they do not address dynamic class evolutions. In this paper, we introduce the Online-MultiClass-Queue (OMCQ) approach, which learns directly from the original data in a multi-class imbalanced setting. Our multi-class learning algorithm maintains separate queues for each class, thus facilitating training based on the current data. In our work, we do not generate any artificial samples, since introducing "synthetic" cases in real-world domains such as health care and cybersecurity may be questionable to domain experts. Our algorithm is able to dynamically adapt to changes in label arrivals. That is, we do not make any assumptions regarding the frequency of classes, which implies that minority classes may become majority classes, and vice versa.

Furthermore, we introduce a drift detection mechanism able to separately detect changes in the individual classes, while simultaneously handling multiple class drifts. The novelty of this approach is that, for each class, we maintain a queue of instances and flag for drifts when a detection threshold is reached. Thus, drifts in minority classes with fewer samples are not ignored. Our algorithm combines batch-incremental and instance-incremental online learning. That is, initially a batch of data with all classes is presented to the learner and it subsequently proceeds to update the model with new instances as they arrive. Our experimental results confirm that our algorithm is efficient in terms of storage space and multi-class concept separation.

The paper is organized as follows. Section 2 presents related work while Section 3 introduces the OMCQ 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 $Y = \{c_1, c_2, \ldots, c_N\}$ and N 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 (Gomes et al., 2019). In this paper, we focus on online learning from multi-class imbalanced data, where $N > 2$.

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. Resampling is an effective data-level approach that proceeds independently of the learning algorithm; this method has been used in the data stream setting for binary classification problems. The major types of resampling are oversampling (increasing the number of minority-class examples), undersampling (reducing the number of majority-class examples), and hybrid sampling (Gomes et al., 2019). However, under-sampling methods may lead to crucial information being overlooked, while oversampling may potentially introduce artificial instances, which may be deemed unacceptable in real-world domains.

For example, in (Wang et al., 2014), the authors integrate 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 dynamically 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 (Malialis et al., 2018) 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 (Malialis et al., 2018). The algorithm employs an interleaved test-then-train evaluation (Bifet and Frank, 2010) in which each individual 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 (Malialis et al., 2018) to the multi-class scenario by incorporating explicit concept drift detection during learning, while eliminating resampling.

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 (Fernandez et al., 2018). Current approaches in a batch setting are mostly based on binary decomposition techniques, algorithmic-level modifications using misclassification costs, and resampling methods (Wang et al., 2014). These algorithms are typically combinations of binarisation techniques that transform the original multi-class data into binary subsets. For instance, the authors of the One-versus-One (OVO) (Fernández et al., 2013) decomposition strategy, first select a subset from the original data that only contains the instances for each pair of classes, and proceed to train a binary classifier for each pair. On the other hand, in the One-versus-All (OVA) approach (Fernández et al., 2013), the authors decompose a multi-class dataset into several binary class problems and subsequently train single classifiers for each class, i.e. by considering a single class versus a combination of all the remaining classes. A disadvantage of binarisation techniques is that the interactions between multiple classes cannot be considered simultaneously.

A limited number of research studies considered multi-class online learning from evolving streams, focusing on resampling techniques. Specifically, in a recent study (Wang et al., 2016), 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 re-sampling to overcome class imbalance, with the framework of Online Bagging (OB), as introduced above (Wang et al., 2014). In MOOB, the minimum (w_{min}) and maximum (w_{max}) class sizes among all classes

