Journal of Information and Data Management, 2023, 14:1, doi: 10.5753/jidm.2023.3150

Adaptive Fast XGBoost for Multiclass Classification

Fabiano Baldo [**Santa Catarina State University** | ***fabiano.baldo@udesc.br***]
Julia Grando [**Santa Catarina State University** | ***juliagrando@gmail.com***]
Yuji Yamada Correa [**Santa Catarina State University** | ***yujiyc01@gmail.com***]
Deividy Amorim Policarpo [**Santa Catarina State University** | ***deividy.policarpo@udesc.br***]

Department of Computer Science, Santa Catarina State University, Rua Paulo Malschitzki, 200, Zona Industrial Norte, Joinville, SC, 89219-710, Brazil.

Received: 26 February 2023 • **Published:** 20 October 2023

Abstract The popularization of sensoring and connectivity technologies like 5G and IoT are boosting the generation of data streams. Such kinds of data are one of the last frontiers of data mining applications. However, data streams are massive and unbounded sequences of non-stationary data objects that are continuously generated at rapid rates. To deal with these challenges, the learning algorithms should analyze the data just once and update their classifiers to handle the concept drifts. The literature presents some algorithms to deal with the classification of multiclass data streams. However, most of them have high processing time. Therefore, this work proposes a XGBoost-based classifier called AFXGB-MC to fast classify non-stationary data streams with multiple classes. We compared it with the six state-of-the-art algorithms for multiclass classification found in the literature. The results pointed out that AFXGB-MC presents similar accuracy performance, but with faster processing time, being twice faster than the second fastest algorithm from the literature, and having fast drift recovery time.

Keywords: Multiclass Classification, XGBoot, Fast Classification, Data Stream Mining, Supervised Classification

1 Introduction

The generation of digital data streams is getting more common at the same pace as the sensoring and connectivity technologies are becoming widely applied. This phenomenon is especially driven by technologies like smart devices (e.g. smartphones), IoT (Internet of Things) and 5G. In the contemporary world, the possibility of making appropriate business decisions on the basis of knowledge hidden in large amounts of data is one of the critical success factors for organizations [Krawczyk *et al*., 2017]. Considering the large amount of available data, Data Streams are one of the last frontiers to mining data in order to take more appropriate decisions.

Some scenarios of application of data streams mining include data generated by sensor networks, meteorological stations, stock markets, computer networks, traffic control systems, ubiquitous computing, GPS and mobile device tracking, user's click log and sentiment [Krawczyk *et al*., 2017; Silva *et al*., 2013]. However, data streams have special characteristics that distinguish them from other ordinary datasets. They are massive and unbounded sequences of non-stationary data objects that are continuously generated at rapid rates [Aggarwal, 2006; Gama and Gaber, 2007]. The non-stationary phenomenon, also known as concept drift, means that the concept about each data may shift from time to time, each time after some minimum permanence[Silva *et al*., 2013].

In order to deal with the concept drift manifestation, the models need to be trained in an online fashion by processing data sequentially when they arrive, incorporating their knowledge into the model through continuous updates [Fields *et al*., 2019]. In machine learning, model is the name given to the result of the training process. The model generalizes the patterns contained in the training dataset and thus it can be used to predict the output pattern of new instances of data. To the multiclass classification, the pattern is the instance class. However, to obtain the maximum precision in the shortest time possible, the models have to be rapidly updated as soon as the data arrive through the stream [Bifet *et al*., 2018; Gomes *et al*., 2017]. By doing so, the models are ready to handle the drifts once they appear, keeping high precision in the prediction process.

The literature presents some state-of-the-art algorithms for data stream analysis that handle concept drift. Some examples are: Adaptive Random Forest (ARF)[Gomes *et al*., 2017], Hoeffding Adaptive Trees (HAT) [Bifet and Gavalda, 2009], Leveraging Bagging (LbHT) [Bifet *et al*., 2010], Oza Bagging (ObHT) [Oza and Russell, 2001], Self Adjusting Memory KNN (SAMKNN) [Losing *et al*., 2016a], and Improved Online Ensembles (IOE) [Vafaie *et al*., 2020].

However, despite they present quite suitable precision, most of them consume a considerable amount of processing time. This happens because either they create ensemble learners, such as ARF, LbHT, ObHT and IOE, or have high computational complexity, like in the SAM-kNN, which is based on the KNN approach [Deng *et al*., 2016].

In contrast to the aforementioned algorithms, eXtreme Gradient Boosting (XGBoost) is a boosting-based algorithm that uses decision trees as weak learners, creating each individual tree in parallel, thus decreasing the overall ensemble training time and, hence, improving its time performance [Santhanam *et al*., 2016; Chen and Guestrin, 2016]. Boosting algorithms are iterative processes that evaluate the models' prediction and increase the weight of samples with predictive error in order to improve the learning on them in the creation

of the next weak learner, improving the final prediction result [Schapire, 1999; Ferreira and Figueiredo, 2012]. Despite its fast processing characteristic, XGBoost was not conceived to deal with data stream. By nature, it does not have the ability of online updating its models during data stream processing. As consequence, it is not prepared to treat concept drift.

However, in [Montiel *et al*., 2020] it was developed a method called AXGB that creates an ensemble of XGBoost which updates the model by replacing one of them by a new one whenever a data stream chunk is processed. The data streams are consumed using an adaptative sliding windows mechanism. To actively deal with the concept drift, the authors propose to use the ADWIN mechanism. The main drawbacks of this work are that it uses an ensemble of ensembles, as XGBoost is already an ensemble, which increased quite a lot its processing time, and it works only to predict binary classification. Classification is the machine learning task devoted to assigning a discrete/categorical label to a new unlabeled instance based on the prediction of a model created with past labeled instances, called training set. Based on the work of [Montiel *et al*., 2020], in [Baldo *et al*., 2022] it was proposed a method called AFXGB that trains a new weak learner of an existing XGBoost for each incoming chunk of data, until it consumes a certain amount of computational resources and, then, substitutes the existing XGBoost model to a new one that started to be trained in the last processed chunks. The AFXGB has the advantage to process the data stream considerably faster than AXGB, but it is also limited to performing binary classification and does not actively treat the concept drift. It means that it does not use any active mechanism such as ADWIN, for instance.

In this work, we propose an extension of the method proposed by [Baldo *et al*., 2022], called Adaptive Fast XGBoost for Multiclass Classification (AFXGB-MC), to classify non-stationary data streams with multiple classes. To do so, it is proposed to substitute the learning objective function and refactor the algorithm entirely to support multiclass classification processing. Also, to deal with concept drift, it is applied an active drift detection algorithm, which implements a reset mechanism to reduce the size of the sliding window. This reset forces the method to intensify the learning process when a concept drift is detected. The proposed method was compared with other algorithms presented in the literature to access its accuracy and processing time performance.

