VFC-SMOTE: very fast continuous synthetic minority oversampling for evolving data streams

Alessio Bernardo · Emanuele Della Valle

Received: 21 November 2020 / Accepted: 21 July 2021 / Published online: 14 September 2021
© The Author(s) 2021

Abstract
The world is constantly changing, and so are the massive amount of data produced. However, only a few studies deal with online class imbalance learning that combines the challenges of class-imbalanced data streams and concept drift. In this paper, we propose the very fast continuous synthetic minority oversampling technique (VFC-SMOTE). It is a novel meta-strategy to be prepended to any streaming machine learning classification algorithm aiming at oversampling the minority class using a new version of SMOTE and BORDERLINE-SMOTE inspired by Data Sketching. We benchmarked VFC-SMOTE pipelines on synthetic and real data streams containing different concept drifts, imbalance levels, and class distributions. We bring statistical evidence that VFC-SMOTE pipelines learn models whose minority class performances are better than state-of-the-art. Moreover, we analyze the time/memory consumption and the concept drift recovery speed.

Keywords SML · Evolving data stream · Concept drift · Balancing · Data sketching

1 Introduction

Data abound as a multitude of smart devices produce massive, continuous, unbounded and non-stationary flows of data, namely data streams. Data streams are different from the batches of data used to train traditional Machine Learning (ML) models. In case of batches, since all the observations are known in advance, it is possible to iterate over them multiple times or to split them into training and testing sets or
inspecting their characteristics, e.g. class imbalance ratio. Instead, in case of data streams, new samples arrive unceasingly over time as mini-batches or even only one at a time. Therefore, it is impossible to iterate over data streams multiple times or to split them into training and testing sets or to inspect their characteristics. Hence, also the traditional/batch-oriented ML techniques cannot be used.

In the 2000s’ (Domingos and Hulten 2000; Hulten et al. 2001), data streams have been recognized as a challenge and the interest in them is growing steadily. ML practitioners often transform streams into sequences of batches and retrain models as new batches become available. However, retraining often a model may become expensive. In recent years, Streaming Machine Learning (SML) (Bifet et al. 2010) was introduced to address this challenge. SML can maintain models online, incorporating one sample at a time and continuously updating the model instead of retraining it anew. To cope with the stream unboundedness, each sample once used is discarded. Moreover, streams can evolve and present forms of non-stationarity (namely, concept drift (Tsymbal 2004)). A concept drift occurs when the function which generates instances at time step \( t \) is not the same as at time step \( t + 1 \). Therefore, a SML model is dynamic and adaptive. Streams can present a class imbalance situation, too. Most of streaming binary classification solutions neglect the minority class instances, preventing the discovery of any existing patterns in them (He and Garcia 2009). Moreover, due to the concept drift occurrence, classes may swap, i.e. all the samples labeled as minority (majority) class before the concept drift would get labeled as majority (minority) class after it.

The combined problems of concept drifts and class imbalance are found in many real-world applications. In social media mining, classifying all the news according to topics involves both concept drift (new topics frequently appear, outdated ones are forgotten with time, and old ones may become popular again) and class imbalance (some topics are more popular than others). Such phenomena can also be observed in product recommendations, since the interests of clients may change over time and some products could be more popular, and so purchased, than others (Ma et al. 2007). Another application is the credit card fraud detection (Pozzolo et al. 2018). The task is to classify if a credit card transaction is fraudulent or not and it involves both concept drift (customers’ habits evolve and fraudsters change their strategies over time) and class imbalance (genuine transactions far outnumber frauds).

To address these problems, techniques to rebalance the training dataset were proposed in the batch scenario. Two of the most used (He and Garcia 2009) are the Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al. 2002) and the Borderline Synthetic Minority Over-sampling Technique (BORDERLINE-SMOTE) (Han et al. 2005). They need the entire batch as input to enable rebalancing but, in the streaming approach, this batch is not available. The main contribution of this paper is the Very Fast Continuous SMOTE (VFC-SMOTE). It is a meta-strategy inspired by SMOTE, BORDERLINE-SMOTE and Data Sketching (Cormode 2017) and, besides improving the classification performances w.r.t. the state-of-the-art methods, it focuses on reducing the time and memory consumption.

VFC-SMOTE can be prepended to any SML model as a sort of magnifying glass that enlarges the poorly represented portion of the stream (minority class instances). Since we are proposing a meta-strategy as a sort of pre-processing step, we expect
it to be data dependent as some other meta-strategies, i.e. temporally augmentations. This is the reason why we are looking for at least one SML algorithm prepended by VFC-SMOTE that outperforms the state-of-the-art ones and not for an algorithm able to outperform all the others.

In investigating VFC-SMOTE, considering that there are SML algorithms *natively able* to rebalance streams in presence of concept drift (say, SML+) and algorithms *unable* to do so (say, SML-), we formulated the following research questions:

Q1 *Comparing VFC-SMOTE to SML+ models, is there at least one SML- model that, prepended by VFC-SMOTE, outperforms any SML+ algorithm presented in literature?*

Q2 *Is VFC-SMOTE consuming less time/memory than the other SML+ techniques?*

Q3 *Is VFC-SMOTE altering the SML- models recovery from a concept drift?*

In more details, the main contributions of this paper are:

- **VFC-SMOTE**, a meta-strategy inspired by both the well-known SMOTE and BORDERLINE-SMOTE techniques (applicable only to batches) and by Dynamic Sketching that can be prepended to any SML- classifier;
- Statistical evidence that VFC-SMOTE can outperform the SML+ methods in terms of F1, Recall and G-mean obtained analysing the results of five SML- classifiers prepended by VFC-SMOTE and four other SML+ strategies (i.e., positively answering to Q1);
- A computational costs analysis of SML- algorithms prepended by VFC-SMOTE w.r.t. other SML+ strategies in the literature, showing that the latter take more time and consume more memory than the former (i.e., positively answering also to Q2); and
- A discussion about the VFC-SMOTE recovery speed analysis from concept drift showing that VFC-SMOTE does not alter the recovery speed of the SML- model to which it is prepended (i.e., negatively answering to Q3).

The remainder of this paper is organized as follows. Section 2 introduces the imbalance problem, some rebalance techniques and some SML+ models. Section 3 describes VFC-SMOTE. Section 4 introduces the streams and SML- models used in the experiments and presents our research hypotheses. Section 5 shows and discusses the evaluation results. Section 6 discusses the VFC-SMOTE limitations, while Sect. 7 discusses the conclusions and some future researches.

## 2 Related work

The first part of this section introduces the class imbalance problem and some sampling techniques able to deal with it, while the second part deals with some state-of-the-art approaches able to learn from imbalanced data streams and to handle concept drift changes.

**Sampling techniques for class imbalance** An unequal distribution among classes characterizes imbalanced data. Since the instances contained in the minority class(es) rarely occur, the patterns for classifying these classes tend to be rare, undiscovered, or ignored.
He and Garcia (2009) characterize the approaches to handle class imbalance as: sampling techniques, cost-sensitive learning, kernel-based methods, and active learning methods. This work focuses on sampling techniques because they allow creating a new meta-strategy that can be run during the pre-processing phase, regardless of the streaming method chosen. Sampling techniques change the data distribution so that standard algorithms focus on the cases that are more relevant to the user.

They are divided in **oversampling** and **undersampling** methods. **Oversampling** methods increase the number of minority class instances through the creation of synthetic instances, until classes are balanced. After that, the minority class, which was originally underrepresented, may exert a greater influence on learning and future predictions. **Undersampling** methods, on the other hand, aim at reducing the number of instances from the majority class by removing instances from this class. They often act by removing noisy instances, or simply reducing instances randomly or by means of some heuristics. Both methods introduce their own set of drawbacks that can worsen the learning phase (He and Garcia 2009). In case of undersampling, removing instances from the majority class may cause important concept loss. In case of oversampling, since data are replicated or synthetically generated, the drawback is that multiple iterations over the same instances can result in overfitting.

**SMOTE** (Chawla et al. 2002) is a popular oversampling balancing technique (see Fig. 1a) that synthetically generates instances for the minority class to balance the training data. For each minority class sample \( x_i \) (orange triangle), SMOTE finds its \( K \) nearest neighbours among the other minority class samples, it randomly chooses one \( \hat{x}_i \) from them, and its distance from \( x_i \) is multiplied by a random number \( \delta \in [0, 1] \). The resulting new sample \( x_n \) (black circle) is located between \( x_i \) and the selected neighbor \( \hat{x}_i \). In general SMOTE has been shown to improve classification, but it may also suffer from drawbacks related to the way it creates synthetic samples. Specifically, SMOTE generates new samples without considering the neighbour examples, which increases the occurrence of overlapping between classes.

To this end, **BORDERLINE-SMOTE** (Han et al. 2005) (see Fig. 1b), instead of using all the minority class samples to generate new instances, uses only the borderline samples. For each minority class sample \( x_i \) (orange triangle), BORDERLINE-SMOTE finds its \( K \)-nearest neighbours among the whole samples. If all of them are majority class samples (case A), \( x_i \) is considered noise. If there are more majority class neighbours than minority class ones (case B), \( x_i \) is considered easily misclassified and put into
a danger set. Otherwise, if there are more minority class neighbours than majority class ones (case C), $x_i$ is considered safe. Only the samples in the danger set are the borderline ones and they are used by SMOTE to generate new instances (black circles).