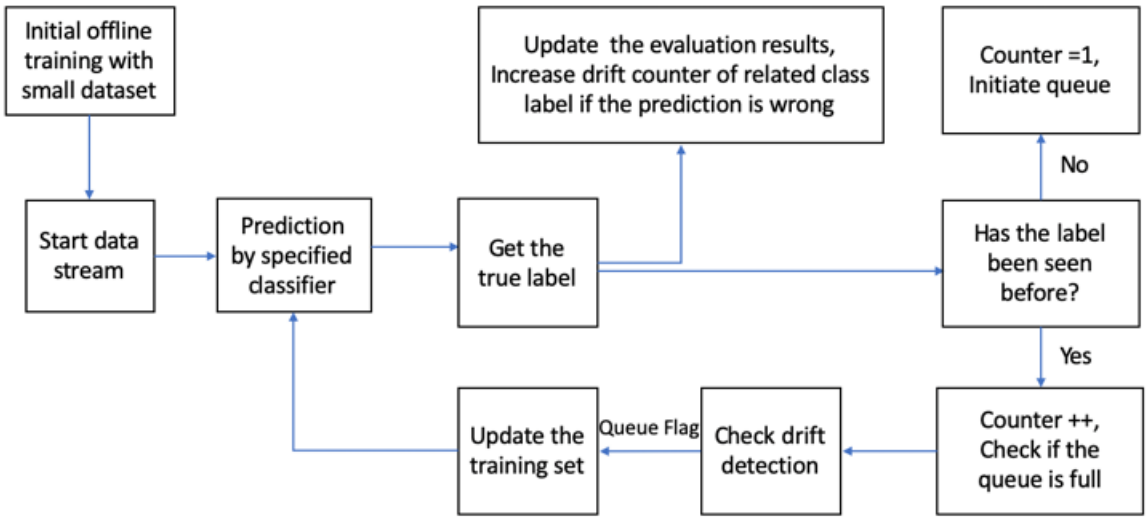

Figure 1: High-level overview of Online-MC-Queue methodology

are calculated at each time step. Subsequently, the algorithm sets $\lambda = \frac{w_{\max}}{w_j^t}\hat{y}_t$, where w_j^t is the size of class c_k at time t. This approach thus ensures that minority classes have a larger sampling rate. The MUOB algorithm inverses this idea, where λ controls the undersampling rate. Thus, 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, while in MUOB undersampling is used to reduce the chance of learning majority-class examples. These algorithms are able to process multi-classes directly without using class decomposition, but the performance is based on underlying assumptions, such as that sampling is efficient, useful, and does not introduce bias. In addition, MOOB and MUOB do not include a drift detector.

In the next section, we introduce our OMCQ algorithm, which learns directly from the original data without any resampling and incorporates a drift detector mechanism.

3. Online Multi-Class Queue-Based Learning

Our OMCQ framework maintains a queue for each one of our multiple classes, consisting of two stages: queue construction and online learning. Initially, all queues will be empty. As instances arrive, they are added to the appropriate queue, as per their true label. When a queue has been filled up, training commences.

We subsequently implement a forgetting mechanism, where the first elements are removed as the data continue to stream in. In other words, our algorithm initially 'waits' until we have enough instances in our queues and then proceeds to incrementally update the model using a sliding window (Read et al., 2012). That is, we assume that the number of instances from the minority classes are sufficient in order to fill a queue, in order to commence training. Figure 1 illustrates how our contributions fit together and operate in one iteration of an interleaved test-then-train loop using instance-by-instance learning.

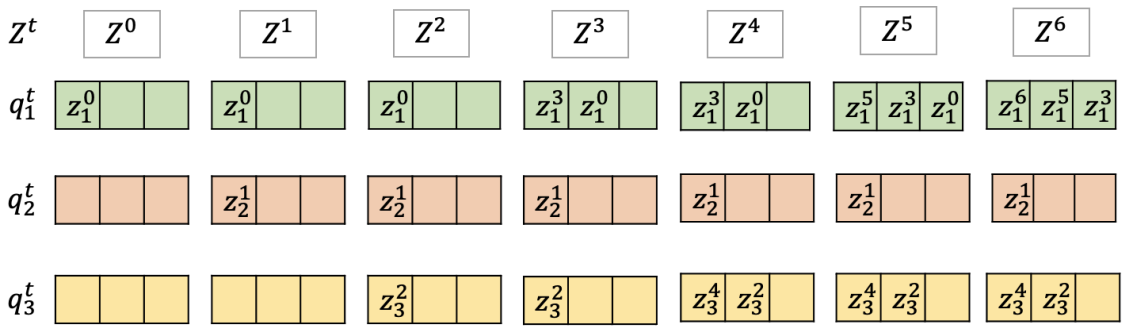

Figure 2: Example of $Queue_3$ resampling (adapted from (Malialis et al., 2018))

The online learning phase of OMCQ incorporates three processes; dynamic class balancing, concept drift detection, and evaluation. In the class balancing module, our algorithm creates a queue space to separate the instances from each class as they arrive within the stream. The concept drift detector captures changes in the data distributions by adapting the DDM algorithm (Wang et al., 2013) and subsequently updating the instances in the queues. The evaluator is used to predict the class label of arriving instances and to update the evaluation metrics.

