On Ensemble Techniques for Data Stream Regression

Heitor Murilo Gomes*, Jacob Montiel*, Saulo Martiello Mastelini†, Bernhard Pfahringer* and Albert Bifet*‡
* Department of Computer Science, University of Waikato, Hamilton, New Zealand
Email: {heitor.gomes,jmontiel,bernhard,abifet}@waikato.ac.nz
† Institute of Mathematics and Computer Sciences, University of São Paulo, São Carlos, Brazil
Email: mastelini@usp.br
‡ LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

***Abstract*—An ensemble of learners tends to exceed the predictive performance of individual learners. This approach has been explored for both batch and online learning. Ensembles methods applied to data stream classification were thoroughly investigated over the years, while their regression counterparts received less attention in comparison. In this work, we discuss and analyze several techniques for generating, aggregating, and updating ensembles of regressors for evolving data streams. We investigate the impact of different strategies for inducing diversity into the ensemble by randomizing the input data (resampling, random subspaces and random patches). On top of that, we devote particular attention to techniques that adapt the ensemble model in response to concept drifts, including adaptive window approaches, fixed periodical resets and randomly determined windows. Extensive empirical experiments show that simple techniques can obtain similar predictive performance to sophisticated algorithms that rely on reactive adaptation (i.e., concept drift detection and recovery).**

***Index Terms*—data streams, regression, ensemble, random patches, random subspaces**

I. Introduction

The application of machine learning to data streams has grown in importance in recent years due to a large amount of real-time data generated by networks, mobile phones, and the wide variety of sensors currently available. Building predictive models from data streams are central to many applications. One example is the Internet of Things (IoT) applications, where connected sensors yield a large amount of data in short periods. To build predictive models from streaming data, we need to either settle for traditional offline learning or employ algorithms capable of learning incrementally. A significant setback with the offline learning approach is that it is slow to react to changes in the domain, and these changes can have a catastrophic impact on the predictive model performance since the patterns in which the model was trained on are no longer valid.

Often, the application of a single decision model may lead to subpar performance in online scenarios, given the previously mentioned challenges. As a consequence, algorithms that combine several models, i.e. ensemble methods, are a trend for supervised learning for both static and streaming data. Ensembles enable leveraging the power of multiple learners towards the same goal, whereas alleviating their individual limitations.

Ensemble learning has been thoroughly investigated for data stream classification [1]. Consequently, several methods were proposed [2], [3] to this end. Recently, more methods were proposed for data stream regression, such as the Adaptive Random Forest Regressor (ARF-Reg) [4]. There are at least three relevant aspects to be considered when proposing an ensemble learner, either for regression or classification: combination, generation, and the update dynamics. The combination (or voting) strategy describes how the individual predictions are aggregated to obtain the ensemble prediction. For classification, a common method is majority vote, while for regression, the mean is commonly used. The generation method defines how the base models are trained, commonly including some mechanism to enforce diversity among the base learners. A traditional approach is to train learners on different subsets of instances (e.g., Bagging [5]), features (e.g., the Random Subspaces Method [6]) or both (e.g., Random Patches [7] and Random Forests [8]). The update dynamics is fundamental when dealing with streaming data, specifically evolving data streams, as it defines how (and when) base models will be reset or updated to reflect changes to the underlying data distribution.

These three aspects of ensemble learning were thoroughly investigated for data stream classification. For example, the Adaptive Random Forest (ARF) for classification algorithm [3] includes an empirical comparison between majority vote against a weighted majority vote. Conversely, the regression version of ARF, namely ARF-Reg [4], presents only results considering a simple linear combination (the mean) of the predictions. Therefore, there is room for the investigation of other combination techniques, the impact of different reset strategies to deal with concept drifts, and how to generate diverse learners for a regression problem.

The main contributions of this work are the following:

- We discuss and analyze several techniques to train base learners, combine their predictions and update them actively (or reactively) to address concept drifts.
- We benchmark existing algorithms and the proposed ensemble variants using 17 datasets. This lead us to insightful conclusions, such as the high performance obtained by relatively simple models (e.g., k Nearest Neighbors) in comparison to ensemble models that implement complex

drift detection and recovery.
- We provide empirical evidence that indicate that using reactive strategies to adapt to concept drifts might not be necessary given an ensemble where base learners are trained using windows of varying sizes, thus reset at different time intervals.

The remainder of this work is organized as follows. In Section II, we review ensemble methods for data stream regression and classification. Section III contains the description of strategies to build, combine and update ensemble models designed for evolving data stream regression. Section IV, introduces the four sets of experiments that provide an insightful analysis of the presented ensemble strategies. Finally, in Section V we present our concluding remarks and directions for future work.