The next sections are organized as follows. Section 2 presents the state-of-the-art learning methods to multiclass classification. Section 3 details the proposed method and its main characteristics. Section 4 assesses the results obtained by the experiments performed over synthetic and real datasets. Finally, section 5, presents the conclusion and future works.

2 Related Works

This section presents an overview of the state-of-the-art presented in the literature concerning multiclass classification algorithms of data streams, highlighting their benefits and drawbacks. It also presents a comprehensive review of the existing approaches to actively deal with concept drift.

The review starts detailing the algorithms proposed to perform multiclass classification of data streams. The first one analyzed is Hoeffding Adaptive Trees (HAT) [Bifet and Gavalda, 2009]. It was developed to improve the Hoeffding Tree by allowing it to adaptively learn from data streams that change over time, without requiring a fixed sliding window size. Additionally, it eliminates the necessity of storing the samples of the current sliding window, which reduces memory consumption. Finally, it uses the ADWIN as drift active detection mechanism.

Another important algorithm is the Adaptive Random Forests (ARF) [Gomes *et al*., 2017]. It is an online bootstrap aggregating algorithm based on Random Forests, which limits each leaf split decision to a subset of features from the data stream. Random Forests, introduced by [Breiman, 2001], is a popular ensemble learning algorithm that creates a tree for each pass through the data using a bootstrap aggregating and a random feature selection approach to avoid overfitting. ARF uses ADWIN and Page Hinkley Test as active drift detection mechanisms.

Leveraging Bagging (LbHT) [Bifet *et al*., 2010] also aims at creating multiple models, each one trained on a different random subset of the training data. Similar to the Random Forest, the idea behind bagging is to reduce the variance of the final prediction by training multiple models on different subsets of the data, and then combining their predictions. LbHT deals with concept drift using the ADWIN mechanism. It can be parallelized, making it suitable for use with large datasets or in distributed computing environments.

Besides that, Oza Bagging (ObHT) [Oza and Russell, 2001] is a variant of the traditional bagging ensemble in which new models are trained online as new data become available. Therefore, in ObHT new data is continually added to the training set and used to update the ensemble of models. This allows the ensemble to adapt to changes in the underlying data distribution and improve its performance over time. It does not use any active drift detection mechanism.

Self Adjusting Memory KNN (SAM-KNN) [Losing *et al*., 2016b] uses a Short-Term Memory (STM) and a Long-Term Memory (LTM) to separate current and past knowledge into dedicated memories with different conservation spans. It combines dedicated models for the current and past concepts to maximize prediction accuracy. It uses k-nearest neighbors (KNN) and assumes that new data is more relevant for current predictions, thus removing conflicting information from past concepts while preserving the rest in a compressed format. It does not use any active drift detection mechanism.

Improved Online Ensembles (IOE) [Vafaie *et al*., 2020] is a multiclass online ensemble algorithm that uses sampling with replacement, while dynamically increasing the weights of underrepresented classes based on recall, to produce models that benefit all classes. It identifies if there is a change in the distribution of the confidence scores, then a flag of drift is set and a new model is trained based on new data. It uses ADWIN as the active drift detection mechanism.

Adaptive eXtreme Gradient Boosting (AXGB) [Montiel *et al*., 2020] performs only binary classification. It creates a new XGBoost model and adds it to the ensemble whenever the dynamic sliding window of data reaches its maximum size. The maximum number of XGBoost classifiers is fixed

and the older classifiers are substituted for new ones when the ensemble is full. It uses ADWIN to reset the sliding window to its minimum size when a drift is detected to speed up the ensemble update. Its main shortcoming is the processing time since it implements an ensemble of XGBoost, which makes the AXGB quite time-consuming.

Adaptive Fast XGBoost for Binary Classification (AFXGB) [Baldo *et al*., 2022] is an algorithm inspired in the AXGB [Montiel *et al*., 2020]. Its main difference is the utilization of only one XGBoost classifier that is updated with new weak learners trained with the data from the stream, and which is replaced by a new XGBoost whenever the former achieves a certain amount of computational resources consumption. It does not use any active drift detection mechanism. It considerably reduces the processing time when compared to the AXGB, keeping the same average accuracy. Its main shortcoming is the reduction in the accuracy when the XGBoost is substituted and the limitation of only performing binary classification.

Regarding concept drift detection mechanisms, ADWIN is one of the most utilized. It divides the data stream into a sequence of sliding windows and looks for a difference between two cuts higher than a threshold [Bifet and Gavaldà, 2007]. When this happens, a drift is detected and a new model is trained based on new data [Lu *et al*., 2019]. The difference is detected by comparing the mean and standard deviation between the two cuts.

KSWIN is a method for detecting concept drift based on the statistical test of Kolmogorov-Smirnov (KS) [Lopes *et al*., 2007]. KSWIN adds the arrived data in a sliding window that has a fixed size, removing the old data when the sliding window is full, and executes the change function, which informs whether the drift is detected or not [Togbe *et al*., 2021]. KSWIN has a constraint to be a one-dimensional method. However, data streams can be multidimensional, so it is important to use a method that supports N-Dimensions data.

Drift Detection Method (DDM) assumes that when the data distribution is stable the model error decreases during the time and vice-versa [Abbaszadeh *et al*., 2015]. It has two stages of detection: the warning level and the change level [Krawczyk *et al*., 2017]. When it detects that the error rate within the window has reached the warning level, it starts building a new learner, but continues using the old learner. If the change reached the concept change level, the old learner is replaced by the new learner [Lu *et al*., 2019].

The Page-Hinckley detector (PHT) is a memoryless, cumulative approach to monitoring learner performance over time. PHT is an adaptation of the detection of abrupt change in the average of a Gaussian signal. It works by accumulating the difference between the observed values and their average until the current moment. Whenever the average exceeds a threshold defined as a parameter, a deviation is signaled [Barddal, 2019].

Regarding the literature review, it is perceived that most of the learning algorithms use ADWIN as the drift detection mechanism. This can be partially explained by the fact that ADWIN, unlike other window-based drift detection algorithms, does not require the window size to be defined as a hyperparameter. This makes ADWIN potentially less sensitive to changes in class imbalance [Vafaie *et al*., 2020].

Table 1 summarizes the main aspects of the related works. For each reviewed algorithm, it is presented its underlying architecture (single or ensemble) and the active drift detection mechanism applied.