3.1. Queue-Based Learning

Incremental online learning is a subarea of online learning that are additionally bounded by memory resources and capability of continuous learning with limited data compared to offline learning. This method considers a given stream of data $x_1, x_2, ..., x_t$ and learns a sequence of models $h_1, h_2, ..., h_t$, where the models are updated as instances arrive. In our work, following (Malialis et al., 2018), we consider a sequence of streaming data $\{(x_1, y_1)$,- $\ldots, (x_t, y_t)\} \in R^n \times \{1, \ldots, K\}$ where n is the data dimension and K is the total number of classes. The key idea is to keep fixed number of examples (queue size denoted by L), for each class in a stream, to form the initial 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 $q_{C_k}^t = L$, where c_k is the class label received with the data. Together, the queues form a batch.

Figure 2 illustrates how $Queue_L$ works when $q = 3$. The upper part shows the examples that arrive at each time step, e.g., z^0 and z^4 arrive at $t = 0$ and $t = 4$, respectively. Assume that the dataset 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_k are placed at the front of the $queue_{c_k}$. When the queues fill up, we combine the full queues in order to form a training set and commence online learning. Here, we employ a forgetting mechanism, where the oldest instance will be removed from the head of the queue, thus implementing a sliding window.

As shown in Figure 3, B_t is a batch of data with size p, defined as the number of instances used to train the model. As an initial training step, we construct an initial model

with instances from all classes. For each arriving instance (x_t, y_t) at time t, the oldest sample from the queue to which y_t belong, will be removed, and the recent sample is added to batch B_t. Next, the learner will use batch B_t to update its model, i.e., the training process utilises a balanced set consisting of the most recent data. In this way, the algorithm waits until it has enough instances from the classes, including the current minority classes, before updating its model. It follows that both the batch size p and the sizes of the individual queues are highly domain dependent; these values are set by inspection. Recall that an underlying assumption of our approach is that we will have enough instances from all minority classes in order to fill a small queue that will be able to facilitate accurate model construction.

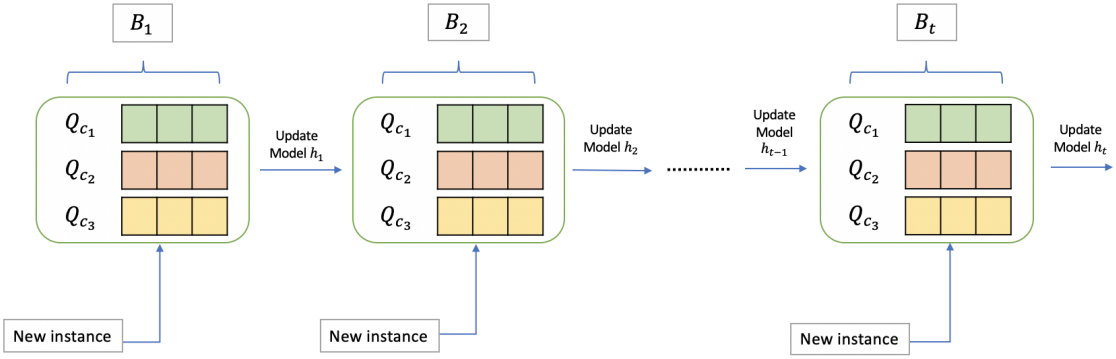

Figure 3: Illustration using batch-instance incremental model.

Algorithm 1 illustrates our methodology. Intuitively, our aim is to select examples from all classes in order to balance the training set during learning. When online learning starts, each instance receives the true label, and the example is then inserted into the queue corresponding to the actual label. Next, the label is compared with a list of class labels that have been seen so far. When an instance with a new class label arrives, the queue for the corresponding class is initiated. That is, if the true label of a new instance has not been seen before, we initialise Q_{c_k} corresponding to class c_k and this process continues until Q_{c_k} is full. At every iteration, the queue sizes for all current classes are assessed; if the queues are full, the classifier is able to update the learning model. The newly arriving instance in a queue replaces the oldest example. In this way, we always maintain the latest examples for each class and train using these sliding batches. Our framework is thus designed so that each class that appears in the stream will be used to update the model after reaching the queue size L, ensuring enough instances from minority classes are available prior to training.