II. Related work

The **Fast and Incremental Model Trees (FIMT-DD)** [9] is the most widely used algorithm to build incremental regression trees for streaming data. Similarly to the Hoeffding Tree algorithm [10], FIMT-DD starts with an empty tree that keeps statistics from arriving data until a grace period is reached. The features are ranked according to their variance w.r.t the target variable to decide for splits, and if the two best-ranked differ by at least the Hoeffding Bound [11], the node splits. FIMT-DD includes a change detection scheme that periodically flags and adapts subbranches of the tree where significant variance increases are observed. In [12], the authors propose the **Online Regression Trees with Options (ORTO**) algorithm, that introduces 'option' nodes, which allow an instance to follow all the branches available in a tree node. The **Adaptive Model Rules (AMRules)** [13] learns both an ordered and an unordered rule set from a data stream. To detect and adapt to concept drifts, each rule is associated with a Page-Hinkley drift detector [14], which prunes the rule set given changes in the incoming data.

The development of ensembles of regressors attracted less attention than ensembles of classifiers for streaming data, even though there is a sizeable amount of literature on this topic for batch learning [15]. For streaming data, Ikonomovska et al. [16] proposed the **online random forest (ORF)** and **online bagging (OBag)** ensembles that use the FIMT-DD as the base learner. Based on empirical experiments, the authors concluded that the ORTO-A (online option trees with averaging) outperformed both OBag and ORF in terms of mean squared error (MSE). More recently, the **Adaptive Random Forest regressor (ARF-Reg)** [4], an adaptation of the data stream classifier [3] of the same name was proposed. ARF-Reg builds a forest of FIMT-DD trees as ORF, the main difference between both algorithms is that ARF-Reg employs one instance of the Adaptive WINdow (ADWIN) algorithm [17] per tree to detect concept drifts. The way in which randomization is added during model generation in a random forest is particular to decision trees. A more general approach is to use the random subspaces method [6] as in **Heuristic Updatable Weighted Random Subspaces (HUWRS)** [18] and **Streaming Random Patches (SRP)** [19] algorithms. SRP trains each base learner on a subset of features and instances from the original data, namely a random patch [7]. This strategy to enforce diverse base models is similar to the one in the random forest, yet it is not restricted to using decision trees as base learner. Moreover, in [19] the overall results (in terms of accuracy) for SRP outperformed the adaptive random forest (ARF) [3] in a multitude of datasets.

In this work, we introduce techniques that can be applied to ensembles of regressors for streaming data. We focus on four aspects: combination, generation, base learner, and reset strategies. For generation, we explore techniques that induce a diverse set of base models by training them on different subsets of instances (bagging), features (random subspaces) or both (random patches). To combine the predictions, we investigate the benefits of using the median instead of the mean. We also investigate the impact of different base learners (incremental trees and k-nearest neighbors) or variations of these (i.e., adapting the leaves of the tree). Finally, we challenge the well-established strategy of resetting base learners according to some drift detection algorithm [3], [4], [19] against simpler strategies, such as fixed windows of varying sizes.

III. Ensemble strategies for data stream regression

In simple terms, an ensemble learner is a set of base models and an integration method to combine their predictions. When applied to stream data, it may also incorporate some reset dynamics to adapt the ensemble to potential changes in the data distribution. There is a vast literature concerning strategies to improve ensemble classifiers for streaming data [1], yet not as many approaches have been thoroughly investigated for ensembles of regressors. In this section, we discuss strategies that can potentially leverage an ensemble learner for data stream regression.

A. Generation - Training and Diversity Induction

There are different approaches for generating (training) base models. The motivation for the development of such strategies is to enforce diversity into the ensemble. If all the base models make homogeneous predictions, it is clear that their combination is no better than just using one of them. Many algorithms provide mechanisms to induce diversity implicitly by training each base model on different subsets of the data. Canonical examples include bagging [5], the random subspaces method [6], random forests [8], and random patches [7]. These techniques were successfully adapted and applied to data stream classification [2], [3], [18], [19], and some of them to regression [4], [16]. We employ generation techniques that do not rely on a specific algorithm as the base learner. Precisely, we explore Random Subspaces (RS), Bagging (BAG)[1], and Random Patches (RP). These generation techniques enforce that each base model is trained with different subsets of instances (BAG), features (RS) or both (RP).

[1]We refer to it simply as BAG. However, we are in fact using the resampling strategy introduced in [20] where Poisson($\lambda = 6$) is used instead of Poisson($\lambda = 1$), as in the original Online Bagging adaptation by Oza [2].

On top of that, these are techniques that modify the training data presented to each base learner without explicitly changing the base learner algorithm (e.g., random forests manipulate the individual tree construction algorithm). A similar approach is explored in [19] for data stream classification.

B. Combination

We analyze two strategies for aggregating the predictions, the mean and the median. Aggregating the ensemble prediction as the mean of its members' predictions is a simple and often effective strategy used in a multitude of algorithms [15]. One drawback of the mean as a measure of central tendency is that it can be affected by any single value that is either too high or too low in comparison to the rest of the sample. In our context, it is desirable to avoid situations where a single model prediction can have a potentially harmful influence on the overall prediction. For example, after resetting a base model, the relatively 'new' model may produce predictions that deviate too far from others, simply because it has not been trained on a sufficiently large amount of data. However, the other way around can be true as well, i.e., a single learner's prediction positively influence the overall combined vote. To investigate further we propose comparing the mean aggregation against a median aggregation. Using the median to aggregate ensemble predictions was used in the Bragging (bootstrap robust aggregating) algorithm, a variant of Bagging proposed by Bühlmann in [21].