Table 1. Underlying architecture of all algorithms considered in the study

Algorithm	Base learner	Drift detection
HAT	Single	ADWIN
ARF	Ensemble	ADWIN and Page Hinkley
LBHT	Ensemble	ADWIN
OBHT	Ensemble	–
SAMKNN	Single	–
IOE	Ensemble	ADWIN
AXGB	Ensemble	ADWIN
AFXGB	Ensemble	–

In the reviewed works, there are quite a few ones that address multiclass classification for data streams. Besides that, most of them are based on ensemble models, which are more time-consuming than single learners. However, [Baldo *et al*., 2022] proposed an algorithm based on the ensemble XGBoost to speed up the binary data stream classification. XGBoost is a prominent ensemble algorithm to be applied due to its fast processing time. Considering the concept drift detection, most of them utilize ADWIN. Therefore, in this work we adapt and extend the AFXGB [Baldo *et al*., 2022] to support multiclass classification, maintaining competitive accuracy with considerably lower processing time.

3 Proposed Method

The proposed method, called AFXGB-MC, explores the benefit of XGBoost concerning fast model induction to perform multiclass classification of data streams. As it can be seen in Figure 1, AFXGB-MC uses a similar approach as in AFXGB [Baldo *et al*., 2022] of replacing the current XGBoost model after a certain period, preventing it from consuming more computational resources than expected, and becoming slow and even overfitted. However, besides that, AFXGB-MC includes a mechanism that resets the size of the sliding window to its minimum when the current model is replaced. This accelerates the model's update, because as shorter as the window is more often the model is updated, which can contribute to keep or even increase its accuracy, since the stream is not stationary and thus can present concept drift. This behavior is presented by arrow numbers 1 and 3 of Figure 1.

Besides that, in order to improve drift detection the proposed method added the ADWIN mechanism. Therefore, AFXGB-MC can actively detect when drifts occur in the non-stationary data stream. When ADWIN detects a concept drift, the size of the sliding window is also reset to its minimum size. By doing this, the algorithm accelerates the model's update, leading to a faster assimilation of the new data distribution. This behavior is presented by arrow number 2 of Figure 1.

Moreover, unlike AFXGB [Baldo *et al*., 2022] that trains a new weak learner once the sliding window is full, AFXGB-

MC updates all the weak learners already trained and belonging to the XGBoost ensemble. With the introduction of this feature not only the last trained weak learner will be aware of the characteristics of the newest data, but also the older weak learners will partially incorporate the characteristics of the newest data.

Finally, to perform the multiclass classification as proposed, the objective function of the XGBoost was changed from binary logistic to multi softmax. Besides that, the evaluation metric was switched from logloss to mlogloss and the hyperparameter number of classes from the data stream was included.

The proposed algorithm does not deal with outliers detection. Indeed, this is a limitation of tree-based learning algorithms, because this class of machine learning algorithms is not tolerant to outliers presented in the training dataset.

3.1 Implementation

Algorithm 1 presents the pseudo-code of the proposed method depicted in Figure 1. It starts adding into the sliding window W the instances that are arriving from the data stream (line 1). Then, it checks if the window has reached its capacity w (line 3). If so, then it is made a copy of the first w instances from window W to window W' (line 4), and the current and next Models are loaded (lines 5 and 6).

After that, it is checked if there is a classifier already built (line 7). If not (line 29), a new XGBoost classifier is trained and saved for later updates (line 30 and 31). However, if there is already a trained XGBoost classifier, if model_update is True (line 25) the existing weak learners are updated (line 26), a new weak learner is trained and the XGBoost classifier is saved for later updates (lines 27 and 28).

Depending on how many windows are processed, it is necessary to start training a new classifier to substitute the current classifier later on. Therefore, when the time of training the current XGBoost classifier overpasses a predefined period of time (line 8), a new XGBoost classifier needs to be trained. If a new XGBoost classifier already exists (line 10), if model_update is True (line 11) the existing weak learners are updated (line 12), a new weak learner is trained and the XGBoost (lines 13) and the new XGBoost classifier is saved for later updates (line 14). Otherwise, a new XGBoost classifier is trained and saved for later updates (lines 16 and 17).

However, if the current classifier reaches its lifetime (line 18), it is replaced by the new classifier (line 20) and the processed sliding window counter is reset (line 22). If the active reset is True (line 23), then the size w of the window is reset to its minimum value (line 24).

Also, for each processed sliding window, if the concept drift detection is on (line 32), the ADWIN verifies if a drift is detected (lines 33 to 35) and then the size w of the window is reset to its minimum value (line 36).

Finally, the sliding window is shifted (line 37), the counter $count$ of processed sliding window is increased (line 38) and the size w is increased if there was not reached the max window size (lines 39 and 40).

Algorithm 1: Adaptative Fast XGBoost for Multiclass Classification – AFXGB-MC

Input: (x, y) ϵ Data Stream
Data: *max_w* = max window size, W = sliding window, *lifetime* = lifetime of each classifier, *training_time* = training time of each classifier, count = count how many sliding window was consumed

1 Adds (x, y) to window (W);
2 $count \leftarrow 0$;
3 **if** $|W| > w$ **then**
4 $W' \leftarrow$ copy the first w^{th} data from W;
5 load_previous_M_model(M); # M can not exist and be NULL
6 load_previous_nextM_model($nextM$); # nextM can not exist and be NULL
7 **if** $M \neq NULL$ **then**
8 **if** *training_time* $\geq$ *(lifetime - count)* **then**
9 # Check if it is inside the training time of next classifier
10 **if** $nextM \neq NULL$ **then**
11 **if** $model\_update = True$ **then**
12 nextM $\leftarrow$ update_nextM(W');
13 nextM $\leftarrow$ train_new_tree_nextM(W');
14 save_nextM(nextM); # Saves nextM model updated;
15 **else**
16 nextM $\leftarrow$ train_new_classifier(W');
17 save_nextM(nextM); # Save new nextM model recently trained;
18 **else if** *count* $\geq$ *lifetime* **then**
19 # Reset count and start to use the new classifier
20 M $\leftarrow$ nextM;
21 nextM $\leftarrow$ NULL;
22 count $\leftarrow$ 0;
23 **if** $active\_reset = True$ **then**
24 w $\leftarrow$ 0;
25 **if** $model\_update = True$ **then**
26 M $\leftarrow$ update_M(W');
27 M $\leftarrow$ train_new_tree_M(W');
28 save_M(M); # Saves M model updated;
29 **else**
30 M $\leftarrow$ train_new_classifier(W');
31 save_M(M); #Save new M model recently trained
32 **if** *detect_drift = True* **then**
33 incorrect_class $\leftarrow$ not(M.predict(X) = y);
34 ADWIN.add(incorrect_class);
35 **if** *ADWIN.drift_detection = True* **then**
36 $w \leftarrow 0$;
37 $W \leftarrow W - W'$;
38 $count \leftarrow count + 1$; # Increase the number of processed sliding window
39 **if** $w < max\_w$ **then**
40 $w \leftarrow w + 1$;
41 returns M;