Recall that we also included a concept drift detector to handle evolving streams. To this end, we adapted the DDM drift detection algorithm in our framework (Wang et al., 2013). 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 (Wang et al., 2013), we employ two counters for each class, where w_i denotes a warning level and d_i refers to 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

Algorithm 1: OMCQ algorithm

while *stream.has_more_instances() at each time step* t **do**
  x_i^t, y_i^t = get.next_instance();
  y_i^t_predict = Classifier.Predict(x_i^t); `// Test then train evaluation`
  $Drift\_list_{C_i} = [\,]$;
  $\alpha_{C_i}^t$ = TestForDrift(x_i^t, y_i^t);
  `// Test for concept drifts`
  if y^t *in previously seen class* C_i `// Add to queue`
  then
    Increment $Counter_{C_i}$; $Q_{C_i}^t = {}_{C_i}^{(t-1)}.append(i)$; Update($\alpha_{C_i}^t$)
  end
  else
    $Counter_{C_i} = 1$; Initialize$_{C_i}^t$,$Q_{C_i}^t.append(i)$; Update($\alpha_{C_i}^t$)
  end
  if *(*$\alpha_{C_i}^t \geq w_{C_i}^t$*) is TRUE* `// Warning, Start Drift list`
  then
    Add instance i to $Drift\_list_{(C_i)}$
  end
  if *(*$\alpha_{C_i}^t \geq d_{C_i}^t$*) is TRUE* `// Drift detected`
  then
    Update $Q_{C_i} = Drift\_list_{C_i}$
  end
  if $Q_{C_i} == L$ `// Online learning`
  then
    $Training\_set = \bigcup_{i=2}^{n} Q_{C_i}$
  end
  Classifier.Incremental.Update(Training_set);
  $Q_{C_i} = Q_{C_i}(x_h, y_h)$;
  Return G_mean, F_Measure, κ, Model;
end

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 rational 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 in order to limit the overhead associated with detection. That is, our aim is 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, i.e. $n = 1$.

Figure 4 shows our results against a radial basis function (RBF) data stream (Bifet et al., 2010), one 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 takes place, we signal a warning for potential drift and start to collect all instances from this point of

time into the drift detector queues. Next, we test for drift when we reach (L/n) instances and proceed accordingly. That is, we either detect a drift (and reset the learner) or continue to monitor. If no drift has been detected but the warning level remains, we proceed to collect and to test with the next (L/n) instances. This process continues until the set of examples is equal to our queue size L. Throughout this process, in the scenario 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, as 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. The next section details our experimental evaluation.

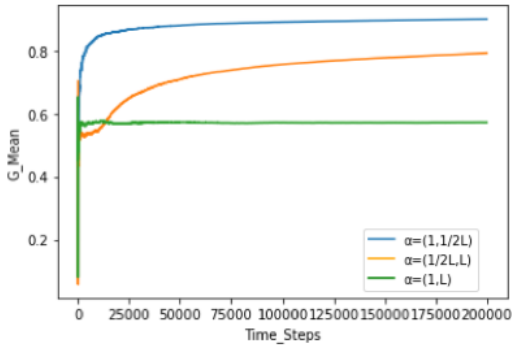

Figure 4: G-mean value with different thresholds against the RBF dataset.

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 *Name Withheld* Cloud with 10 Core CPUs. Our code was implemented using the Scikit-Learn (Pedregosa et al., 2011) and Scikit-Multiflow (Montiel et al., 2018) packages in Python version 3.8.2. The framework's implementation and all the code for the experiments will be made available in 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 (Montiel et al., 2018). Additionally, we employed three classifiers, namely, Hoeffding Adaptive Tree (HAT) (Bifet and Gavalda, 2009), Hoeffding Tree (HT) (Domingos and Hulten, 2000), and the Self-Adjusting Memory (SAM) model for the K-nearest neighbor (KNN), denoted by SAMKNN (Losing et al., 2017). 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.

The performance measures we used are the F-measure, geometric mean (G-mean), and Cohen's κ statistic. The F-measure refers to the harmonic mean of two metrics, recall and precision. The F-measure may be weighted depending on the value assigned to α. We used

a balanced value, which is 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. The F-measure is macro-averaged over the sum of F1-score over all classes which assigns equal weights to the existing classes. Additionally, we employed the G-mean to separately consider the classifier's accuracy rate on each of the classes and calculate the geometric mean of the classes. This measure maximizes the accuracy for each of the classes while keeping these accuracies balanced. Moreover, Bifet and Frank in (Bifet and Frank, 2010) proposed the κ statistic for online learning; this statistic is used to address the effect of predicting a label by chance when calculating a classifier's accuracy in a streaming setting (Montiel et al., 2018).