More than a hundred SMOTE variants have been proposed (Fernández et al. 2018) to overcome the overlapping between classes. The most well-known are ADASYN (He et al. 2008), DBSMOTE (Bunkhumpornpat et al. 2012), MDO (Abdi and Hashemi 2015), SWIM (Bellinger et al. 2020), and G-SMOTE (Douzas and Bação 2019). Due to space limitations, we provide their explanations in Appendix A.

For our proposed method we decided to use SMOTE because it is still considered the "de-facto" standard in the framework of learning from imbalanced data (Fernández et al. 2018). There is also the possibility to use BORDERLINE-SMOTE, with the aim to use less data to generate new synthetic instances, thus reducing the time and memory consumed.

Still another problem remains. As a sampling technique, SMOTE and BORDERLINE-SMOTE cache the entire dataset in memory. This approach goes against the basic principles of the data stream paradigm that inspects only a sample at a time, as fast as possible, and then discards it. In Sect. 3, we explain how we overcome this problem.

State-of-the-Art Approaches The first part of the section wraps up the various methods, while the remainder of the section describes the methods summarized in Table 1.

They are commonly categorized into two major groups: passive versus active approaches, depending on whether an explicit drift detection mechanism is employed. Passive approaches train a model continuously without an explicit trigger reporting the drift, while active approaches determine whether a drift has occurred before taking any actions. Examples of passive approaches are RLSACP, ONN, ESOS-ELM, an ensemble of neural networks (NN), OnlineUnderOverBagging, OnlineSMOTEBagging, OnlineAdaC2, OnlineCSB2, OnlineRUSBoost and OnlineSMOTEBest, while ARFRE, RebalanceStream, C-SMOTE, OOB, UOB, WEOB1 and WEOB2 are considered active approaches.

Passive approaches RLSACP (Ghazikhani et al. 2013b) is inspired by the recursive least square (RLS) filter error model. In the proposed error model, non-stationarity is handled including the forgetting factor ($k$) present in the RLS error model, while for handling class imbalance, two adaptive error weighting strategies are proposed. In the first one, error weights are adapted based on classifier results in different classes. In the second one, the number of instances in the minority and majority classes are counted from a fixed window of the most recent samples and the weights are assigned accordingly.

ONN (Ghazikhani et al. 2014) is a similar approach. It is an online Multi Layer Perceptron model composed by two parts. The first is a forgetting function for handling concept drift while the second is an error weighting function for handling class imbalance. EONN (Ghazikhani et al. 2013a) is an online ensemble neural network model composed by two layers. The first layer is a cost-sensitive neural network for handling class imbalance, while the second layer contains a method for weighting classifiers of the ensemble. Another passive technique is ESOS-ELM (Mirza et al. 2015). It is an ensemble approach that, to tackle class imbalance, resamples the data using fixed weights to train each classifier with an approximately equal number of
| Method                                      | Classifier type | Approach type | Approach for CD     | Approach for Class Imbalance |
|---------------------------------------------|-----------------|---------------|---------------------|------------------------------|
| RLSACP (Ghazikhani et al. 2013b)            | Single          | Passive       | Forgetting factor   | Cost weight                  |
| ONN (Ghazikhani et al. 2014)                | Ensemble        | Passive +     | Forgetting factor   | Cost weight                  |
| EONN (Ghazikhani et al. 2013a)              | Ensemble        | Passive       | Weighted ensemble   | Cost weight                  |
| ESOS-ELM (Mirza et al. 2015)                | Ensemble        | Passive       | Weighted ensemble   | Cost weight                  |
| OnlineUnderOverBagging (Wang and Pineau 2016) | Ensemble      | Passive       | Weighted ensemble   | Undersampling + Oversampling  |
| OnlineSMOTEBagging (Wang and Pineau 2016)   | Ensemble        | Passive       | Weighted ensemble   | SMOTE                        |
| OnlineAdaC2 (Wang and Pineau 2016)          | Ensemble        | Passive       | Weighted ensemble   | Cost weight                  |
| OnlineCSB2 (Wang and Pineau 2016)           | Ensemble        | Passive       | Weighted ensemble   | Cost weight                  |
| OnlineRUSBoost (Wang and Pineau 2016)       | Ensemble        | Passive       | Weighted ensemble   | Undersampling                |
| OnlineSMOTEBoost (Wang and Pineau 2016)     | Ensemble        | Passive       | Weighted ensemble   | SMOTE                        |
| ARFRE (Ferreira et al. 2019)                | Ensemble        | Active        | ADWIN               | Cost weight                  |
| RB (Bernardo et al. 2020a)                  | Multiple (4)    | Active        | ADWIN               | SMOTE                        |
| C-SMOTE (Bernardo et al. 2020b)             | Meta-strategy   | Left to algorithm | Left to algorithm | SMOTE variant + ADWIN        |
| OOB (Wang et al. 2013)                      | Ensemble        | Active        | Weighted ensemble   | Oversampling + Cost weight   |
| UOB (Wang et al. 2013)                      | Ensemble        | Active        | Weighted ensemble   | Undersampling + Cost weight  |
| WEOB1/WEOB2 (Wang et al. 2015)              | Ensemble        | Active        | Weighted ensemble   | Cost weight                  |
| VFC-SMOTE                                   | Meta-strategy   | Left to algorithm | Left to algorithm | SMOTE\Borderline-Smote variant + ADWIN |
majority and minority class samples. Instead, to tackle concept drifts, it uses a voting weights system according to the G-mean performance metric. ESOS-ELM also has an active module to handle recurring concept drifts.

OnlineUnderOverBagging, OnlineSMOTEBagging, OnlineAdaC2, OnlineCSB2, OnlineRUSBoost and OnlineSMOTEBoost (Wang and Pineau 2016) are the online extensions of the popular batch cost-sensitive ensemble learning algorithms Under-OverBagging, SMOTEBagging, AdaC2, CSB2, RUSBoost and SMOTEBoost, respectively. The main challenge for adapting them to the online settings resides in finding a way to embed costs into online ensembles for boosting algorithms without having all the data. They reformulate the batch cost-sensitive boosting algorithms avoiding the normalization step at each iteration, and then to incrementally estimate the quantities embedded with the cost setting in the online learning scenario. Whereas cost sensitivity in the batch setting is achieved by different resampling mechanisms. In the online ensembles this is achieved by manipulating the parameters of the Poisson distribution for different classes.

**Active approaches** ARFRE (Ferreira et al. 2019) is an extension of the ARF (Gomes et al. 2019) algorithm. ARFRE resamples instances based on the current class label distribution, so that it adapts the weights of the Poisson distribution to simulate a balance of the instances to the base models of the forest. Thus, if a sample is part of the minority class, its weight used during the training phase will be increased w.r.t a sample from the majority class.

RebalanceStream (RB) (Bernardo et al. 2020a) uses ADWIN (Bifet and Gavaldà 2007) to detect concept drift in the stream by training four models \(m_1, m_2, m_3,\) and \(m_4\) in parallel: \(m_1\) is trained with the original samples in input; \(m_2\) uses the samples collected and rebalanced using SMOTE from the beginning to a change (when the last concept drift occurred); \(m_3\) uses the samples collected from a warning (when the most recent concept drift started to occur) to a change; \(m_4\) uses the same data of \(m_3\) but rebalanced using SMOTE. After a change, the best model among them is chosen and it is used to proceed with the execution.

C-SMOTE (Bernardo et al. 2020b) is a meta-strategy similar to VFC-SMOTE, designed to be prepended to any SML- techniques. It is inspired by SMOTE. It uses ADWIN (Bifet and Gavaldà 2007) to save only the recently-seen samples and uses the minority class samples stored in the ADWIN window to apply SMOTE. C-SMOTE, saving all the entire instances in a window and not only a summary of them as VFC-SMOTE does, focuses only on improving the classification performances, neglecting the computational ones.

Oversampling-based Online Bagging (OOB) and undersampling-based Online Bagging (UOB) (Wang et al. 2013) are other two resampling-based ensemble methods. When class imbalance is detected, oversampling or undersampling embedded in Online Bagging (Oza 2005) is triggered to either increase the chance of training minority class samples or reduce the chance of training majority class samples. If the new sample \((x, c_k)\) belongs to one of the minority classes \((c_k \in Y_{\text{min}})\), OOB will tune the parameter \(\lambda\) of Poisson distribution to \(1/w_k\), which indirectly increases the number of copies of the current sample for training. In other words, it combines oversampling with Online Bagging. If \((x, c_k)\) belongs to one of the majority classes \((c_k \in Y_{\text{maj}})\), UOB will set \(\lambda\) to \((1 - w_k)\). Training samples from the major class
will be undersampled. Some performance analysis (Wang et al. 2015) on OOB and UOB show that UOB is a better choice than OOB in terms of minority-class Recall and G-mean. However, it has some weaknesses when the majority class in the data stream turns into the minority class. OOB is more robust against changes.

To combine the strength of OOB and UOB, WEOB1 and WEOB2 (Wang et al. 2015) are proposed. They are based on the idea of ensembles, which train and maintain both OOB and UOB. A weight is maintained for each of them, adjusted adaptively according to their current G-mean performance. The weight evaluates the degree of inductive bias in terms of a ratio of positive and negative accuracy. Their combined weighted rating will decide the final prediction. WEOB1 and WEOB2 differ in the weight adjusting strategy that they use.

3 VFC-SMOTE

This section describes the proposed meta-strategy VFC-SMOTE. Its acronym stands for Very Fast Continuous SMOTE because new variants of SMOTE and BORDERLINE-SMOTE, to which it is inspired by, are applied continuously, trying to consume as little time and memory as possible. VFC-SMOTE is designed to rebalance an imbalanced data stream and it can be prepended to any SML- models. The section firstly introduces the concepts underlying VFC-SMOTE and then it explains in detail the algorithm.