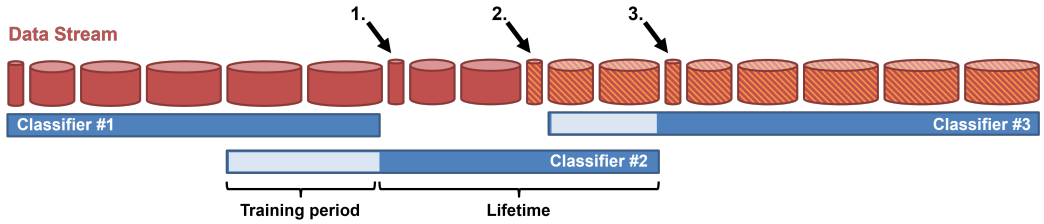

Figure 1. Alternating classifiers method. Dynamic window size is represented by mini-batches and change of color represents concept drift. Arrows 1 and 3 highlight the window size reset caused by the model replacing and arrow 2 highlights the window size reset caused by drift detection.

3.2 Hyperparameters

The proposed algorithm presents some hyperparameters that need to be fine-tuned. A short description of them is presented below.

- **Learning Rate (eta)**: A value between 0 and 1 that states how fast the model will learn from the processed data when a new weak learner is induced.
- **Maximum depth (*max_depth*)**: The maximum depth of a weak learner tree can reach. This value uses to range from 2 to 10.
- **Maximum window size (*max_window_size*)**: An integer value that defines the maximum size of the sliding window that stores the data arrived from the stream.
- **Classifier lifetime (*lifetime*)**: An integer value that defines the lifetime of the current XGBoost model before being substituted.
- **Training time (*training_time*)**: An integer value that defines the training time of each auxiliary classifier. This value uses to range from 60% to 80% of the current XGBoost model's lifetime.
- **Concept drift (*detect_drift*)**: A boolean value that determines whether the ADWIN is used.
- **Number of classes (*num_classes*)**: An integer value that determines how many classes are in the data stream.

4 Results Assessment

This section describes the results of the experiments performed using AFXGB-MC. The proposed method was compared with the algorithms state-of-the-art stated in the literature and presented in section 2. This section is organized as follows. Subsection 4.1 outlines the methodology used to perform the experiments, subsection 4.2 details used datasets, subsection 4.3 shows the assessment results of the different flavors of AFXGB-MC and, finally, subsection 4.4 presents the assessment comparison regarding accuracy and processing time between the algorithm proposed in this work and the state-of-the-art algorithms from the literature.

4.1 Assessment Methodology

The assessment of accuracy and processing time between AFXGB-MC and the algorithms from the literature wants to identify if the proposed algorithm can create accurate classifiers with lower processing time. However, first, it is necessary to select a comprehensible and robust bunch of benchmark datasets to perform the assessments. We selected some of the most utilized datasets in the literature revised in section 2. More details about the selected datasets can be found in subsection 4.2.

AFXGB-MC algorithm has some features that can be switched on or off and which impact its performance. Therefore, it is necessary to identify the most suitable configuration of AFXGB-MC algorithm to use in the comparison tests. This analysis was performed with the same benchmark datasets selected previously. This assessment is detailed in subsection 4.3.

Having the most suitable AFXGB-MC configuration defined, it is performed the accuracy and processing time assessment between AFXGB-MC and the state-of-the-art algorithms in the literature. AFXGB-MC was compared with other six multiclass classification algorithms for data streams, which are: ARF [Gomes *et al*., 2017], IOE [Vafaie *et al*., 2020], HAT [Bifet and Gavalda, 2009], LBHT [Bifet *et al*., 2010], OBHT [Oza and Russell, 2001] and SAMKNN [Losing *et al*., 2016a]. This comparison analysis is presented in subsection 4.4.

The evaluation method selected is the Prequential Evaluation, where the data are analyzed sequentially as they arrive through the stream. The experiments were executed 5 times for each algorithm and dataset to obtain the average accuracy and processing time. The hyper-parameters chosen for ARF, IOE, HAT, LBHT, OBHT and SAMKNN were set based on the recommendations of the respective literature. The hyper-parameters of AFXGB-MC were defined after a broad set of empirical experiments which presented the following values: Number of Estimators = 30, Learning Rate = 0.3, Max Window Size = 10000, Max Tree Depth = 6, Lifetime = 25, Training time = 15.

The experiments with all the assessed algorithms were executed in the following environment: Intel® Core™ i7-11700 (8 cores, 16MB cache, up to 4.9GHz); with 32GB (2 of 16GB) of memory (DDR4, UDIMM, non-ECC memory); with 512 GB of M.2 Class 40 SSD storage; running Ubuntu 20.04.4 LTS (CLI Server); Linux 5.4.0-113-generic. The AFXGB-MC algorithm was implemented in Python 3, the IOE implementation was obtained with the authors [Vafaie *et al*., 2020] and the implementation of the other assessed algorithms are available in the Scikit-Multiflow library of Python [Scikit-Multiflow, 2020a].

4.2 Datasets

To perform the experiments, we select a broad set of synthetic and real datasets. The usage of synthetic datasets intends to execute controlled experiments concerning the as-

sessment of concept drift detection and recovery. On the other hand, the usage of real datasets aims to assess the performance of the proposed algorithm in real-case scenarios. The synthetic datasets were created using stream generators available in the Scikit-Multiflow library of Python [Scikit-Multiflow, 2020b]. The real datasets were extracted from UCI Machine Learning Repository [Dua and Graff, 2017].

Table 2 summarizes the datasets selected to execute the experiments. The synthetic datasets used are generated with the following functions: LED, Random RBF and SEA. For each generation function were created two synthetic datasets, one with gradual concept drift and another with abrupt concept drift. The drifts were positioned every 100.000 instances. The real datasets used are the following: Gas Sensor, Poker Hand, KDD Cup 1999, Covertype and Winnipeg [Dua and Graff, 2017]. Both synthetic and real datasets were selected due to their wide application in the experiments of the literature reviewed. The execution of the experiments over the selected datasets was run 5 times because the whole bunch of 5 executions with 7 algorithms in 11 datasets spent more than 14 days.

Table 2. Datasets used on the experiments. Concept drifts: Abrupt (A) and Gradual (G).