We conducted two sets of experiments to assess our OMCQ algorithm. First, we considered the performance of our algorithm when utilizing the baseline and component classifiers. Second, we compared OMCQ to the previously mentioned state-of-the-art MOOB and MUOB algorithms. We chose these algorithms for our comparison because they are directly applicable to multi-class streams. MOOB and MUOB are implemented using the parameters detailed in (Wang et al., 2016). Note that the algorithms do not handle concept drift. In order to conduct a fair comparison, we incorporated the DDM method with MOOB and MUOB in a manner similar to that of the OMCQ algorithm. For each class k, two values (p_i and s_i) will update if the new sample x_i belongs to the class c_k, where p_i is the probability of misclassifying and $s_i = \sqrt{\frac{p_i(1-p_i)}{i}}$ is the standard deviation of p_i. A significant increase in the error rate of each class signals the drift detector. Throughout online learning, we calculate $(p_i + s_i > p_{\min} + \lambda s_{\min})$ where $\lambda = 2$ and $\lambda = 3$ correspond to w_i and d_i, respectively. If a drift is detected, the model constructed by MOOB or MUOB is reset, and a new model is learned using the examples stored since the warning trigger.

4.1. Datasets

Our experimental study was based on the following multi-class datasets as depicted in Table 1: historical weather data obtained from Open Data Canada (Government, 2020), the Shuttle dataset from KEEL (Alcala et al., 2011), the LED dataset (Bifet et al., 2010), the RBF stream (Bifet et al., 2010), the Gas Sensor dataset (Vergara et al., 2012), the Human Activity Recognition (HAR) stream (Casale et al., 2012) and the Covertype dataset (Frank and Asuncion, 2010). The weather dataset contains data from probes located across Canada to detect adverse weather. The Shuttle dataset 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 dataset comprises of 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 concept drifts in the datasets by simply changing the attribute positions. The radial basis function (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 datasets have ten numerical attributes and 50 centers with four classes. The Gas Sensor dataset contains 13610 measurements from 16 chemical sensors utilised in simulations for drift compensation in a discrimination task of six gases at various levels of concentrations. The HAR dataset contains uncalibrated accelerometer data collected from 15 participants

performing seven activities. We combined activity of three participants to create drift in the dataset. Covertype is a benchmark dataset for evaluating stream classifiers. This dataset represents a forest cover type for 30 × 30m cells, where each cover type is represented by one of the seven classes.

Table 1: Data streams and their properties.

Dataset	Size	Number of Class	Class Imbalance Ratio
Weather	29375	4	1:2.5:1.5:13
Shuttle	2167	3	1:5:13
LED	7205	4	1:1.5:2.7:5.7
RBF	200000	4	1:1:1:2
Gas Sensor	13610	4	1:1:1:4.4
HAT	35300	4	1:1:1.7:6.3
Cover Type	50000	7	1:1:1:1:1:5:13

4.2. Experimental results

First, we investigated the effect of queue size on our OMCQ algorithm to assess how the size of L affects the performance of queue-based learning. Figure 5 depicts the behavior of the proposed method on different sizes of queue $L \in \{1, 10, 20, 30, 50\}$. As expected, the figure shows that the smaller the queue length, the faster the learning speed. Q_1 dominated in the first 500 time steps; however, at the end of training it reached a G-mean similar to that of Q_{10} and Q_{20}. In online learning, there is an obvious interplay between accuracy and learning time. Our results indicate that, for our experiments, a queue size of 20 resulted in a good trade-off between accuracy and speed, and we subsequently report the results of using this queue size against all datasets. It follows that queue sizes are highly domain-dependent; this value was set by inspection.