C. Reset Strategy

The reset strategy in an ensemble, designed to cope with evolving data streams, is an utterly important component. The strategies for maintaining learning algorithms up-to-date even when faced with concept drifts are often categorized as explicit (or reactive) and implicit (or proactive). Reactive methods rely on an algorithm (i.e., drift detector) that indicates the need to reset the base models, while active methods constantly resets the base models according to some predefined strategy. Reactive strategies are popular and often employed alongside ensemble methods for data stream classification [3], [22] and regression [4], [9].

We now describe three strategies for resetting the ensemble model, depicted in Figure 1. All of them follow the same principles of replacing an active base model with a model that has been trained without influencing the ensemble decisions (namely, a background model). The intuition behind training a model before adding it to the ensemble is to avoid an underfitted model interfering with the ensemble predictions. The differences among these strategies lie on how they determine the start of the training for the background model (t_s) and the replacement (t_r) of the current model.

- **Adaptive Window.** A drift detection algorithm monitors the error of each base learner, and whenever a drift is signaled the associated base learner is reset (thus ending its training window);
- **Fixed Window.** The length of each training window is predefined, such that each model is reset after reaching the maximum number of instances.
- **Random Window.** This approach adds another level of randomization by determining the window length of each learner randomly.

The **adaptive window** depends on the change detection algorithm used to identify potential drifts (warnings) and actual drifts. We use the ADaptive WINdow (ADWIN) [17] algorithm, but other detectors could be used as well. The error of each base model is monitored by a different instance of ADWIN. A warning signal determines when to start training the background model (t_s), and a drift signal determines when to replace the model (t_r). Effectively, t_s and t_r are automatically set, but the detection algorithm itself adds some hyperparameters (i.e., the confidence λ for ADWIN), thus it is not entirely automatic. This strategy has been explored in multiple works for both regression [4] and classification [3], [19].

In the **fixed window** strategy, we avoid simultaneously resetting the base models by adding a small shift to the hyperparameters t_s and t_r, such that the training of the background model and the replacement of the current model are slightly different for each base model. This way, we also avoid replacing all members of the ensemble at the same time, which effectively represents resetting the ensemble. We can associate t_s and t_r with the warning and drift detection from the adaptive window. This strategy is similar to how data is buffered to sequentially train batch learners in the Fast and Slow Framework [23], where batch and stream learning methods operate together.

To generate **random windows** that are neither too short nor too large, we constraint the length of the window l to be a positive integer approximately in the interval $l \in (t_0 - 1/W, t_0 + 1/W)$. Given hyperparameters t_0 and W, the probability that the window of training ends after observing t instances is given by Eq. 1.

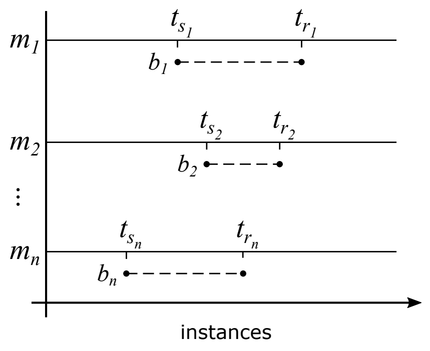
$$Pr[reset] = 1/(1 + e^{-4(t-t_0)/W}) \tag{1}$$

As the number of instances observed t approaches $t_0 - W/2$, the probability that a reset is triggered starts increasing, reaching $Pr[reset] = 0.5$ when $t = t_0$, and approaching $Pr[reset] = 1.0$ after $t_0 + W/2$. Other similar approaches, such as uniformly choosing a random number between $t_0 - W/2$ and $t_0 + W/2$, could be used, but we remark that they would produce similar results.

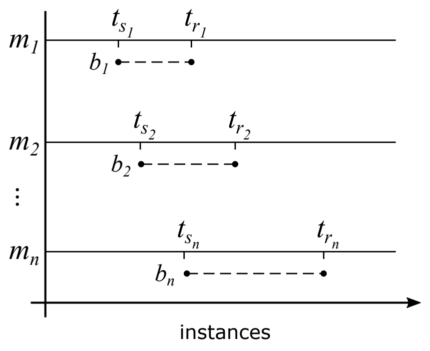
The question that we want to answer with both the fixed and random window approaches is:

Does signaling when to reset base models using an accurate drift detector positively influences the predictive performance of the ensemble, OR the positive effects are caused by periodically resetting the base learners?

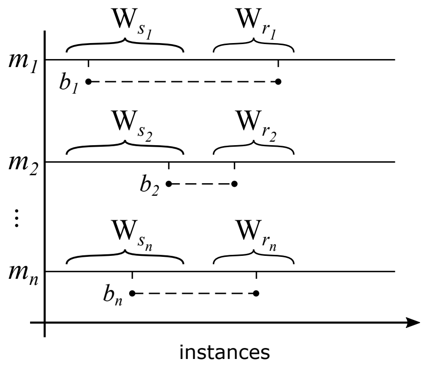
In [24], the authors presented a similar empirical study to verify the relevance that a drift detector plays on a classification system. The conclusions obtained by the authors were



(a) Adaptive Window.



(b) Fixed Window.