Dataset	Instances	Attributes	Classes	Real
LED_A	500000	24	3	no
LED_G	500000	24	3	no
RBF_A	500000	10	3	no
RBF_G	500000	10	3	no
SEA_A	500000	3	2	no
SEA_G	500000	3	2	no
GAS	13910	128	6	yes
POKER	1000000	11	10	yes
KDDCUP99	311029	41	23	yes
COVERTYPE	581012	54	7	yes
WINNIPEG	325834	175	7	yes

4.3 AFXGB-MC Sensitivity Analysis

AFXGB-MC algorithm has two features that can be switched on or off and may bring benefits in certain data stream processing scenarios. These features are the Drift Detection (D-act./D-inact), which determines if the ADWIN is executed to identify a concept drift, and the Model's Update (U-act./U-inact.), which determines if the weak learners inside the XGBoost classifier are updated with the new data coming from the stream when active.

Considering that there are two features with two possible states, we have four different AFXGB-MC setup options to analyze. Therefore, to assess which is the most competitive setup among the four possible ones, we perform a sensitivity analysis of them using the 11 datasets shown in Table 2.

Table 3 presents the average accuracy for 5 executions of the four different setups. The highest accuracy for each dataset is highlighted, and the algorithm's corresponding ranking is placed next to its accuracy. The ranking is calculated by ordering the accuracy values of all algorithms for a given dataset, where rank 1 represents the algorithm with the highest accuracy and rank 4 with the lowest one.

To ensure whether there is a statistically significant difference in the results presented by the assessed setups of AFXGB-MC algorithm, we performed the Friedman Test. Considering a significance value of 95%, the Friedman test presented a p-value of 0.0065, which rejects the null hypothesis. This means that the differences in the accuracies produced in the experiments are statistically significant. However, to determine if there is a significant difference among the rankings we applied the Nemenyi post-hoc pair-wise statistical test. Nemenyi calculates the CD (Critical Difference), which indicates that whether two algorithms have a ranking difference greater than the CD, it can be assumed that they have different rankings. For the ranks presented in Table 3, the Nemenyi test resulted in a CD of 0.63. The pairwise comparisons among the four setups of AFXGB-MC are shown in Figure 2.

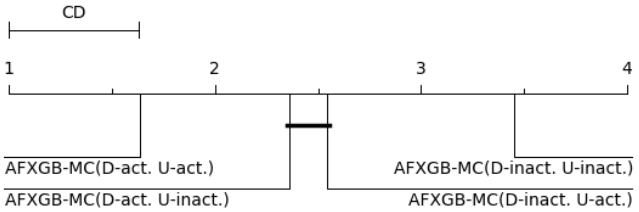

Figure 2. Nemenyi test of the four AFXGB-MC flavors

As can be seen in Figure 2, the Nemenyi calculated with the ranking of the four AFXGB-MC setups indicates that AFXGB-MC with Active Drift Detection (D-act.) and Active Model's Update (U-act.) has the best ranking, with a significant difference. AFXGB-MC with Active Drift Detection (D-act.) and Inactive Model's Update (U-inact.), and with Inactive Drift Detection (D-inact.) and Active Model's Update (U-act.) did not present significant differences. Finally, AFXGB-MC with Inactive Drift Detection (D-inact.) and Inactive Model's Update (U-inact.) presented the worst performance, with a significant difference. Therefore, AFXGB-MC with D-act. and U-act. is the selected setup to be used in the comparison with the algorithms from the literature.

4.4 Algorithms Comparative Analysis

This section presents the comparison analysis performed between AFXGB-MC and the state-of-the-art algorithms for multiclass classification encountered in the literature. The selected flavor of AFXGB-MC was with D-act. and U-act. It was compared with six different algorithms, which are the state-of-the-art concerning multiclass classification of data stream. Section 4.4.1 presents the accuracy analysis and section 4.4.2 presents the processing time analysis of such comparison.

4.4.1 Accuracy Analysis

Table 4 presents the average accuracy for 5 executions of each algorithm assessed algorithm over the 11 datasets selected. Next to the accuracy value, it is presented inside parenthesis the algorithm's ranking for each dataset. The

Table 3. Average accuracies and rankings of the four AFXGB-MC flavors over the 11 datasets

Dataset	D-act. and U-act.	D-act. and U-inact.	D-inact. and U-act.	D-inact. and U-inact.
RBF_A	0.8883 (2)	**0.8894** (1)	0.8558 (3	0.7725 (4)
RBD_G	0.8797 (2)	**0.8819** (1)	0.8627 (3)	0.7753 (4)
SEA_A	**0.9904** (1)	0.9898 (2)	0.9793 (3)	0.9406 (4)
SEA_G	**0.9859** (1)	0.9845 (2)	0.9788 (3)	0.9613 (4)
LED_A	0.3592 (2)	0.3215 (3)	**0.3602** (1)	0.3186 (4)
LED_G	0.3581 (2)	0.3215 (3)	**0.3609** (1)	0.3188 (4)
GAS	**0.7260** (1)	0.6827 (2)	0.4658 (4)	0.5820 (3)
POKER	0.6142 (3)	0.5792 (4)	**0.6419** (1)	0.6324 (2)
KDDCUP99	**0.9872** (1)	0.9834 (2)	0.8391 (3)	0.2139 (4)
COVERTYPE	**0.8246** (1)	0.7908 (2)	0.7324 (3)	0.6130 (4)
WINNIPEG	0.9867 (2)	0.9842 (4)	0.9857 (3)	**0.9870** (1)
Avg. acc.	0.7818	0.7644	0.7330	0.6469
Avg. rank	1.6	2.3	2.5	3.4

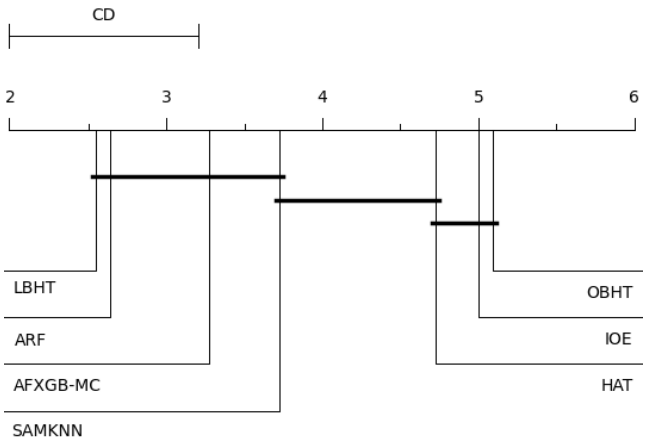

Figure 3. Nemenyi test of the algorithms average rankings

rank ranges from 1 to 7, where 1 represents the algorithm with the highest accuracy and 7 with the lowest.