Algorithm concepts As said before, the real problem is the impossibility to access the entire data during the pre-processing (rebalancing) phase. Indeed, it is impossible to store every new sample in memory until the stream ends for two reasons: 1) the streams are assumed infinite, and 2) this would be against the stream paradigm approach. Our solution (see Fig. 2) is inspired by Data Sketching (Cormode 2017). In the Data Stream Management System context, Frequency Based Sketches are used to summarize the observed frequency distribution of a dataset. From these sketches, accurate estimations of individual frequencies can be extracted (Cormode 2017). This concept of using statistical summaries for data streams is well-known in the stream
clustering context, too (Li et al. 2008; Kranen et al. 2011). Intuitively, our proposal uses a dynamic summary data structure, called sketch, to maintain an approximation of the attribute distribution seen so far on the underlying continuous stream and then uses the sketch to generate new synthetic instances. VFC-SMOTE uses an incremental decision tree, in particular a Hoeffding Adaptive Tree (HAT) (Bifet and Gavaldà 2009), as a data structure to memorize the sketch with. HAT is based on Hoeffding Tree (HT) (Domingos and Hulten 2000), an incremental very fast decision tree algorithm able to learn from non-evolving streaming data. To make HT able to adapt to concept drifts, HAT implements, at each node, an ADWIN change detector (Bifet and Gavaldà 2007). ADWIN keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis “there has been no change in the average value inside the window”. More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. Moreover, automatically detecting and adapting to the current rate of change, ADWIN is parameter and assumption-free. Its only parameter is the confidence bound \( \delta \), indicating how confident we want to be about the algorithm’s output, referring to all algorithms dealing with random processes. In this way, each node can monitor the performances of its sub-branches and, if a concept drift occurs, it can substitute them with new sub-branches. So, VFC-SMOTE is always applied on data that are consistent with the current concept, and the new synthetically generated samples will be consistent as well. We chose the HAT model as sketch since it is one of the most used models and it is a state-of-the-art method, while, about ADWIN, it is the most used now in MOA learners,\(^1\) it is cited in different papers (Lu et al. 2020; Grulich et al. 2018; Bifet et al. 2018; Ga mae t al. 2014) and it is already implemented inside the HAT model. Another reason is that ADWIN has theoretical guarantees and it extends such guarantees to the sketch. The VFC-SMOTE memory and time (per sample) asymptotic costs are \( O(\log W) \), where \( W \) is the ADWIN window length. If the SML-model prepended by VFC-SMOTE uses ADWIN, then the total asymptotic costs do not change, otherwise they depend on whether the SML-model costs are more or less than the VFC-SMOTE ones. Appendix B shows the detailed cost analysis.

In particular, Fig. 3 shows a simple example about how the sketch works. Given a stream with two attributes, one nominal \((x_1)\) and one numerical \((x_2)\), the sketch incrementally grows a decision tree. Here, after having seen enough samples, the sketch decides to make a split on a \( \bar{x}_2 \) value of the \( x_2 \) attribute and in its leaves (summaries) it keeps the summary statistics of each attribute (summary). For example, in the left branch, the sketch keeps the summaries of all the samples having the \( x_2 \) value less than or equal to \( \bar{x}_2 \) (all the samples under the dotted line). In case of a nominal attribute \((x_1 \) summary), it keeps the occurrence frequency of each attribute value divided by class, while in case of a numerical attribute \((x_2 \) summary), it keeps the mean (\( \mu \)), variance (\( \sigma^2 \)), minimum (\( m \)) and maximum (\( M \)) values observed divided by class. Moreover, for each new sample in input, the sketch updates its summaries.

Through the hyperparameter \( \text{perCorrClass} \), users can decide how many instances to save into the sketch. Saving all the instances (correctly and incor-

\(^1\) https://moa.cms.waikato.ac.nz/.
directly classified) means using all of them during the rebalancing phase and thus using SMOTE, while saving only the misclassified ones means taking into account only the ones that lie on the border between the two classes instances and therefore using BORDERLINE-SMOTE. Of course, saving less instances (percCorrClass = 0) into the sketch could speed up the process and save memory unlike saving all of them (percCorrClass = 1). In the end, the new generated samples created by VFC-SMOTE, are used by the pipelined SML- algorithm to update its model.

Algorithm explanation Algorithm 1 presents VFC-SMOTE pseudo-code. Given a stream $S \{ (X_1, y_1), (X_2, y_2), \ldots \}$, where $X_i$ is a feature vector, and $y_i$ is the label, VFC-SMOTE sketches them into the variable sketch initialized with a HAT model (Line 2). VFC-SMOTE has also four counters (Line 3): $S_0$ and $S_1$, respectively, count the number of instances in each class; $S_0$ and $S_1$ count, respectively, the number of instances of each class generated by VFC-SMOTE. The reason why VFC-SMOTE keeps track of the instances of both classes is that, after a concept drift, the classes may swap. In this case, VFC-SMOTE will use the samples of the new minority class to introduce new synthetic samples. Every time a new sample $(X, y)$ is available, the prequential evaluation (Gama et al. 2013) approach is applied (testing and then learner training phase) (Line 6). Subsequently, depending on how the instance $(X, y)$ is classified, it can be added to sketch (Lines 7-13). If the sample is misclassified, then it is added directly to the sketch (Line 8). If it is not, it could be added anyway with a probability equal to the percCorrClass value (Lines 10-13). The function updateCounters() (Line 14) updates the relative counter $S_0$ or $S_1$, while the function checkDcInCounters() (Line 15) uses ADWIN to check a concept drift occurrence and to reset the $S_0$ and $S_1$ counters. The next step identifies the actual minority (Line 16) and majority (Line 17) classes and saves the corresponding minority/majority classes and counters into, respectively, minority, $S_{min}$, minority, $S_{min}$, majority, $S_{maj}$ and, $S_{maj}$. Then, Line 18 checks if the number of minority instances is at least equal to the hyperparameter minSizeMinority specified by the user. If it is true, the algorithm can proceed with the rebalance phase, otherwise it waits for another sample in input. If minSizeMinority is equal to $-1$, no checks are performed on the minority class.
Algorithm 1:

minSizeMinority: Minimum number of minority class instances to allow the rebalancement procedure
percCorrClas: Percentage of the correctly classified instances to save in the sketch
l: SML- learner
t: Balance ratio to be achieved
S: Binary classification data stream

1 Function VFC-SMOTE(minSizeMinority,percCorrClass,l,t,S):
2    sketch ← HA T(), cdDet ← ADWIN()
3    $S_0$, $S_1$, $S_0$, $S_1$, nCorrClass,nCorrSaved,imbalanceRatio ← 0
4    while hasNext(S) do
5        $X, y ← next(S)$
6        if predict(l,$X$) $\neq y$ then
7            sketch.add($X, y$)
8        else
9            nCorrClass ← nCorrClass + 1
10           if (nCorrClass × percCorrClass) > nCorrSaved then
11             nCorrSaved ← nCorrSaved + 1
12             sketch.add($X, y$)
13             updateCounters(sketch,$S_0$, $S_1$)
14             cdDet.checkCdInCounters(l,$X$, y,$S_0$, $S_1$)
15             minority,$S_{min}^{S_{min}}$, $S_{maj}^{S_{maj}}$ ← selectMinorityClass(sketch,$S_0$, $S_1$, $S_0$, $S_1$)
16             majority,$S_{maj}^{S_{maj}}$, $S_{maj}^{S_{maj}}$ ← selectMajorityClass(sketch,$S_0$, $S_1$, $S_0$, $S_1$)
17             if checkMinSize(minSizeMinority,$S_{min}$) then
18                imbalanceRatio ← ratio($S_{min}^{S_{min}}$, $S_{min}^{S_{min}}$, $S_{maj}$)
19                summaries ← getSummaries(sketch,minSizeMinority)
20                while t > imbalanceRatio do
21                   $\hat{X}, \hat{y} ← newSample(minority,summaries)$
22                   $S_{min}^{S_{min}} ← S_{min}^{S_{min}}+1$
23                   train($\hat{X}, \hat{y}$, l)
24                imbalanceRatio ← ratio($S_{min}^{S_{min}}$, $S_{maj}$,$S_{min}$,$S_{maj}$)
25            end
26    End Function

size, so the checkMinSize function will always return true. At Line 19, the actual imbalanceRatio between the number of minority and majority instances added to the sketch is calculated as in Eq. 1.

\[
\text{imbalanceRatio} = \frac{S_{min}^{S_{min}} + S_{min}^{S_{min}}}{S_{min}^{S_{min}} + S_{maj}^{S_{maj}} + S_{min}^{S_{min}} + S_{maj}^{S_{maj}}}
\]  

If the imbalanceRatio is less than the balance ratio to achieve t (imbalanced sketch), all the sketch summaries with more than minSizeMinority instances seen are retrieved (Line 20) and they are used to generate some new synthetic samples ($\hat{X}, \hat{y}$) until the imbalanceRatio is equal to t (balanced sketch, Lines 21-25). In particular, the function newSample(), at Line 22, and exploded in Algorithm 2, is responsible for generating a new synthetic sample.