(c) Random Window.

Fig. 1: Reset model strategies. Let m_i denote the i-th model in the ensemble, in all strategies a background model b_i starts training at t_{s_i} and it replaces m_i at time t_{r_i}. After a model is replaced, the reset process starts again. In Fig. 1a the ADWIN drift detector is used, t_{s_i} and t_{r_i} are set according to changes in the performance of a model i as detected by ADWIN. In Fig. 1b a fixed-size window is used between t_{s_i} and t_{r_i}. We vary the size of such window for each member i of the ensemble, to ensure that background models are trained on different number of instances and to avoid replacing all members of the ensemble at the same time. In Fig. 1c two adjacent windows W_{s_i}, W_{r_i} are defined. These windows correspond to the range in which the training of the background model starts (t_{s_i}) and when the model replacement takes place (t_{r_i}). Both events happen at random within the corresponding window.

that a fine-tuned fixed window length was able to overcome a system that relies on an adaptive window length (drift detector). The caveat is that it is not trivial to determine the window length ahead of time in an optimal way. However, one may want to avoid too short or too large (or infinite) window lengths. Windows that are too short do not allow the base model to learn any concept effectively, resulting in suboptimal performance. Conversely, windows that are too large imply the risk of keeping old concepts within the ensemble.

D. Base learner

We experiment with two popular regression algorithms as the base learners for the ensemble variations. The first is the Hoeffding Tree Regressor (HTR), which is a variation to its classifier counter-part by Domingos and Hulten [10]. Similarly to FIMT-DD [9], HTR split decisions are based on the variance information, and the aggregation at the leaves can either be performed by a linear model (i.e., a perceptron) or the mean target values of examples reaching the leaf. Nonetheless, HTR does not include mechanisms for concept drift adaptation as FIMT-DD. This fact, though making HTR not coping with non-stationary distributions when working standalone, shall improve its computation resource usage over FIMT-DD. The second algorithm is k nearest neighbors (KNN). KNN is a common baseline for both classification and regression. The basic kNN regressor searches for the k instances that are closer (w.r.t. a given distance metric) to an instance whose target value has to be predicted. The predicted value is the unweighted mean of the k nearest instances found. KNN is known to be a stable learner, i.e., given a small variation in the training sample for two KNN models, their predictions will be fairly similar. There are different approaches to enforce instability to KNN models, such as injecting randomness to the distance metrics [25] or using different random subspaces to build the training samples [26].

It was observed in [16] and [19] that as Hoeffding trees grow larger they become more similar w.r.t. their predictions. We shall observe a similar behavior if KNN is employed with bagging. In this work, we refrain from applying adaptations to the base models with the sole intention of inducing diversity as we lack the space for appropriate analysis and discussion of such an important topic. Furthermore, we rely on ensemble generation strategies to enforce diverse predictions for KNN and HTR, precisely by using random subspaces and random patches.

IV. Experiments

For every experiment we apply a test-then-train evaluation strategy, i.e., each instance is used for testing and then used for training. We analyze how the learning algorithms performs in terms of Root Mean Square Error (RMSE) in different scenarios including real and synthetic data. There are 17 datasets used in the experiments, including real (7) and synthetic datasets (10), such as Hyperplane (Hyper) and Radial Basis Function (RBF) variations. We use three variations of Hyper and RBF synthetic datasets, each of them simulating a different type of drift. Synthetic datasets variants identified with (G) and (A) simulates gradual and abrupt drifts every $125K$ instances (i.e., $125K$, $250K$ and $375K$). The window of change for abrupt drifts is 1, and $20,000$ for gradual drifts. Variants (I) simulate incremental concept drifts. The summary statistics of the datasets are shown in Table I.

Some hyperparameters were fixed throughout all experiments. The ensemble variants (e.g., ARF-Reg) were executed with 30 base learners and a subspace size of 60%. Most of the base learners are based on variants of a Hoeffding

TABLE I: Characteristics of the evaluated datasets. Simulated Drifts: (A) Abrupt, (G) Gradual, (I) Incremental. The first group (top) contains the real-world datasets, and the second group (bottom), the synthetic datasets.

Dataset	#Instances	#Numeric features	#Categorical features
Abalone	4977	7	1
Bike	17379	12	0
CalHousing	20500	8	0
House8L	22784	8	0
House16H	22784	16	0
MetroTraffic	48204	4	3
Pol	15600	48	0
Ailerons	13750	40	0
Elevators	16599	18	0
Fried	40768	10	0
MVDelve	40967	7	3
Hyper(A)	500000	10	0
Hyper(G)	500000	10	0
Hyper(I)	500000	10	0
RBF(A)	500000	20	0
RBF(G)	500000	20	0
RBF(I)	500000	20	0

Tree, including the HTR and FIMT-DD. In these cases, we used a configuration that has been successful for ensemble classifiers, i.e., a grace period of 50 and a higher confidence ($\delta = 0.01$). BAG, RP, and RS, their variants and base learners, were executed and implemented[2] in *scikit-multiflow* [27]. The remaining considered algorithms are available in *MOA* [28]. Unless otherwise indicated, all the algorithms were performed using their standard hyperparameters, according to their implementations.