The Friedman test considering a significance value of 95%, presented a p-value of 6.85×10^{-7}, which rejects the null hypothesis and ensure that the experiments are statistically significant. However, to identify if there are significant differences regarding the average accuracy of the algorithms, it was applied the Nemenyi test over the algorithms' average ranking, resulting in a CD of 1.21. The pair-wise comparison among the algorithms is shown in Figure 3.

The Nemenyi test presented in Figure 3 shows that LBHT, ARF, AFXGB-MC and SAMKNN did not present a statistically significant difference in their accuracy ranking because their distances are smaller than the CD. However, they presented a better accuracy ranking than HAT, IOE and OBHT. This result means that AFXGB-MC can achieve similar results as the more accurate algorithms presented in the literature. Therefore, AFXGB-MC can substitute any of the 6 assessed algorithms keeping competitive performance in terms of accuracy.

When analyzing in detail the average accuracies presented in Figure 3, it can be seen that LBHT and ARF had almost the same average accuracy and average raking. However, LBHT had two best accuracies while ARF had just one. AFXGB-MC had also two best accuracies. Nevertheless, SAMKNN had four best accuracy performances, being the algorithm with more best accuracies among them.

Also analyzing Figure 3, concerning only the synthetic datasets, LBHT, SAMKNN and HAT had two best accuracies, which means that they are more suitable to process such kind of dataset. However, when it is considered the real datasets, AFXGB-MC and SAMKNN had two best accuracies, while ARF had one and LBHT had none. This means that AFXGB-MC and SAMKNN are more suitable to process real data streams, which is a competitive difference, as processing real data is more useful in practice.

4.4.2 Processing Time Analysis

Table 5 presents the average processing time in seconds of the 5 executions of each algorithm in processing all the 11 datasets. This table presents the training and the testing times separately and then sums up both to present the total processing time. The separation of training and testing time aims to provide a more detailed analysis because it is possible to see in which part of the process the algorithm is spending more time.

Figure 4 graphically presents the average processing time of the 7 algorithms in a cumulative bar chart, where in blue is shown the average training time and in red the average testing time. As can be seen, AFXGB-MC had the shortest average total time of all algorithms. When compared with SAMKNN, which had the second shortest processing time, AFXGB-MC is 2.3 times faster. If compared with LBHT, the algorithm with the best accuracy rank, AFXGB-MC is 51.5 times faster. When compared with ARF, the second best accuracy algorithm, AFXGB-MC is 22 times faster.

AFXGB-MC spends around 25% of the processing time with testing, similar to SAMKNN and IOE, which spend around 23% and 25% of the time doing tests. However, the algorithm that spends more time in testing is OBHT, taking around 39% of the time in testing. ARF spends the lowest testing time, (8%), followed by LBHT (9%) and HAT (16%).

Therefore, concerning processing time, AFXGB-MC is the fastest algorithm, just having as competitor with similar accuracy the SAMKNN. Even though, SAMKNN is around twice slower than AFXGB-MC. The other algorithms with similar accuracy had processing times too much slower than AFXGB-MC, which can difficult their utilization in data stream scenarios where the data arriving rate is too fast.

Table 4. Average accuracies and rankings for the 7 algorithms assessed over the 11 datasets

Dataset	LBHT	ARF	AFXGB-MC	SAMKNN	HAT	IOE	OBHT
RBF_A	0.8957 (2)	0.8556 (5)	0.8883 (3)	**0.9289** (1)	0.7675 (7)	0.8812 (4)	0.8475 (6)
RBD_G	0.8934 (2)	0.8521 (5)	0.8797 (3)	**0.9266** (1)	0.7809 (7)	0.8790 (4)	0.8436 (6)
SEA_A	**0.9951** (1)	0.9941 (2)	0.9904 (3)	0.9819 (5)	0.9867 (4)	0.9657 (6)	0.9545 (7)
SEA_G	**0.9903** (1)	0.9889 (2)	0.9859 (3)	0.9771 (5)	0.9832 (4)	0.9609 (6)	0.9519 (7)
LED_A	0.3364 (5)	0.3480 (3)	0.3592 (2)	0.1964 (7)	**0.3638** (1)	0.2973 (6)	0.3410 (4)
LED_G	0.3318 (5)	0.3443 (3)	0.3581 (2)	0.1939 (7)	**0.3581** (1)	0.2975 (6)	0.3402 (4)
GAS	0.8793 (3)	0.9697 (2)	0.7260 (5)	**0.9710** (1)	0.5555 (7)	0.7785 (4)	0.5580 (6)
POKER	0.5854 (2)	0.5373 (3)	**0.6142** (1)	0.4928 (6)	0.5195 (5)	0.3474 (7)	0.5308 (4)
KDDCUP99	0.9989 (3)	**0.9992** (1)	0.9872 (7)	0.9990 (2)	0.9964 (5)	0.9939 (6)	0.9980 (4)
COVERTYPE	0.9290 (3)	0.9406 (2)	0.8246 (7)	**0.9513** (1)	0.8276 (6)	0.9027 (4)	0.8740 (5)
WINNIPEG	0.9813 (2)	0.9813 (2)	**0.9867** (1)	0.5991 (6)	0.9572 (5)	0.9765 (3)	0.9611 (4)
Avg. acc.	0.8015	0.8010	0.7818	0.7471	0.7360	0.7528	0.7455
Avg. rank	2.5	2.6	3.2	3.7	4.7	5	5.1

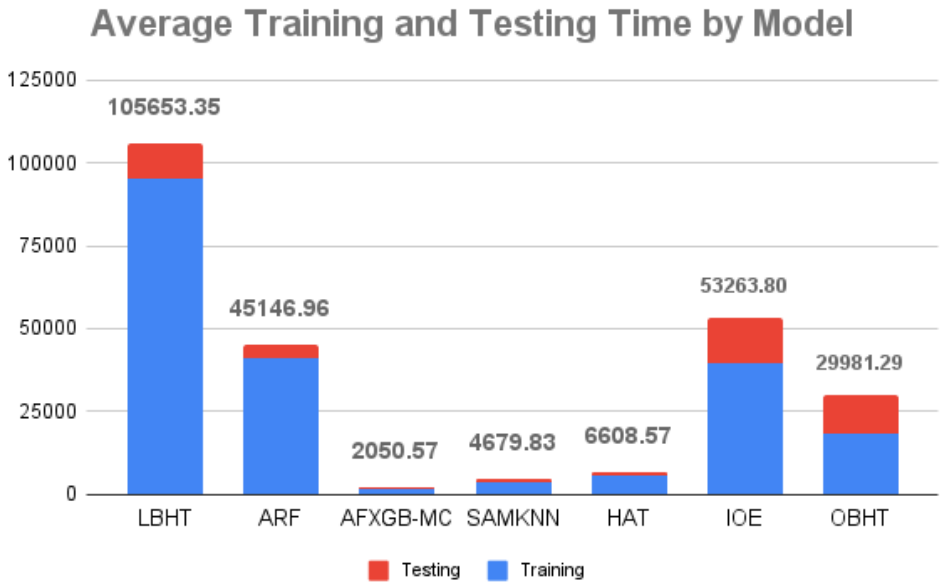

Figure 4. Average training and testing time of all algorithms (in Seconds).