This procedure is different from the original SMOTE and BORDERLINE-SMOTE ones. Before starting executing, SMOTE and BORDERLINE-SMOTE calculate the number of instances to be introduced for each minority sample in the batch. Instead, in our continuous version, not all the summaries are used to generate new instances. VFC-SMOTE uses the discrete probability function to have more chances to select
Algorithm 2:

minority: Minority class
summaries: The sketch summaries with more than minSizeMinority instances seen

Function newSample(minority,summaries):
values ← ∅, index ← 0
selectedSummaries ← DiscreteProbability(summaries,summaries.instancesSeen())
for each summary in selectedSummaries do
  if summary is numeric then
    μ ← summary.observedMean(minority)
    σ² ← summary.observedVariance(minority)
    m ← summary.observedMinValue(minority)
    M ← summary.observedMaxValue(minority)
    values[index] ← BetaDistribution(μ, σ²,m,M)
  else if summary is nominal then
    values[index] ← summary.mostFrequentlySeen(minority)
  index ← index + 1
values[index] ← minority
return value
End Function

the selectedSummaries that saw more instances (Line 3) and then, it uses the attributes observed summary contained into the selectedSummaries to generate a new instance (Lines 4-13). For each summary, a new value is generated for the new instance. In case of a numerical summary, the new value is a sample drawn from a Beta distribution having the mean (μ) and variance (σ²) observed from the minority class instances seen so far by that summary, scaled by the minimum (m) and maximum (M) values seen. In this way, the new sample value will certainly be between the m and M values (Lines 5-10). In case of a nominal summary, the new value is the most frequently seen value among the minority class instances seen so far by that summary (Lines 11-12). In the end, Line 14 assigns the minority class value to the new instance generated.

Back to Algorithm 1, at Line 23, $S_{\text{min}}$ is updated, the new instance $(\hat{X}, \hat{y})$ is used to train the learner $l$ (Line 24) and the new imbalanceRatio is calculated (Line 25).

4 Experimental settings

This section consists of six parts. At the beginning, we introduce the hypotheses to be tested. Then, the following four parts discuss the data stream (both synthetic and real), the SML- algorithms prepended by VFC-SMOTE to run the experiments, and the various experimental settings. The last part selects the best SML+ methods to be compared with VFC-SMOTE.

Research Hypotheses We formulate our hypotheses as follows:

– Hp. 1: VFC-SMOTE compared to the SML+ methods improves the performances for the minority class in at least one SML- algorithm prepended by.
– Hp. 2: Since VFC-SMOTE keeps a summary of the data in a sketch and introduces a rebalance phase, the SML- models to which it is prepended consume less memory and they have a lower latency for sample than the SML+ methods.
Hypothesis 3: Since VFC-SMOTE is only a meta-strategy to be prepended to any SML-model, aiming at improving the minority class performances, it has no impact on the concept drift management and therefore on the recovery speed from a concept drift occurrence.

**Artificial data stream** We decided to synthetically generate some data streams containing both different types of concept drift (Tsymbal 2004) and different class imbalance levels. We decided to start studying the phenomena in a controllable way using only a small number of features. We choose two of the most commonly used artificial data generators (Lu et al. 2020): SINE1 (Gama et al. 2004) and SEA (Street and Kim 2001). SINE1 generates points with two attributes \((x_1, x_2)\) each uniformly distributed in \([0, 1]\). The class is determined by \(x_2 - \sin x_1 < \theta\), where \(\theta\) is a threshold value. In SEA, each sample has three attributes \((x_1, x_2, x_3)\) each uniformly distributed in \([0, 10]\). Only the first two attributes are used to determine the label, while the third one is used as noise. The class label is determined by \(x_1 + x_2 \leq \theta\), where \(\theta\) is a threshold value. For convenience, in all the streams, the minority class is always the class \(1\), while the majority one is the class \(0\). For each type of drift, each generator produces two data streams with a different drift speed (Tsymbal 2004): sudden (in an instant of time the first concept finishes and the next one starts) and gradual (a smooth transition from the first concept to the next one). Streams with a gradual concept drift are denoted by \(g\) (SINE1\(_g\) and SEA\(_g\)). Every data stream has 100,000 instances, with one concept drift starting at time step 50,000. The concept drift in SINE1\(_g\) and SEA\(_g\) takes 10,000 time steps to complete. It starts at the time step 45,000 and it fully replaces the old one by the time step 55,000. Moreover, each stream is generated four times, changing the imbalance ratio \(IR\). We use \(IR = [1:9, 2:8, 3:7, 4:6]\). We consider the following three types of concept drift.

*\(p(y)\) Concept Drift.* In this case, the class prior probability changes and, so, well calibrated classifiers can become miscalibrated or imbalance issues may occur. Data streams SINE1 and SINE1\(_g\) have a significant class imbalance change, in which the minority (majority) class of the first half of the streams becomes the majority (minority) during the latter half. SEA and SEA\(_g\) have a less significant change, in which the streams are balanced during the first half and become imbalanced during the latter half. In the gradual drifting cases, \(p(y)\) is changed linearly during the concept transition period (time step 45,000 to time step 55,000). In SINE1 algorithms, we use \(\theta = 0\), while in the SEA ones, we use \(\theta = 7\).

*\(p(X|y)\) Concept Drift.* In this case, the true decision boundary remains unaffected without creating any particular problem to the classifiers. The data stream is constantly imbalanced. In particular, the class imbalance ratio, respectively in each stream, is 1:9, 2:8, 3:7 and 4:6 both before and after the concept drift occurrence. The concept drift in each data stream is determined by introducing a constraint that changes the \(x_1\) probability of the negative class (0) of being lower than a certain value \(n\). Before the drift occurrence, the probability is \(p(x_1 < n) = 0.9\) while after, it is \(p(x_1 < n) = 0.1\). In the gradual drifting cases, it changes linearly during the concept transition period. In SINE1 algorithms, we use \(\theta = 0\) and \(n = 0.5\), while in the SEA ones, we use \(\theta = 7\) and \(n = 5\).
\( p(y|X) \) \textit{Concept Drift}. This is the most critical form of concept drift. The true boundary between classes changes after the drift, so that the previously learnt function does not apply any more. The data stream is constantly imbalanced. In particular, the class imbalance ratio, respectively in each stream, is 1:9, 2:8, 3:7 and 4:6 both before and after the concept drift occurrence. The data distribution in SINE1 and SINE1\(_g\) involves a concept swap i.e. in class 0 from \( x_2 − \sin x_1 ≥ \theta \) to \( x_2 − \sin x_1 < \theta \), while the data distribution in SEA and SEA\(_g\) undergoes a concept drift due to the \( \theta \) value change. Before the concept drift we use \( \theta = 7 \) and after \( \theta = 13 \). The change in SEA and SEA\(_g\) is less critical than the change in SINE1 and SINE1\(_g\), because some of the examples from the old concept are still valid under the new concept after the threshold has moved completely.

So, we have 4 streams for each stream generator and concept drift speed, for a total of 16 streams for each concept drift type and, in the end, a total of 48 data streams.

\textbf{Real data stream} After having studied the phenomena with some low dimensionality synthetic streams, we tested three real data streams having a higher number of features, often used as a benchmark for concept drift problems (Lu et al. 2020): PAKDD, Elec2, and KDD99. The \textit{minority} class is always the class 1, while the \textit{majority} one is the class 0.

PAKDD 2009 credit card data (PAKDD) (Linhart et al. 2009) are collected from the private brand credit card operation of a Brazilian retail chain. The task is to identify whether the client has a \textit{good} or \textit{bad} credit. The \textit{bad} credit is the minority class (20\%) of the provided modelling data. Because the data have been collected over a long period of time (one year), gradual market changes occur. It has 28 features, of which 15 numeric and 13 nominal.

In the Electricity data stream (Elec2) (Harries 1999), the task is to predict a direction for electricity price changes w.r.t. the moving average of the last 24h in the Australian New South Wales Electricity Market. Input variables are recorded every 30 min from May 1996 to December 1998. The data are subject to a concept drift due to changing consumption habits, unexpected events and seasonality. For instance, during the recording period, the electricity market was expanded to include adjacent areas, which allowed production surpluses from one region to be sold to another. The minority class appears in 42.45\% of the cases and the stream has 8 attributes, of which 7 numeric and 1 nominal.

The KDD cup intrusion detection data stream (KDD99) (Dua and Graff 2017) records intrusions simulated in a military network environment. The task is to classify network traffic into \textit{normal} (80.31\% of the cases) or some kind of intrusion (19.69\% of the cases) described by 41 features, of which 34 numeric and 7 nominal. The problem of temporal dependence is particularly evident here. Inspecting the raw stream confirms that there are time periods of intrusions rather than single instances of intrusions.

\textbf{Algorithms} As SML- models to be prepended by VFC-SMOTE, we tested the ARF (Gomes et al. 2019), Naïve Bayes (NV), HAT (Bifet and Gavaldà 2009), K-Nearest Neighbor (KNN) and SWT (Bifet et al. 2013b) with ARF as base learner algorithms. Instead, as SML+ models, we tested the AR\(_{RE}\) (Ferreira et al. 2019), RB (Bernardo et al. 2020a), OOB and UOB (Wang et al. 2013) techniques. We did not test C-SMOTE because it mainly focuses on improving the classification performances neglecting the computational ones. In fact, from a preliminary analysis
of computational costs, pipelines that use C-SMOTE shown to improve the classification performances w.r.t. the SML+ methods, but this result is obtained at a high computational cost. In some situations, C-SMOTE consumes $10 \times$ memory and $1000 \times$ time more than SML+ methods. All the experiments were performed by using the MOA framework (Bifet et al. 2010) with default hyperparameter values for all the techniques involved. The only parameters that we set in VFC-SMOTE are $\text{minSizeMinority}$, $\text{percCorrClass}$ and $\text{Adwin} \ \delta$ values. VFC-SMOTE used with $\text{minSizeMinority} = 100$, $\text{percCorrClass} = 1$ and $\delta = 1e^{-5}$ obtained the best results among all the others tested in a hyper-parameter analysis. Using $\text{percCorrClass} = 1$ means that saving all the samples into the sketch is better than saving only the misclassified ones. In this particular case, using SMOTE, the classification performances improvements are greater than the savings in time used and RAM consumed as compared to the BORDERLINE-SMOTE.