To facilitate the identification of the ensemble variants, we introduce the following naming convention. **Generation:** random patches (RP), bagging (BAG), random subspaces (RS); **Combination:** mean (μ), median (med); **Reset strategy:** adaptive window (a), fixed window (f), random window (r); **Base learner:** Hoeffding Tree regressor with mean leaves (HTR$_m$); Hoeffding Tree regressor with perceptron leaves (HTR$_p$); k-Nearest Neighbors regressor with mean aggregation (KNN). For example, when identifying a variant of Random Patches using Mean aggregation, Adaptive Window reset strategy, and HTR with perceptron at the leaves, we write RP$^a_\mu$-HTR$_p$.

Our goal is to present and discuss the impact of the strategies discussed in Section III, as well as compare some of them against existing algorithms. It is infeasible to report all possible configurations due to the large number of combinations. Therefore, we organize the experiments in four groups to balance a breadth analysis with an in-depth analysis. In the first set of experiments we analyze the impact of the Generation and Combination strategies. We follow that experiment with an analysis of the impact of the base learner to some of the ensemble variations and how they compare against single instances of the base leaner algorithms. The third experiment was designed to answer the question posed in section III, which challenged the importance of hybrid solutions that combined drift detectors to the ensemble (i.e., adaptive window reset strategies). Finally, the last set of experiments compare some of the ensemble variants against algorithms from the literature.

[2]https://github.com/jacobmontiel/StreamingRandomPatchesRegressor

A. Generation and Combination

To verify the impact of both the generation and combination strategies we present the experiments in Table II. For this analysis, we fixed the base learner as HTR$_m$. We observe that the BAG variants obtain the best results overall. Therefore, for the given datasets, it is not possible to conclude that improvements could be observed in terms of RMSE when employing random subspaces as part of the generation process. One possible cause can be that the majority of the datasets contain mostly relevant features, i.e., by building models on subsets of the feature set, it is not possible to obtain reasonably accurate models, which in turn negatively impacts the aggregated predictions. In general, the median combination approach was not more accurate than the mean, which shows that even though the mean is a less stable measure of central tendency, it does not seem to influence the ensemble performance negatively. Therefore, the hypothesis that extreme values may lead the combination astray could not be observed in these experiments. A possible explanation is the type of base learner, i.e., HTR$_m$, produced predictions that are more stable than the predictions from HTR$_p$.

TABLE II: Generation and combination analysis (generation=[BAG | RP | RS], combination=[mean (μ) | median (med)], base learner=HTR$_m$), reset strategy=adaptive (a).

Dataset	BAG$^a_\mu$	BAG$^a_{\text{med}}$	RP$^a_\mu$	RP$^a_{\text{med}}$	RS$^a_\mu$	RS$^a_{\text{med}}$
Abalone	2.5911	2.6346	**2.5230**	2.5794	3.0465	3.0629
Bike	**81.0924**	82.0560	92.1240	89.1398	100.9140	98.1813
CalHousing	**63000.7071**	63447.0723	66238.8810	65327.8682	72371.6388	73694.5632
House8L	**36357.7785**	37425.4647	36674.6710	37662.6646	37238.0024	38038.3599
House16H	**39807.9324**	40503.2810	40974.8801	41893.5338	41366.8500	41694.6820
MetroTraffic	**1864.3562**	1878.4567	1868.4516	1881.3474	1909.7701	1926.9327
Pol	**39.8561**	40.9659	40.0456	40.9348	42.5239	43.0442
Ailerons	**0.0000**	0.0000	0.0000	0.0000	0.0000	0.0000
Elevators	0.0048	0.0050	0.0048	0.0049	**0.0046**	0.0047
Fried	2.8504	**2.8436**	3.3891	3.4347	4.7477	5.1362
MVDelve	2.7978	**2.3556**	3.4039	2.8376	7.2215	7.4087
Hyper(A)	**4.7393**	4.7695	5.1514	5.2270	5.6156	5.7496
Hyper(G)	**4.7314**	4.7739	5.1578	5.2315	5.6322	5.7745
Hyper(I)	**73.8283**	75.1720	75.1034	75.5369	75.7321	76.3416
RBF(A)	24.3243	26.4591	**22.6989**	23.8939	29.6176	29.9555
RBF(G)	**24.5419**	26.2645	24.9022	26.5333	29.6681	29.9662
RBF(I)	29.2339	29.2461	**29.1266**	29.1732	30.4916	30.6660
Avg. rank	**1.59**	2.88	2.59	3.65	4.65	5.65
Avg. rank real	**1.29**	3.00	2.57	3.71	4.71	5.71
Avg. rank synth.	**1.80**	2.80	2.60	3.60	4.60	5.60

B. Base learners

To analyze the impact of the base learner we compare variations of BAG$^a_\mu$ with KNN, HTR$_m$ and HTR$_p$. On top of that, we also present the stand-alone results for these three algorithms with the purpose of presenting a clear baseline (i.e., it is not reasonable to use an ensemble if a single model is more accurate). The results are depicted on Table III. We highlight that KNN obtained a low RMSE for many datasets, and in overall it was on pair with BAG$^a_\mu$-HTR$_p$. It was not possible to leverage the good individual results of KNN in BAG$^a_\mu$-KNN. One possible explanation is that KNN is a stable learner and just by slightly changing the subset of instances being used by each of the models it was unable to produce better results. In fact, by comparing KNN and BAG$^a_\mu$-KNN, they are quite similar and often the best result is in favor of KNN. When comparing BAG$^a_\mu$-HTR$_m$ and BAG$^a_\mu$-HTR$_p$ against HTR$_m$ and HTR$_p$ we can observe that the ensemble models were able to outperform a single base learner. In general, using the perceptron improves the performance in comparison to using the mean to aggregate the predictions at the leaves. This fact can be observed when comparing HTR$_m$ and HTR$_p$, as well as their ensemble versions.