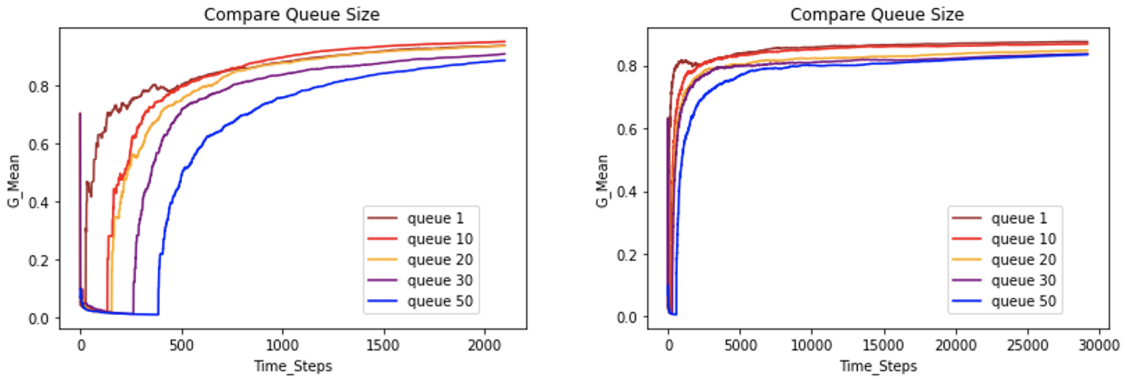

Figure 5: Results against different queue sizes (RBF stream).

Table 2 depicts the G-mean results for all four datasets when assessing the performance of our OMCQ algorithm with various base learners compared to the two baseline algorithms.

The results clearly show the benefit of our method compared to the majority class and no-change learners, which were not able to learn the concepts within our multi-class imbalanced streams. The other algorithms performed comparably. Readers should notice that fast drift detection, together with the associated recovery process, prevent a significant performance drop in our results.

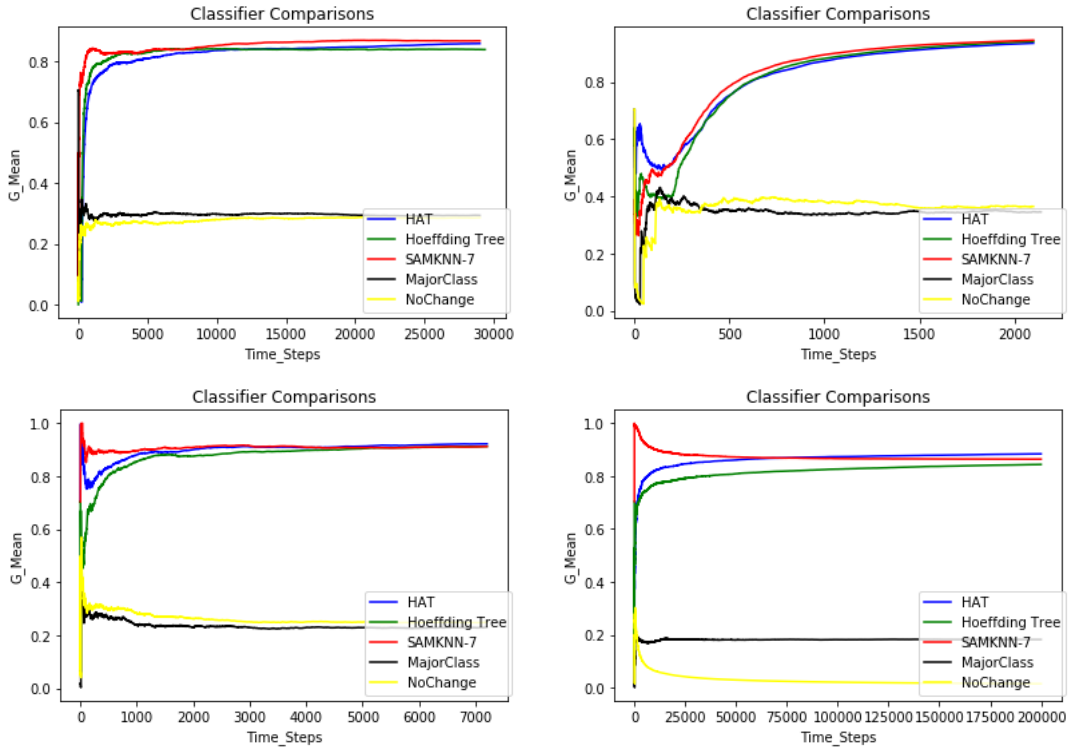

Table 2: Results of model construction for different learners.