**Settings** All the tests were run in a machine that used 2 virtual CPUs Intel Skylake P-8175 at 2.5 GHz and 8 GiB of RAM and they took about two days for computation. We evaluated the predictive performances using the prequential evaluation approach (Gama et al. 2013) and for each data stream, we performed 10 runs. Following (He and Garcia 2009), we used metrics extracted from a confusion matrix. The model saves the instances that are correctly classified as numbers of True Positives (TP) and True Negatives (TN), while those that are incorrectly classified are saved as numbers of False Positives (FP) and False Negatives (FN). In particular, we used the Recall and F1-Measure metrics for each class: $R[0]$, $F1[0]$ for the majority class and $R[1]$, $F1[1]$ for the minority class. We also used the G-mean (GM) metric. Recall focuses only on the single class, allowing us to see when the recognition of that class drops. In contrast, G-mean captures the balance between recognition ratios of both classes. Therefore, by analyzing both measures it can be noticed whether one class was recognized more often at the cost of the other. Moreover, G-mean is skew-invariant, meaning that its interpretation remains the same for all possible class imbalance ratios, being particularly relevant for studying drifting imbalance ratios. For these experiments, we did not use the Accuracy and the Precision metrics for the following reasons: the former is not reliable in case of imbalanced streams because the impact of the least-represented, and possibly more important, examples is reduced when compared to that of the majority class, while the latter can be easily derived from the comparison between Recall and F1-Measure metrics. We took into account how many seconds and bytes of RAM each algorithm used, too. Since MOA runs in a Java Virtual Machine in which the RAM consumed does not correspond to the real amount consumed, each experiment is run in a Docker container and we tracked the container RAM consumption through InfluxDB.

With synthetic data, knowing when the drift occurs, we calculated the metrics over all the time steps after the concept drift ended. For sudden drifts, we reset the metrics at time step 50,000, while, for gradual drifts, we reset them at time step 45,000 (starting drift) and 55,000 (ending drift). Instead, without knowing the true concept drifts in real-world data, we calculated time-decayed metrics with decay factor 0.995, which means that the old performances were forgotten at the rate of 0.5%.
Results and discussion

In this section, we compare the SML+ methods to the ARF, HAT, NV, KNN, and SWT algorithms prepended by the VFC-SMOTE meta-strategy (from now on called VFC-SMOTE*) in terms of statistical tests, time and memory consumed and recovery speed from concept drift occurrence.

Statistical tests In a tabular form, we presented the measure values averaged over entire streams (mean performance values). We maintain the separation between synthetic and real ones, which we carry out a Nemenyi test (Demsar 2006) with significance level \( \alpha = 0.05 \) to compare the SML+ model performances with the performances achieved by VFC-SMOTE*. Under the Nemenyi test, \( x \succ y \) indicates that algorithm \( x \) is statistically significantly more likely to be more favorable than \( y \). In contrast, \( x \succeq y \) indicates that the former is better than the latter, but without any statistical significance, and we will address this case as \( x \) is statistically similar to \( y \).

In all the tables, \( \text{VFC-SMOTE}_{ARM} \), \( \text{VFC-SMOTE}_{HAT} \), \( \text{VFC-SMOTE}_{KNN} \), \( \text{VFC-SMOTE}_{NV} \), and \( \text{VFC-SMOTE}_{SWT} \) stand for, respectively, VFC-SMOTE strategy used with ARF, HAT, KNN, NV, and SWT as base learners. In particular, Tables 2, 3 and 4 show the average performance results and the ranks using the artificial streams with, respectively, the \( p(y) \), \( p(X|y) \), and \( p(y|X) \) concept drift types. Table 5, instead, shows the results achieved with the real streams.

Tables 2, and 3 show that, with \( p(y) \) and \( p(X|y) \) drift types (virtual drift), VFC-SMOTE is not the best algorithm in terms of average results. Instead, in terms of statistical significance, we can notice that at least one SML- model prepended by VFC-SMOTE (ARF) is, in most cases, similar to the best one. In the case of \( p(y|X) \) drift type, i.e., the most important one to address, Table 4 shows that, in the minority class performances and G-mean, ARF prepended by VFC-SMOTE performed better than all others both in terms of average results achieved and statistical significance. In the case of real streams, Table 5 shows that there are not any statistical differences among the ranks. This is probably due to the limited number of real streams tested (3) w.r.t. the artificial ones (48) and the significance level used. However, we can conclude that there is at least one SML- model prepended by VFC-SMOTE* (in general ARF or SWT) that outperforms the SML+ ones, therefore \( H_p.1 \) is verified.

Time & memory consumption Before illustrating the synthesis of the time & memory comparison using all the SML+ algorithms, we discuss the comparison among the ARFRE, RB, OOB, and UOB models to select the best ones to compare them with VFC-SMOTE *. We used the Nemenyi test (Demsar 2006) with significance level \( \alpha = 0.05 \) to compare the average results, divided for synthetic and real streams, achieved by the ARFRE, RB, OOB, and UOB algorithms in each metric, and to find the best one.

Figures 4 and 5 show the Nemenyi results. The axis represents the average rank achieved, while \( CD \) is the critical difference, i.e., the minimum difference that two ranks must have to be considered statistically different. From the former, we infer that, with synthetic streams, the best SML+ model is ARFRE, while the latter shows that, even if there is not any statistical evidence due to the limited number of streams tested, the best SML+ model to use with real streams is RB. Henceforth regarding the synthetic data streams, we proceeded to compare the ARFRE technique, while
| Algorithm          | F1[1]$^1$  | F1[0]  | R[1]$^2$  | R[0]  | GM$^3$  |
|--------------------|-----------|--------|-----------|--------|--------|
| VFC - SMOTE$_{ARF}$ | 91.12 (3.00) | 94.48 (3.06) | 89.33 (3.44) | 96.02 (3.62) | 68.02 (2.62) |
| VFC - SMOTE$_{HAT}$ | 89.29 (6.31) | 92.94 (6.88) | 88.47 (4.69) | 93.77 (6.94) | 67.43 (6.88) |
| VFC - SMOTE$_{KNN}$ | 88.31 (7.69) | 92.42 (7.44) | 87.71 (6.44) | 92.89 (7.75) | 67.14 (7.69) |
| VFC - SMOTE$_{NV}$  | 82.95 (9.00) | 85.50 (9.00) | 84.68 (6.00) | 88.15 (8.12) | 65.68 (9.00) |
| VFC - SMOTE$_{SWT}$ | 90.66 (4.06) | 93.88 (4.31) | 89.47 (3.88) | 94.76 (4.50) | 67.80 (4.50) |
| ARF$_{RE}$          | 92.07 (1.31) | 95.32 (1.31) | 88.86 (5.75) | 97.72 (1.31) | 68.25 (1.88) |
| OOB                | 91.33 (2.81) | 94.73 (2.50) | 89.68 (3.88) | 96.30 (3.12) | 68.13 (2.50) |
| RB                 | 90.25 (5.06) | 94.15 (5.50) | 87.57 (7.50) | 96.12 (3.62) | 67.71 (5.56) |
| UOB                | 90.00 (5.75) | 94.11 (5.00) | 89.76 (3.44) | 95.35 (6.00) | 67.97 (4.38) |

$^1$ $ARF_{RE}$ $\succ$ (OOB $\succeq$ VFC - SMOTE$_{ARF}$) $\succ$ Others

$^2$(UOB $\succeq$ VFC - SMOTE$_{ARF}$) $\succ$ Others

$^3$ (ARF$_{RE}$ $\succeq$ OOB $\succeq$ VFC - SMOTE$_{ARF}$) $\succ$ Others
### Table 3
Avg. results and ranks (in brackets) on synthetic streams with $p(X|y)$ drift type comparing VFC-SMOTE * with the other SML+ models. Results in **bold** are the best for that metric

| Algorithm         | F1[1] $^1$ | F1[0] | R[1] $^2$ | R[0] | GM $^3$ |
|-------------------|------------|-------|-----------|------|---------|
| VFC − SMOTE$_{ARF}$ | 83.86 (1.94) | 94.64 (2.88) | 86.52 (3.12) | 93.78 (4.00) | 67.06 (2.19) |
| VFC − SMOTE$_{HAT}$ | 81.52 (5.75) | 93.58 (6.50) | 84.96 (5.38) | 92.38 (6.31) | 66.50 (5.88) |
| VFC − SMOTE$_{KNN}$ | 81.58 (5.69) | 93.95 (6.19) | 84.00 (6.50) | 93.36 (5.69) | 66.52 (5.75) |
| VFC − SMOTE$_{NV}$ | 74.63 (8.62) | 90.27 (8.94) | 82.45 (6.94) | 87.30 (8.69) | 65.04 (8.38) |
| VFC − SMOTE$_{SWT}$ | 82.03 (4.38) | 94.80 (2.88) | 80.53 (5.50) | 95.25 (2.94) | 66.13 (4.81) |
| ARF$_{RE}$         | 71.16 (4.06) | **95.19 (2.25)** | 67.91 (6.62) | **98.20 (1.00)** | 63.98 (4.62) |
| OOB                | **84.73 (2.75)** | 94.44 (4.38) | 88.42 (2.44) | 93.10 (5.06) | **67.29 (2.19)** |
| RB                 | 79.89 (5.75) | 94.93 (3.88) | 76.32 (7.38) | 96.23 (3.06) | 65.49 (6.81) |
| UOB                | 80.43 (6.06) | 92.43 (7.12) | **89.80 (1.12)** | 89.43 (8.25) | 66.85 (4.38) |