Table 5. Average algorithms' processing time.

	Time (seconds)		
Algorithm	**Avg. training**	**Avg. testing**	**Avg. total**
LBHT	95349.84	10303.51	105653.35
ARF	41342.81	3804.15	45146.96
AFXGB-MC	1533.67	516.90	2050.57
SAMKNN	3594.41	1085.42	4679.83
HAT	5488.29	1120.28	6608.57
IOE	39675.44	13588.36	53263.8
OBHT	18184.37	11796.92	29981.29

4.4.3 Concept Drift Recovery Analysis

To assess the ability of the models to recover from concept drift, we conducted an analysis of their accuracy recovery after each drift. Figure 5a shows the current accuracy of the models at the valuation of each arrived instance for the simulation of the RBF_A dataset. Therefore, axis X presents the number of analyzed instances and axis Y the accumulated accuracy values of the models. As can be observed, AFXGB-MC demonstrates a fast recovery in accuracy, being the second faster algorithm to be recovered in the four simulated abrupt drifts, losing only to SAMKNN. This can be seen in more detail in Figures 5b and 5c, which highlight the second and fourth drifts, respectively.

Table 6 shows the accuracy of all models in four data points related to the second concept drift of the RBF_A dataset. Before the drift, at 200k all models have their highest accuracy. After the drift, at 200.2k there is a significant decrease in accuracy of all, where SAMKNN was the one with less decrement. LBHT, ARF, AFXGB-MC and SAMKNN were almost fully recovered at 201.4k. At 201.6k, LBHT, ARF, AFXGB-MC and SAMKNN were recovered, unlike HAT, IOE and OBHT which were still trying to be recovered. Besides that, AFXGB-MC had a slightly better recovery than SAMKNN.

Table 6. Accuracy of models before and after the 2nd concept drift of RBF_A dataset.

Data	LBHT	ARF	AFXGB-MC	SAMKNN	HAT	IOE	OBHT
200k	0.93	0.915	0.915	0.935	0.845	0.915	0.92
200.2k	0.245	0.34	0.225	0.63	0.325	0.27	0.17
201.4k	0.815	0.78	0.86	0.92	0.61	0.715	0.445
201.6k	0.83	0.815	0.905	0.91	0.59	0.705	0.46

In contrast, Table 7 displays the accuracy of all models

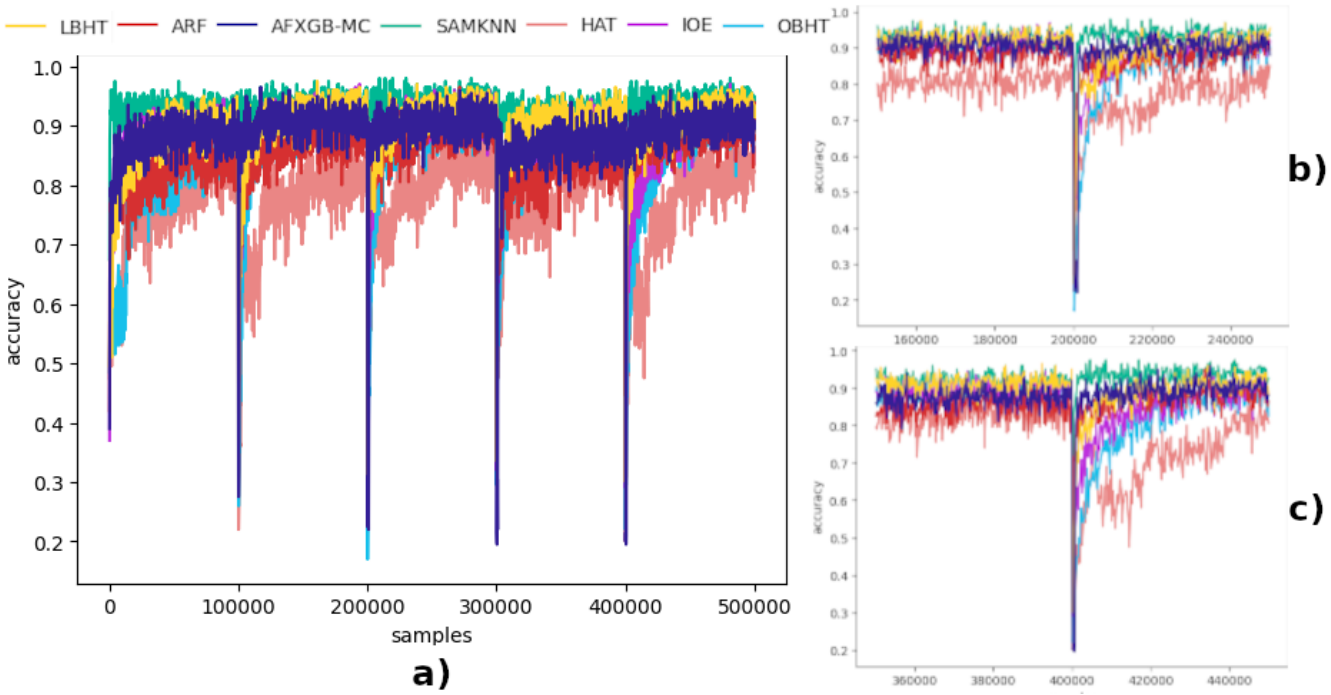

Figure 5. In a) accuracy over time for all models on the RBF_A dataset. In b) and c) accuracies of 2nd and 4th concept drifts.

in four data points related to the fourth concept drift of the RBF_A dataset. Again, all models have their maximum accuracy before the drift at 400k. At 400.2k they presented a significant loss of accuracy due to the occurrence of drift, having SAMKNN with the least decrement. SAMKNN has already recovered from the drift by 402.2k, while AFXGB-MC recovered from the drift at 403.4k, and the other algorithms are still recovering. In this scenario again AFXGB-MC was the second fastest model to recover from the drift.