Table 3 presents the results of our comparative study when contrasting the OMCQ, MOOB, and MUOB algorithms. The three component classifiers (HT, HAT and SAMKNN) had comparable results across all data sets. In addition, the OMCQ algorithm produced the highest values in terms of G-mean for all data streams and for all component classifiers. The same observation holds for the F-measure, where again, OMCQ produced higher results. Notably, this was especially the case for the LED, RBF, Gas Sensor and Covertype data streams, which contained concept drift, where the MOOB and MUOB algorithms struggled to obtain high values for these two metrics. Regarding the κ statistic, the three algorithms yielded comparable values against the weather, Shuttle, and LED streams. For the RBF dataset, OMCQ used together with HAT and HT produced significantly higher values, while the MOOB algorithm using SAMKNN-7 as base-line gave the best results, with OMCQ as a close second. In the case of the Gas Sensor, HAR and Covertype datasets, OMCQ achieved significantly higher results when utilising SAMKNN-7. These results indicate that the queuing mechanism of OMCQ, where we consider all original instances rather than resampling, clearly benefits online learning.

Table 3: Evaluation results against data streams.

		OMCQ			MOOB			MUOB		
Stream	**Classifier**	G-mean	F-score	κ	G-mean	F-score	κ	G-mean	F-score	κ
Shuttle	HAT	**0.950**	**0.945**	0.915	0.867	0.833	**0.930**	0.826	0.796	0.891
	HT-Tree	**0.951**	**0.942**	**0.922**	0.838	0.815	0.883	0.800	0.764	0.882
	SAMKNN-7	**0.968**	**0.964**	0.943	0.896	0.879	**0.960**	0.845	0.723	0.931
Weather	HAT	**0.854**	**0.838**	0.780	0.785	0.733	**0.785**	0.790	0.750	0.774
	HT-Tree	**0.857**	**0.831**	0.771	0.791	0.752	**0.825**	0.787	0.749	0.801
	SAMKNN-7	**0.880**	**0.868**	0.813	0.839	0.786	**0.872**	0.771	0.694	0.851
LED	HAT	**0.921**	**0.902**	**0.876**	0.784	0.756	0.850	0.789	0.761	0.854
	HT-Tree	**0.912**	**0.888**	**0.860**	0.780	0.747	0.855	0.780	0.747	0.855
	SAMKNN-7	**0.911**	**0.893**	0.866	0.767	0.699	0.890	0.792	0.754	**0.895**
RBF	HAT	**0.886**	**0.854**	**0.801**	0.620	0.478	0.460	0.645	0.494	0.525
	HT-Tree	**0.844**	**0.819**	**0.785**	0.701	0.590	0.650	0.617	0.476	0.575
	SAMKNN-7	**0.861**	**0.841**	0.810	0.777	0.721	**0.865**	0.643	0.547	0.822
Gas Sensor	HAT	**0.846**	**0.826**	**0.756**	0.619	0.640	0.686	0.640	0.544	0.400
	HT-Tree	**0.982**	**0.963**	**0.949**	0.740	0.587	0.711	0.550	0.414	0.495
	SAMKNN-7	**0.975**	**0.965**	**0.954**	0.964	0.910	0.899	0.857	0.837	0.769
HAR	HAT	**0.839**	**0.745**	0.684	0.806	0.674	0.681	0.796	0.640	**0.687**
	HT-Tree	0.854	**0.781**	0.701	**0.859**	0.743	**0.755**	0.812	0.652	0.712
	SAMKNN-7	**0.883**	**0.817**	**0.784**	0.839	0.704	0.783	0.767	0.655	0.664
Cover Type	HAT	**0.844**	**0.728**	**0.714**	0.835	0.715	0.705	0.772	0.661	0.600
	HT-Tree	**0.888**	**0.844**	**0.811**	0.777	0.661	0.698	0.662	0.387	0.591
	SAMKNN-7	**0.909**	**0.846**	**0.837**	0.818	0.682	0.680	0.800	0.641	0.623

5. Conclusion

The paper addressed the challenge of online learning from evolving multi-class imbalanced data streams, which are susceptible to concept drifts. An advantage of our OMCQ method is that it operates independent of a base classifier, thus providing a general framework for dealing with evolving multi-class streams. The OMCQ algorithm maintains queues for each of the classes and thus implicitly balances the data, without having to resort to resampling. In this way, all classes are equally represented during online learning. Concept drift is handled by considering individual classes, which implies that drifts in minority classes are detected as soon as enough instances have been collected. Our experimental results show the benefits of our approach, and we determined that the OMCQ method is a promising candidate for further experimentation.