$^1$ (VFC − SMOTE$_{ARF}$ ⪰ OOB) ≻ Others

$^2$ UOB ≻ (OOB ⪰ VFC − SMOTE$_{ARF}$) ≻ Others

$^3$ (OOB ⪰ VFC − SMOTE$_{ARF}$) ≻ Others
Table 4  Avg. results and ranks (in brackets) on synthetic streams with $p(y|X)$ drift type comparing VFC-SMOTE * with the other SML+ models. Results in bold are the best for that metric.

| Algorithm          | F1[1] | F1[0] | R[1] | R[0] | GM  |
|--------------------|-------|-------|------|------|-----|
| VFC − SMOTE_{ARF} | 78.80 (1.50) | 92.97 (3.19) | 81.97 (1.50) | 91.90 (4.50) | 65.81 (1.44) |
| VFC − SMOTE_{HAT} | 76.19 (3.25) | 92.01 (5.56) | 79.78 (2.69) | 90.75 (6.50) | 65.19 (3.00) |
| VFC − SMOTE_{KNN} | 75.81 (3.69) | 92.16 (4.75) | 77.36 (4.19) | 91.72 (4.88) | 64.92 (4.12) |
| VFC − SMOTE_{NV}  | 50.54 (7.88) | 75.62 (8.25) | 52.51 (7.31) | 73.89 (7.94) | 53.74 (7.88) |
| VFC − SMOTE_{SWT} | 74.63 (4.44) | 93.12 (3.75) | 72.21 (4.56) | 94.13 (3.19) | 64.23 (4.31) |
| ARFRE              | 63.89 (5.00) | 93.78 (1.50) | 60.11 (5.44) | 97.49 (1.25) | 62.27 (5.00) |
| OOB                | 66.11 (5.38) | 87.65 (5.25) | 68.13 (6.38) | 87.28 (5.12) | 61.95 (5.94) |
| RB                 | 59.05 (7.75) | 88.28 (4.69) | 54.59 (8.00) | 90.82 (3.88) | 59.89 (7.75) |
| UOB                | 61.15 (6.12) | 81.57 (8.06) | 71.01 (4.94) | 78.12 (7.75) | 60.22 (5.56) |

1 VFC − SMOTE_{ARF} > (VFC − SMOTE_{HAT} ≥ VFC − SMOTE_{KNN}) > Others
2 VFC − SMOTE_{ARF} > VFC − SMOTE_{HAT} > Others
3 VFC − SMOTE_{ARF} > (VFC − SMOTE_{HAT} ≥ VFC − SMOTE_{KNN}) > Others
Table 5  Avg. results and ranks (in brackets) on real streams comparing VFC-SMOTE * with the other SML+ models. Results in **bold** are the best for that metric

| Algorithm          | F1[1] | F1[0] | R[1] | R[0] | GM |
|--------------------|-------|-------|------|------|----|
| VFC – SMOTE_ARF    | 62.77 (4.67) | 92.44 (3.33) | 62.33 (3.67) | 96.05 (3.67) | 62.14 (4.67) |
| VFC – SMOTE_HAT    | 58.33 (6.67) | 89.61 (6.33) | 61.06 (6.00) | 92.80 (7.67) | 61.45 (4.33) |
| VFC – SMOTE_KNN    | 59.04 (6.33) | 89.81 (6.00) | 57.46 (6.67) | 94.29 (5.33) | 60.86 (7.33) |
| VFC – SMOTE_NV     | 60.74 (6.00) | 71.31 (8.67) | 72.53 (6.33) | 72.69 (6.67) | 59.91 (7.00) |
| VFC – SMOTE_SWT    | 63.67 (**2.67**) | 92.36 (3.67) | 62.74 (**3.33**) | 95.85 (3.33) | 62.19 (4.00) |
| ARF_RE             | 61.38 (5.33) | **92.73 (**2.33**)** | 60.58 (5.67) | **97.36 (**3.00**)** | 62.04 (6.00) |
| OOB                | **70.59 (4.67)** | 88.88 (4.33) | 72.07 (4.67) | 88.31 (4.67) | **62.93 (2.67)** |
| RB                 | 62.27 (3.33) | 92.58 (2.33) | 62.32 (3.67) | 96.16 (**3.00**) | 62.15 (3.67) |
| UOB                | 66.24 (5.33) | 79.39 (8.00) | **82.53 (5.00)** | 76.73 (7.67) | 62.62 (5.33) |

1 F1[1] \(\geq\) RB \(\geq\) Others
2 VFC – SMOTE_SWT \(\geq\) RB \(\geq\) Others
3 OOB \(\geq\) RB \(\geq\) VFC – SMOTE_SWT \(\geq\) Others
regarding the real ones, we proceeded to compare the RB technique. Appendix C shows the comparisons with OOB, UOB, and RB algorithms using synthetic data streams and with OOB, UOB, and ARFRE algorithms using real streams, too.

Figures 7 and 8 illustrate a synthesis of the time & memory comparison between VFC-SMOTE* and the ARFRE and RB algorithms.
Each row represents the comparison between a particular algorithm prepended by VFC-SMOTE and the state-of-the-art technique, while each column represents a different data stream tested. In particular, Fig. 6 shows a single cell of Fig. 7. For each combination of algorithms tested and streams used, Fig. 6 shows a scatter plot that compares the ratio between the time used and the RAM consumed by a specific algorithm prepended by VFC-SMOTE and the state-of-the-art one. Red, yellow, and blue points, respectively, refer to the $P(y)$, $P(X|y)$, and $P(y|X)$ concept drift types. If a point is on the bottom-left quadrant of the grid, it means that the algorithm prepended by VFC-SMOTE takes less time and consumes less RAM than the state-of-the-art algorithm. If a point is on the bottom-right quadrant of the grid, it means that the algorithm prepended by VFC-SMOTE consumes less RAM than the state-of-the-art algorithm, but it takes more time. Instead, if a point is on the top-left quadrant of the grid, it means that the algorithm prepended by VFC-SMOTE consumes less RAM than the state-of-the-art algorithm, but it takes more time. In the last case, if a point is on the top-right quadrant, it means that the algorithm prepended by VFC-SMOTE takes more time and consumes more RAM than the state-of-the-art one.

In particular, if a point lays on the vertical dotted line, it means that the two algorithms consume the same RAM, while if a point lays on the horizontal dotted line, it means that the two algorithms take the same time. To enhance the data visualization, all the time and RAM ratios are scaled using the maximum time and RAM values achieved by the comparison of the VFC-SMOTE*, ARF$_{RE}$, RB, OOB and UOB algorithms in case of both artificial and real streams.

A quick look at Figs. 7 and 8 is enough to say that the ratios change depending on the algorithms and the data streams tested, but there is at least one algorithm

---

2 Max Time Ratio = 24.02; Max RAM Ratio = 13.56.

3 Max Time Ratio = 100.18; Max RAM Ratio = 6.87.
Fig. 7 Ratio between time & memory consumed by VFC-SMOTE* and the ARFRE algorithm with synthetic streams. In cells with green borders, all the points are on the bottom-left quadrant and thus, the algorithm prepended by VFC-SMOTE takes less time and consumes less RAM with all the concept drift types.

Fig. 8 VFC-SMOTE* and RB ratio with real streams.

Prepended by VFC-SMOTE* that is faster and a better RAM saver than the state-of-the-art methods (green bordered cells). More specifically, Fig. 7 shows the ratios between time and memory consumed by VFC-SMOTE* and ARFRE algorithm for each synthetic stream used and concept drift type. We can notice that both time and RAM used are pretty similar among all the methods involved except for the case of the KNN model prepended by VFC-SMOTE that always consumes less RAM than ARFRE. Moreover, there are 14 cases (green bordered cells) in which, in all the concept drift types, VFC-SMOTE* takes both less time and consumes less RAM than ARFRE.

Figure 8 shows the ratios between the time and memory consumed by VFC-SMOTE* and RB algorithm for each real stream. We can notice that VFC-SMOTE* never consumes more RAM than RB and, in more than half of the cases (9), VFC-SMOTE* is also faster.

We can conclude that, in the majority of cases, VFC-SMOTE* uses less time and consumes less RAM than the SML+ methods or, in the worst-case scenario, it uses the same amount of time and RAM. So, looking also at Appendix C, Hp. 2 is verified.
**Recovery speed analysis** VFC-SMOTE is just a meta-strategy focusing on improving the minority class performances. As Table 1 shows, it leaves to the algorithm to which it is prepended (SML-) to manage the concept drift. If the SML- model uses a concept drift detector then also the VFC-SMOTE pipeline will be able to manage concept drifts, otherwise it will not. In Appendix D we compared, using different datasets and metrics, the performances of two SML- models with and without VFC-SMOTE. We can notice that, despite the fact that the VFC-SMOTE pipelines performances are better than the single models performances, they all start reacting to the concept drift (red line) at the same time. This means that VFC-SMOTE does not have any impact on the concept drift recovery and so *Hp. 3* is verified. Moreover, for this reason we did not compare the VFC-SMOTE recovery speed analysis to the state-of-the-art methods one.