Table 7. Accuracy of models before and after the 4th concept drift of RBF_A dataset.

Data	LBHT	ARF	AFXGB-MC	SAMKNN	HAT	IOE	OBHT
400k	0.95	0.89	0.92	0.935	0.875	0.91	0.915
400.2k	0.3	0.445	0.2	0.6	0.2	0.29	0.22
402.2k	0.81	0.84	0.89	0.925	0.54	0.7	0.57
403.4k	0.77	0.855	0.925	0.94	0.635	0.78	0.58

5 Conclusion

The analysis of data streams is getting even more attention with the augment of 5G and IoT usage. However, processing such kinds of data is not a trivial task, especially when considering that they are continuous, endless and non-stationary. To deal with these challenges, learning algorithms should analyze the data just once, discarding them after, update their classifiers to handle the non-stationary behavior of the stream, and also detect when a concept drift occurs.

The literature presents some algorithms that can deal with the classification of multiclass data streams. However, most of them are slow, prejudicing their utilization in the processing of fast data streams. Therefore, this work proposed the AFXGB-MC to fast classify non-stationary data streams with multiple classes. It uses XGBoost as basis classifier, extending the approach proposed by [Baldo *et al.*, 2022] to support multiclass classification. Additionally, the ADWIN drift detection algorithm is used to reset the sliding window size to intensify the learning process when a concept drift is detected.

To assess the proposed algorithm, we compared it with the six state-of-the-art algorithms for multiclass classification found in the literature. The results pointed out that AFXGB-MC has similar accuracy to the best three assessed algorithms, which are LBHT, ARF and SAMKNN, with statistical significance. Also, it presented a low processing time, being 2.3 times faster than SAMKNN, 22 times faster than ARF and 51.5 times faster than LBHT. Finally, it is the second fastest algorithm to recover from an abrupt concept drift.

Therefore, as can be seen, AFXGB-MC presents similar results when considering accuracy performance, compared with the six most suitable algorithms found in the literature. However, it is quite faster than those algorithms, presenting at least a processing time twice faster than the second fastest algorithm, and with fast drift recovery time. Therefore, in scenarios with fast data generation, such as in sensing networks, with milliseconds of collecting period, algorithms like AFXGB-MC can provide competitive advantages

As future works, it is possible to improve even more the learning approach by avoiding substituting the current classifier by applying a slicing in the classifier to exclude the older weak learners and thus limiting its growth. Besides that, it is planned to adapt this algorithm to the analysis of data stream regression.

Funding

We would like to specially thanks FAPESC – Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina – to partially funded this research work.

References

Abbaszadeh, O., Amiri, A., and Khanteymoori, A. R. (2015). An ensemble method for data stream classification in the presence of concept drift. *Frontiers of Information Technology & Electronic Engineering*, 16:1059–1068.

Aggarwal, C. C. (2006). *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag, Berlin, Heidelberg.

Baldo, F., Grando, J., Weege, K., and Bonassa, G. (2022). Adaptive fast xgboost for binary classification. In *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 13–25, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbbd.2022.224291.

Barddal, J. P. (2019). Vertical and horizontal partitioning in data stream regression ensembles. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. DOI: 10.1109/IJCNN.2019.8852244.

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

Bifet, A., Gavalda, R., Holmes, G., and Pfahringer, B. (2018). *Machine learning for data streams: with practical examples in MOA*. MIT press.

Bifet, A. and Gavaldà, R. (2007). *Learning from Time-Changing Data with Adaptive Windowing*, pages 443–448. DOI: 10.1137/1.9781611972771.42.

Bifet, A., Holmes, G., and Pfahringer, B. (2010). Leveraging bagging for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 135–150. Springer.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2939672.2939785.

Deng, Z., Zhu, X., Cheng, D., Zong, M., and Zhang, S. (2016). Efficient knn classification algorithm for big data. *Neurocomputing*, 195:143–148. Learning for Medical Imaging. DOI: https://doi.org/10.1016/j.neucom.2015.08.112.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Ferreira, A. J. and Figueiredo, M. A. T. (2012). *Boosting Algorithms: A Review of Methods, Theory, and Applications*, pages 35–85. Springer US, Boston, MA. DOI: 10.1007/978-1-4419-9326-7_2.

Fields, T., Hsieh, G., and Chenou, J. (2019). Mitigating drift in time series data with noise augmentation. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 227–230.

Gama, J. and Gaber, M. M. (2007). *Learning from Data Streams: Processing Techniques in Sensor Networks*. 1 edition.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., and Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., and Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156. DOI: https://doi.org/10.1016/j.inffus.2017.02.004.

Lopes, R. H., Reid, I., and Hobson, P. R. (2007). The two-dimensional kolmogorov-smirnov test.

Losing, V., Hammer, B., and Wersing, H. (2016a). Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 291–300. DOI: 10.1109/ICDM.2016.0040.

Losing, V., Hammer, B., and Wersing, H. (2016b). Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 291–300. DOI: 10.1109/ICDM.2016.0040.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363. DOI: 10.1109/TKDE.2018.2876857.

Montiel, J., Mitchell, R., Frank, E., Pfahringer, B., Abdessalem, T., and Bifet, A. (2020). Adaptive XGBoost for evolving data streams. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. DOI: 10.1109/IJCNN48605.2020.9207555.

Oza, N. C. and Russell, S. J. (2001). Online bagging and boosting. In *International Workshop on Artificial Intelligence and Statistics*, pages 229–236. PMLR.

Santhanam, R., Raman, S., Uzir, N., and Banerjeeb, S. (2016). Experimenting XGBoost algorithm for prediction and classification of different datasets. *International Journal of Control Theory and Applications*, 9(40).

Schapire, R. (1999). A brief introduction to boosting. In *IJCAI-99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, v. 1 & 2*, pages 1401–1406.

Scikit-Multiflow, A. (2020a). Scikit-multiflow – ensemble methods api reference.

Scikit-Multiflow, A. (2020b). Scikit-multiflow – stream generators api reference.

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. P. L. F. d., and Gama, J. a. (2013). Data stream clustering: A survey. *ACM Comput. Surv.*, 46(1). DOI: 10.1145/2522968.2522981.

Togbe, M. U., Chabchoub, Y., Boly, A., Barry, M., Chiky, R., and Bahri, M. (2021). Anomalies detection using isolation in concept-drifting data streams. *Computers*, 10(1). DOI: 10.3390/computers10010013.

Vafaie, P., Viktor, H., and Michalowski, W. (2020). Multi-class imbalanced semi-supervised learning from streams through online ensembles. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pages 867–874. DOI: 10.1109/ICDMW51313.2020.00124.