TABLE III: Base learner analysis (generation=BAG, combination=mean (μ), learner=[KNN | HTR$_m$ | HTR$_p$], reset strategy=adaptive (a)).

Dataset	KNN	HTR$_m$	HTR$_p$	BAG$^a_\mu$-KNN	BAG$^a_\mu$-HTR$_m$	BAG$^a_\mu$-HTR$_p$
Abalone	**2.3264**	3.0540	2.8726	2.3268	2.5911	2.5719
Bike	**62.3067**	108.0877	85.1775	62.3440	81.0924	69.0090
CalHousing	89876.9806	85204.2071	72327.5408	90212.1338	**63000.7071**	63307.2304
House8L	51046.4284	40883.3315	40874.6389	51026.8462	36357.7785	**35860.5305**
House16H	51164.6143	43890.7514	44301.9024	51151.9541	**39807.9324**	40028.5144
MetroTraffic	1945.0847	1951.2740	1910.3972	1945.8638	1864.3562	**1858.1606**
Pol	**18.2383**	26.4236	26.5224	23.1435	39.8561	36.1823
Ailerons	**0.0000**	0.0000	0.0000	0.0000	0.0000	0.0000
Elevators	0.0069	0.0054	0.0052	0.0068	**0.0048**	0.0048
Fried	2.6746	2.7908	2.8103	**2.6731**	2.8504	2.9802
MVDelve	8.3187	3.8745	3.8762	8.4289	**2.7978**	3.6996
Hyper(A)	**3.0400**	5.9205	5.0287	9.7130	4.7393	4.5308
Hyper(G)	**3.3039**	5.9094	5.0714	9.3744	4.7314	4.5717
Hyper(I)	50.9431	79.8120	54.4025	**50.9248**	73.8283	65.4014
RBF(A)	17.9073	23.2324	20.4353	17.9242	24.3243	**14.8843**
RBF(G)	18.7157	23.1230	20.5220	18.7317	24.5419	**14.5837**
RBF(I)	29.2988	**28.0289**	28.0496	29.2921	29.2339	28.4256
Avg. rank	3.06	4.18	3.65	3.94	3.47	**2.71**
Avg. rank real	3.43	4.57	3.86	3.86	2.86	**2.43**
Avg. rank synth.	**2.80**	3.90	3.50	4.00	3.90	2.90

C. *Reset strategy*

We apply three techniques to continuously reset the base models, and, thus, keep the ensemble up-to-date with the latest concepts. Each of these techniques are highly influenced by their hyperparameters, which directly influence how many instances will be used for training each base model before it is reset. We experimented with three variations of hyperparameters for the fixed and random windows, alongside a version that never resets the base models (no-reset), and one that uses an ADWIN change detector (adaptive window). More details about these variations are presented below.

- **fixed (f$_s$) and random (r$_s$) small.** Trigger background learner creation: $t_s = 400$; Trigger replace: $t_r = 700$; random window: $W_s = W_r = 200$
- **fixed (f$_m$) and random (r$_m$) medium.** Trigger background learner creation: $t_s = 1500$; Trigger replace: $t_r = 2500$; random window: $W_s = W_r = 800$
- **fixed (f$_l$) and random (r$_l$) large.** Trigger background learner creation: $t_s = 2500$; Trigger replace: $t_r = 5000$; random window: $W_s = W_r = 2000$
- **adaptive.** Trigger background and replace according to the drift detector.
- **no-reset.** The base models are never reset.

One of the goals of the fixed and random reset strategies was to avoid resetting the base models simultaneously. The fixed window reset learners at different times and with different window sizes, the hyperparameters only define the length of the ‘first’ ensemble member, the others have increasing window lengths. Similarly, the random window strategy reset learners with about the same window size, but at slightly different times (depending on hyperparameter W).