**6 Limitations**

In this section we discuss the most important VFC-SMOTE limitations. The first one is the inability to know in advance which specific SML- model prepended with VFC-SMOTE can outperform the other SML+ methods. For this purpose, we need to test different SML- models to find out which is the best one in each situation. Another limitation is that, for the time being, VFC-SMOTE can only deal with binary classification problems. Our aim was to start from the easiest problem to determine whether our work could achieve good performances w.r.t. the state-of-the-art methods. Moreover, VFC-SMOTE uses the SMOTE, BORDERLINE-SMOTE, ADWIN and HAT techniques. In a future extension of this work, we will focus on both addressing multi-class problems and using different techniques. Another important aspect to take into consideration is the synthetic oversampling. The risk is that not all the synthetic generated instances are consistent with the underlying context. In our case, considering that we are using the most seen value with nominal attributes, and a Beta distribution scaled between the minimum and maximum values observed with numerical attributes, we have thus minimized the risk of generating non-coherent instances. This also depends on the stream class distribution. In case of a sharp boundary where the classes do not overlap, we avoided the problem, otherwise the risk would still remain. Another class distribution consequence is the presence of a sort of trade-off between improving the minority class performances and decreasing the majority class ones. In case of non-overlapping classes, minority class performances can be improved without decreasing significantly the majority class ones, but, in case of overlapping classes, in light of a minority class performances improvement, a majority class performances decrease is unavoidable. Lastly, we chose a Beta distribution to generate the new synthetic samples. A sensitivity analysis could be performed comparing the results achieved using the Beta distributions to the results achieved using different distributions having bounded intervals, i.e. Truncated normal distribution, Logit-normal distribution, Irwin-Hall distribution or Bates distribution.
Table 6  Number of times that VFC-SMOTE prepended to the SML- model performed better than the SML+ models tested

| Data | F1[1] | F1[0] | R[1] | R[0] | GM |
|------|-------|-------|------|------|----|
|      | ARF   | HAT   | SWT  | ARF  | HAT | SWT |
| Syn. | 141   | 86    | 105  | 115  | 55  | 103 |
|      | 136   | 108   | 111  | 92   | 54  | 93  |
|      | 98    | 56    | 78   | 3    | 6   | 3   |
| Real | 6     | 3     | 7    | 6    | 4   | 7   |
|      | 8     | 3     | 8    | 7    | 3   | 7   |
|      | 6     | 3     | 6    | 6    | 3   | 6   |

The table with all the SML- models tested is shown in Appendix E. **Bold** means that it performed better in more than half of the occurrences.
7 Conclusions

In this work, we present the VFC-SMOTE meta-strategy, inspired by the popular SMOTE and BORDERLINE-SMOTE techniques, that allows balancing an evolving data stream one sample at a time, and that can be used as a data filter with all the streaming machine learning techniques. Compared to SMOTE and BORDERLINE-SMOTE, our meta-strategy does not need i) a static batch during the pre-processing phase, and ii) to check the number of minority and majority class samples among the neighbors to generate a new sample. Instead, it saves the samples in a sketch that incorporates ADWIN and, it uses the summaries contained into the sketch to introduce new synthetic samples. The VFC-SMOTE memory and time (per sample) asymptotic costs are $O(\log W)$, where $W$ is the ADWIN window length.

We tested VFC-SMOTE prepended to some SML- and SML+ algorithms on different cases of imbalance, concept drifts, and swapping between minority and majority classes. Furthermore, we measured the time and memory consumed.

The results summarized in Table 6 empirically demonstrate that there are at least two SML- models (ARF and SWT) that, prepended by VFC-SMOTE, improve both the minority and majority class performances in more than half of the cases if compared to the SML+ methods (Q1). Moreover, with reference to time and memory consumed, the SML- algorithms prepended by VFC-SMOTE are faster and less memory eager than the SML+ algorithms (Q2), while with reference to the recovery speed analysis, the SML- models behaviour is not influenced by VFC-SMOTE (Q3).

In future works, we would improve the VFC-SMOTE performances for both classes, not only for the minority class ones. Another aim is to investigate other meta-strategies based on different rebalance techniques, different sketch structures and different concept drift detectors and to compare them with VFC-SMOTE. In the long term, adapting VFC-SMOTE to multiclass and regression tasks could be interesting.

Author Contributions A. Bernardo is a PhD student of Prof. E. Della Valle.

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement. No funds, grants, or other support were awarded.

Availability of data and material https://dataverse.harvard.edu/dataverse/cd_ir.

Declarations

Conflict of interest All authors declared that they have no conflict of interest.

Code availability https://github.com/alessiobernardo/VFC-SMOTE.

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
Appendices

Appendix A: Further sampling techniques for class imbalance

The ADASYN (He et al. 2008) main idea proceeds from the assumption of utilizing a weighted distribution depending on the type of minority examples according to their learning complexity. The quantity of synthetic data for each one is associated with the level of difficulty of each minority example. This difficulty estimation is based on the ratio of examples belonging to the majority class in the neighborhood. Then, a density distribution is computed using all the ratios of the minority instances, which will be used to compute the number of synthetic examples required to be generated for each minority example. DBSMOTE (Bunkhumpornpat et al. 2012) relies on a density-based approach to clustering called DBSCAN and performs oversampling by generating synthetic samples along the shortest path from each minority instance to a pseudo-centroid of a minority class cluster. DBSMOTE was inspired by BORDERLINE-SMOTE in the sense that it operates in an overlapping region, but unlike BORDERLINE-SMOTE, it also tries to maintain both the minority and majority class accuracies. MDO (Abdi and Hashemi 2015) builds synthetic examples having the same Mahalanobis distance from each examined class mean as the other minority examples. Thus, the region of minority instances can be better learned by preserving the co-variance during the generation of synthetic examples along the probability contours. Also, the risk of overlapping between different class regions is reduced. SWIM (Bellinger et al. 2020) is based on the Mahalanobis distance, too. It generates synthetic minority training examples that are (1) near to their minority seed and (2) have the same Mahalanobis distance from the mean of the majority class as their seed. This ensures that the synthetic instances do not spread into denser regions of the majority class where there is no statistical evidence that they should be. The last technique proposed is G-SMOTE (Douzas and Bação 2019). It substitutes the SMOTE data generation mechanism by defining a flexible geometric region around each minority class instance. Then, synthetic instances are generated inside the boundaries of the region. At the most general choice of hyper-parameters, this geometric region of the input space is a truncated hyper-spheroid.

Appendix B: Asymptotic complexity analysis

From Table 7, which shows the memory, asymptotic memory, time per sample, and asymptotic time per sample complexity of the SML- models used to test the VFC-SMOTE meta-strategy, we can notice that all the models that use ADWIN (e.g. ARF, HAT and SWT) require $O(\log W)$ asymptotic memory and time, where $W$ is the ADWIN window length. Instead, the other two models (e.g. KNN and NV) require $O(1)$ asymptotic memory and time.
| SML- model         | Mem                                      | Asym. Mem       | Time                                      | Asym. Time     |
|--------------------|------------------------------------------|-----------------|-------------------------------------------|----------------|
| ARF (Gomes et al. 2019) | $O(E(TAVC + TM\log(W/M))$                | $O(\log W)$    | $O(E(AVC + \log W))$                      | $O(\log W)$   |
| HAT (Bifet and Gavaldà 2009) | $O(TAVC + TM\log(W/M))$                  | $O(\log W)$    | $O(AVC + \log W)$                        | $O(\log W)$   |
| KNN (Bifet et al. 2013a) | $O(AW_{KN})$                             | $O(1)$          | $O(AW_{KNN})$                             | $O(1)$         |
| NV (John and Langley 2013) | $O(AVC)$                                 | $O(1)$          | $O(AC)$                                   | $O(1)$         |
| SWT (Bifet et al. 2013b)  | $O(E(T(A+1)VC + M\log(W/M))$             | $O(\log W)$    | $O(E((A+1)VC + \log W))$                 | $O(\log W)$   |

Here, $A$ is the number of attributes, $V$ is the maximum number of values for an attribute, $C$ is the number of classes ($2$), $T$ is the number of tree nodes, $W$ is the Adwin window length, $M$ is the number of buckets used by Adwin, $I$ is the number of synthetic samples to introduce at each moment, $K$ is the nearest neighbors to use ($10$), $W_{KNN}$ is the KNN window length ($1000$), and $E$ is the number of ARF and SWT ensemble size ($100$). **Note that all the time complexity are per sample.**
The total memory complexity of the proposed solution is the sum of the space used to store the sketch and the SML- model to which VFC-SMOTE is prepended. Since the sketch is stored in a HAT model, its asymptotic complexity is $O(\log W)$. Thus, whether the SML- model uses Adwin or not, the total asymptotic memory complexity is still $O(\log W)$. Notice that, in the latter case, the asymptotic complexity worsens from $O(1)$ to $O(\log W)$.