Our OMCQ algorithm is highly suitable for scenario where the minority class instances are numerous enough to fill a queue. We plan to also study highly skewed multi-class scenarios where minority classes are very scarce or may only arrive after long delays. In addition, we aim to investigate the trade-offs between performance and time complexity. Moreover, we plan to conduct ablation studies to explore the influence of queue sizes on the learning outcomes. Further, a detailed investigation of the interplay between multi-class

imbalanced learning and concept drift detection is needed. Future work will also include a comparison with other resampling methods, such as SCUT-DS (Olaitan and Viktor, 2018), an approach for stationary streams that combines over-sampling based on SMOTE with cluster-based undersampling, and SOUP, a Bagging ensemble that uses the notion of safe levels to resample data in an offline setting (Janicka et al., 2019).

References

J. Alcala, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, and F. Herrera. Keel data mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *In Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3): 255–287, 2011.

A. Bifet and E. Frank. Sentiment knowledge discovery in Twitter streaming data. *In International Conference on Discovery Science*, page 1–15, 2010.

A. Bifet and R. Gavalda. Adaptive parameter-free learning from evolving data streams. *In International Symposium on Intelligent Data Analysis*, pages 249–260, 2009.

A. Bifet, G. Holmes, B. Pfahringer, P.Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive online analysis, a framework for stream classification and clustering. *In Proceedings of the First Workshop on Applications of Pattern Analysis*, pages 44–50, 2010.

P. Casale, O. Pujol, and P. Radeva. Using information on class interrelations to improve classification of multi-class imbalanced data. *In: Personal and Ubiquitous Computing*, 16(5):563–580, 2012.

P. Domingos and G. Hulten. Mining high-speed data streams. *In Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

A. Fernandez, S. Garcia, M. Galar, R.C. Prati, B. Krawczyk, and F. Herrera. Learning from imbalanced data stream. *In Learning from Imbalanced Data Sets*, page 279–303, 2018.

A. Fernández, V. Lopez, M. Galar, M.J. Del Jesus, and F. Herrera. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *In Knowledge-Based Systems*, 42:97–110, 2013.

A. Frank and A. Asuncion. UCI machine learning repository, University of California, Irvine. 2010. URL "http://archive.ics.uci.edu/ml".

H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *In ACM SIGKDD Explorations Newsletter*, 21(2):6–22, 2019.

Canada Government. Historic climate data from Environment and Climate Change Canada, 2020. URL https://climate.weather.gc.ca/historical_data/search_historic_data_e.html.

M. Janicka, M. Lango, and J. Stefanowski. Using information on class interrelations to improve classification of multi-class imbalanced data: a new re-sampling algorithm. *In International Journal of Applied Mathematics and Computer Science*, 29(4):769–781, 2019.

V. Losing, B. Hammer, and H. Wersing. Self-adjusting memory: How to deal with diverse drift types. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 4899–4903, 2017.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

K. Malialis, C. Panayiotou, and M.M. Polycarpou. Queue-based resampling for online class imbalance learning. *In International Conference on Artificial Neural Networks*, page 498–507, 2018.

J. Montiel, J. Read, A. Bifet, and T. Abdessalem. Scikit-multiflow: A multi-output streaming framework. *The Journal of Machine Learning Research*, 19(72):1–5, 2018.

O.M. Olaitan and H.L. Viktor. SCUT-DS: Learning from Multi-class imbalanced Canadian weather data. *In International Symposium on Methodologies for Intelligent Systems*, pages 291–301, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

B. Read, A. Bifet, B. Pfahringer, and G. Holmes. Batch-incremental versus instance-incremental learning in dynamic and evolving data. *In International Symposium on Intelligent Data Analysis*, pages 313–323, 2012.

A. Vergara, S. Vembu, T. Ayhan, M.A Ryan, M.L. Homer, and R. Huerta. Chemical gas sensor drift compensation using classifier ensembles. *In Sensors and Actuators B—Chem*, 166:320–329, 2012.

S. Wang, L.L. Minku, D. Ghezzi, D. Caltabiano, P. Tino, and X. Yao. Concept Drift Detection for Online Class Imbalance Learning. *In International Joint Conference on Neural Networks (IJCNN '13)*, pages 1–10, 2013.

S. Wang, L.L. Minku, and X. Yao. Resampling-based ensemble methods for online class imbalance learning. *In IEEE Transactions on Knowledge and Data Engineering*, 27(5): 1356–1368, 2014.

S. Wang, L.L. Minku, and X. Yao. Dealing with multiple classes in online class imbalance learning. *In International Joint Conferences on Artificial Intelligence*, pages 2118–2124, 2016.