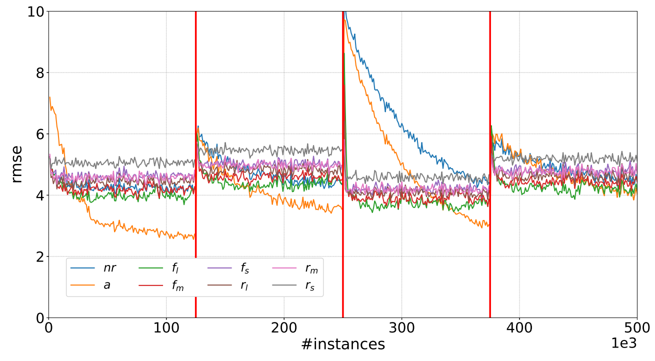
Table IV presents the results for the different reset strategies. We observe that using larger windows, for both fixed and random, improve the overall results. Analyzing the RMSE over time in Figure 1, it is noticeable that the fixed and random windows tend to adapt to concept drifts fast and without major (and long) variations to the average RMSE. This can be attributed to the fact that the base learners are reset at different times, which generates a mix of learners trained only on the latest concept and learners trained on a larger window. A counter-intuitive result is that no-reset outperforms the adaptive strategy for most of the synthetic datasets with simulated concept drifts in Table IV. Complementing the analysis with the plots from Figure 1, we can observe that the adaptive (a) variation closely resembles the no-reset results, and often recovers from concept drifts faster (Figures 2a and 2b). The ability to adapt to new concepts even if the base learners are never reset is justified by the use of no-reset with Hoeffding trees that are allowed to keep growing indefinitely. Even though this allows the trees to adapt to new concepts, it applies a heavy toll on the computational resources. The best average rankings are obtained when using a fixed window and the ‘large’ parametrization (f$_l$), which has a reasonable compromise between smaller and longer windows, i.e., a configuration in-between ‘small’ (f$_s$) and no-reset (nr).

D. *Comparison against other algorithms*

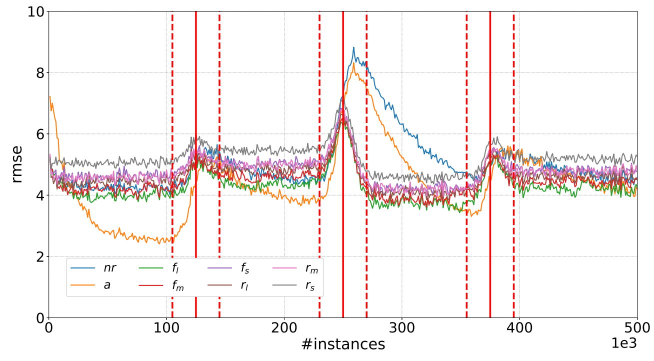
In Table V, we compare two variations of the ensemble techniques discussed in this paper against algorithms from the literature. Precisely, we use BAG$^{\text{f-l}}_\mu$-HTR$_p$ and BAG$^a_\mu$-HTR$_p$, which differ only on the reset strategy used. From these experiments we highlight that ARF-Reg tends to outperform all others in the synthetic datasets, including those with simulated concept drifts, while BAG$^{\text{f-l}}_\mu$-HTR$_p$ obtains the best results for the real datasets. We highlight that, contrary to what was observed in the experiments varying the reset strategy, the ARF-Reg algorithm, which includes an active drift detection strategy, was able to outperform other methods in the synthetic datasets that simulate concept drifts. However, if we compared it against the no-reset (n$_r$) from Table IV, it would not differ much in terms of RMSE. We also replicate the results for KNN in Table V to highlight how well it performs in comparison to algorithms specially designed to address evolving data streams, such as FIMT-DD, ORTO, and AMRules.

TABLE IV: Reset strategy (generation=RP, combination=median, learner=HTR$_p$, reset strategy=fixed | no-reset | random | adaptive (a)).

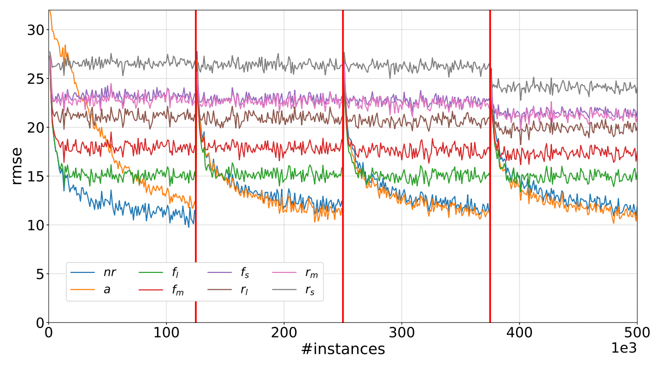
Dataset	f$_s$	f$_m$	f$_l$	no-reset	r$_s$	r$_m$	r$_l$	adaptive
Abalone	2.2405	2.2851	2.2804	2.2813	**2.2154**	2.2592	2.2870	2.2862
Bike	92.7782	81.9534	72.1119	**67.0228**	108.2110	100.1860	93.8778	73.4341
CalHousing	61709.5065	63947.9028	65687.8560	65809.6456	**60076.6916**	62724.3415	62055.3426	66349.6414
House8L	39343.1472	37711.8951	37184.1753	**37137.5179**	41328.1239	39208.1627	38596.7054	37718.3326
House16H	43781.4843	42412.0779	42122.2324	43091.4831	45354.9294	43583.8446	42921.1098	**41855.8577**
MetroTraffic	1811.5875	1824.2094	1842.6123	1852.6115	**1782.2871**	1811.0393	1818.1665	1859.2225
Pol	25.7078	**22.3910**	22.8605	23.1437	29.9345	25.1886	24.1786	24.8740
Ailerons	**0.0000**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Elevators	0.0054	0.0052	0.0051	**0.0050**	0.0054	0.0054	0.0054	0.0050
Fried	3.3059	3.0930	2.9059	**2.5176**	3.6136	3.2882	3.1752	3.0728
MVDelve	4.2186	3.3576	3.3580	4.7204	5.2824	4.1712	3.9506	**2.5276**
Hyper(A)	4.6818	4.3225	**4.1641**	5.1618	5.0834	4.6458	4.5177	4.9436
Hyper(G)	4.8392	4.4764	**4.2935**	5.1692	5.2234	4.8038	4.6746	4.9526
Hyper(I)	51.9050	**51.4357**	52.1003	67.0243	53.0485	51.6913	51.6470	67.2361
RBF(A)	22.6324	17.8684	15.4194	**13.5661**	25.8346	22.3086	20.7128	14.8983
RBF(G)	22.9855	18.3157	15.7911	**13.6283**	26.0139	22.6718	21.1338	15.2901
RBF(I)	28.8979	28.7884	28.7029	**28.3937**	29.0184	28.8918	28.8548	28.5633
Avg. rank	5.12	3.29	**3.00**	3.94	6.29	5.12	4.76	4.47
Avg. rank real	4.71	3.86	**3.43**	4.14	5.00	4.86	4.86	5.14
Avg. rank synth.	5.40	2.90	**2.70**	3.80	7.20	5.30	4.70	4.00