Instead, the time complexity per sample of the proposed solution is the sum of the time used to (1) save the element into the sketch, (2) retrieve all the summaries from the sketch, (3) sort the summaries by the number of instances seen, (4) generate new synthetic samples, (5) train the SML- model with the newly generated synthetic samples, and (6) train the SML- with the sample arrived in input. So, the total time complexity is, respectively, $O(\text{AVC} + \log W) + O(T + L) + O((T + L)\log(T + L)) + O(AI) + O(I \ast \text{SML} - ) + O(\text{SML} - )$, where here $A$ is the number of attributes, $V$ is the maximum number of values for an attribute, $C$ is the number of classes (2), $T$ is the number of tree nodes, $L$ is the number of tree leaves, $M$ is the number of buckets used by Adwin, $I$ is the number of synthetic instances to generate to rebalance the stream in that moment and $O(\text{SML} - )$ stands for the time complexity of the SML- model. Asymptotically, also this cost becomes $O(\log W)$ and so, as before, whether the SML-model uses Adwin or not, the total time asymptotic complexity is still $O(\log W)$.

**Appendix C: Further data on time & memory consumption**

Figure 9 shows the ratio between time & memory consumed by VFC-SMOTE* and, respectively, OOB, UOB, RB algorithms with synthetic data streams and ARF\textsubscript{RE}, OOB and UOB with real data streams.
Fig. 9 Ratio between time & memory consumed by VFC-SMOTE* and the state-of-the-art algorithms with synthetic and real data streams. In cells with a green border, all the points are on the bottom-left quadrant and so, the algorithm prepended by VFC-SMOTE takes less time and consumes less RAM with all the concept drift types.
Appendix D: Recovery speed analysis

Figure 10 shows the performances comparison between the ARF and HAT techniques prepended with and without VFC-SMOTE. The red solid lines represent the concept drift occurrence, while the red dashed lines represent the start and the end of a gradual drift.

Appendix E: Summary of results

Table 8 shows the summary results of all the SML- methods prepended by VFC-SMOTE.

![Graphs showing performance comparison](image)

(a) SINE1 with IR=2:8 and $p(y)$ concept drift
(b) SEA with IR=1:9 and $p(X|y)$ concept drift
(c) SEA with IR=2:8 and $p(y|X)$ concept drift
(d) SEAg with IR=1:9 and $p(y|X)$ concept drift

Fig. 10 Comparison between ARF and HAT prepended with and without VFC-SMOTE
Table 8  Number of times that VFC-SMOTE prepended to the SML- model performed better than the SML+ models tested

| Data | F1[1] | F1[0] | R[1] | R[0] | GM |
|------|-------|-------|------|------|----|
|      | ARF   | HAT   | KNN  | NV   | SWT | ARF | HAT | KNN  | NV   | SWT | ARF | HAT | KNN  | NV   | SWT | ARF | HAT | KNN  | NV   | SWT | ARF | HAT | KNN  | NV   | SWT |
| Syn. | 141   | 86    | 78   | 20   | 105 | 115 | 55  | 54  | 6    | 103 | 136 | 108 | 97  | 63  | 111 | 92  | 54  | 53  | 15  | 93  | 143 | 78  | 72  | 21  | 98  |
| Real | 6     | 3     | 4    | 4    | 7   | 6   | 4   | 3   | 1    | 7   | 8   | 3   | 3   | 4   | 8   | 7   | 3   | 5   | 3   | 7   | 6   | 3   | 3   | 2   | 6   |

**Bold** means that it performed better in more than half of the occurrences
References

Abdi L, Hashemi S (2015) To combat multi-class imbalanced problems by means of over-sampling and boosting techniques. Soft Comput 19(12):3369–3385. https://doi.org/10.1007/s00500-014-1291-z

Bellinger C, Sharma S, Japkowicz N, Zaiane OR (2020) Framework for extreme imbalance classification: SWIM - sampling with the majority class. Knowl Inf Syst 62(3):841–866. https://doi.org/10.1007/s10115-019-01380-z

Bernardo A, Della Valle E, Bifet A (2020a) Incremental rebalancing learning on evolving data streams. In: Fatta GD, Sheng VS, Cuzzocrea A, Zaniolo C, Wu X (eds) 20th International Conference on Data Mining Workshops, ICDM Workshops 2020, Sorrento, Italy, November 17-20, 2020, IEEE, pp 844–850, https://doi.org/10.1109/ICDMW51313.2020.00121

Bernardo A, Gomes HM, Montiel J, Pfahringer B, Bifet A, Della Valle E (2020b) C-SMOTE: continuous synthetic minority oversampling for evolving data streams. In: Wu X, Jermaine C, Xiong L, Hu X, Kotevska O, Lu S, Xu W, Aluru S, Zhai C, Al-Masri E, Chen Z, Saltz J (eds) IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020, IEEE, pp 483–492, https://doi.org/10.1109/BigData50022.2020.9377768

Bifet A, Gavaldá R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA, SIAM, pp 443–448, https://doi.org/10.1137/1.9781611972771.42

Bifet A, Gavaldá R (2009) Adaptive learning from evolving data streams. In: Adams NM, Robardet C, Siebes A, Boulicaut J (eds) Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009 Proceedings, Springer, Lecture Notes in Computer Science, vol 5772, pp 249–260, https://doi.org/10.1007/978-3-642-03915-7_22

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: Massive online analysis. J Mach Learn Res 11:1601–1604

Bifet A, Pfahringer B, Read J, Holmes G (2013a) Efficient data stream classification via probabilistic adaptive windows. In: Shin SY, Maldonado JC (eds) Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC ’13, Coimbra, Portugal, March 18-22, 2013, ACM, pp 801–806, https://doi.org/10.1145/2480362.2480516

Bifet A, Read J, Zliobaite I, Pfahringer B, Holmes G (2013b) Pitfalls in benchmarking data stream classification and how to avoid them. In: Blockeel H, Kersting K, Nijssen S, Zelezný F (eds) Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I, Springer, Lecture Notes in Computer Science, vol 8188, pp 465–479, https://doi.org/10.1007/978-3-642-40988-2_30

Bifet A, Gavaldá R, Holmes G, Pfahringer B (2018) Machine learning for data streams with practical examples in MOA, MIT Press, Cambridge

Bunkhumpornpat C, Sinapiromsaran K, Lursinsap C (2012) DBSMOTE: density-based synthetic minority over-sampling technique. Appl Intell 36(3):664–684. https://doi.org/10.1007/s10489-011-0287-y

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) SMOTE: Synthetic minority over-sampling technique. J Artif Intell Res 16:321–357. https://doi.org/10.1613/jair.953

Cormode G (2017) Data sketching. Commun ACM 60(9):48–55. https://doi.org/10.1145/3080008

Demurs J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Domingos PM, Hulten G (2000) Mining high-speed data streams. In: Ramakrishnan R, Stolfo SJ, Bayardo RJ, Parsa I (eds) Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000, ACM, pp 71–80, https://doi.org/10.1145/347090.347107

Douzas G, Baçın F (2019) Geometric SMOTE a geometrically enhanced drop-in replacement for SMOTE. Inf Sci 501:118–135. https://doi.org/10.1016/j.ins.2019.06.007

Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml, Accessed 16 June 2021

Fernández A, García S, Herrera F, Chawla NV (2018) SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. J Artif Intell Res 61:863–905, https://doi.org/10.1613/jair.1.11192

Ferreira LEB, Gomes HM, Bifet A, Oliveira LS (2019) Adaptive random forests with resampling for imbalanced data streams. In: International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019, IEEE, pp 1–6, https://doi.org/10.1109/IJCNN.2019.8852027

Springer
VFC-SMOTE: very fast continuous synthetic minority…

Engineering - WISE 2007, 8th International Conference on Web Information Systems Engineering, Nancy, France, December 3-7, 2007, Proceedings, Springer, Lecture Notes in Computer Science, vol 4831, pp 633–642, https://doi.org/10.1007/978-3-540-76993-4_55

Mirza B, Lin Z, Liu N (2015) Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift. Neurocomputing 149:316–329. https://doi.org/10.1016/j.neucom.2014.03.075

Oza NC (2005) Online bagging and boosting. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, Hawaii, USA, October 10-12, 2005, IEEE, pp 2340–2345, https://doi.org/10.1109/ICSMC.2005.1571498

Pozzolo AD, Boracchi G, Caelen O, Alippi C, Bontempi G (2018) Credit card fraud detection: A realistic modeling and a novel learning strategy. IEEE Trans Neural Networks Learn Syst 29(8):3784–3797. https://doi.org/10.1109/TNNLS.2017.2736643

Street WN, Kim Y (2001) A streaming ensemble algorithm (SEA) for large-scale classification. In: Lee D, Schkolnick M, Provost FJ, Srikant R (eds) Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, 2001, ACM, pp 377–382, https://doi.org/10.1145/502512.502568

Tsymbal A (2004) The problem of concept drift: definitions and related work. Comput Sci Dep Trinity College Dublin 106(2):58

Wang B, Pineau J (2016) Online bagging and boosting for imbalanced data streams. IEEE Trans Knowl Data Eng 28(12):3353–3366. https://doi.org/10.1109/TKDE.2016.2609424

Wang S, Minku LL, Yao X (2013) A learning framework for online class imbalance learning. In: Proceedings of the IEEE Symposium on Computational Intelligence and Ensemble Learning, CIEL 2013, IEEE Symposium Series on Computational Intelligence (SSCI), 16-19 April 2013, Singapore, IEEE, pp 36–45, https://doi.org/10.1109/CIEL.2013.6613138

Wang S, Minku LL, Yao X (2015) Resampling-based ensemble methods for online class imbalance learning. IEEE Trans Knowl Data Eng 27(5):1356–1368. https://doi.org/10.1109/TKDE.2014.2345380

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