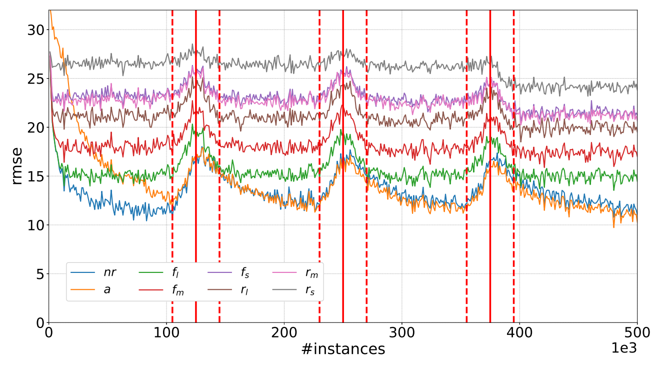
(a) Hyper(A).



(b) Hyper(G).



(c) RBF(A).



(d) RBF(G).

Fig. 2: RMSE over time for varying reset strategies.

V. Conclusion

Ensembles are a popular approach in supervised learning since they improve performance by leveraging the predicting capabilities of a group of weak learners. Regression for evolving data streams, although relevant to many real-world applications and posing specific challenges, has not received as much attention by the research community as classification. In this paper, we study ensemble techniques for regression and show that, although performance is improved, special considerations must be taken in the context of regression, e.g., combination techniques that integrate well with the base learner. To this end, we focused our analysis on techniques for training the base learners, combining predictions, the role of base learners, and the reset strategy that provides robustness

TABLE V: Comparing BAG$^{a}_{\mu}$-HTR$_p$ and BAG$^{\text{f-1}}_{\mu}$-HTR$_p$ against others.

Dataset	FIMT-DD	ORTO	AMRules	ARF-Reg	KNN	HTR$_m$	HTR$_p$	BAG$^{a}_{\mu}$-HTR$_p$	BAG$^{\text{f-1}}_{\mu}$-HTR$_p$
Abalone	2.6227	8.3230	2.3284	2.8277	2.3264	3.0540	2.8726	2.5719	**2.2506**
Bike	572.2625	2882.6485	134.5418	93.5983	**62.3067**	108.0877	85.1775	69.0090	68.3269
CalHousing	77589.2502	141419.9559	72436.8602	64253.7315	89876.9806	85204.2071	72327.5408	63307.2304	**62820.2290**
House8L	40945.7784	84042.0749	41388.1129	36325.3640	51046.4284	40883.3315	40874.6389	**35860.5305**	35966.3236
House16H	46798.5857	96237.2919	46072.4447	**39435.5565**	51164.6143	43890.7514	44301.9024	40028.5144	39461.4041
MetroTraffic	18719714.8607	6017208.9625	8798.4883	**1762.3839**	1945.0847	1951.2740	1910.3972	1858.1606	1842.0460
Pol	50.3320	90.6362	25.9851	**17.8487**	18.2383	26.4236	26.5224	36.1823	18.9284
Ailerons	0.0037	0.0070	0.0020	0.0002	**0.0000**	0.0000	0.0000	0.0000	0.0000
Elevators	0.3380	0.0715	0.0047	**0.0046**	0.0069	0.0054	0.0052	0.0048	0.0049
Fried	2.7390	7.8746	2.4735	**2.2410**	2.6746	2.7908	2.8103	2.9802	2.3569
MVDelve	2.9448	12.0426	3.8574	**1.5152**	8.3187	3.8745	3.8762	3.6996	2.1918
Hyper(A)	**1.8803**	15.8049	1.9713	3.3463	3.0400	5.9205	5.0287	4.5308	3.6148
Hyper(G)	**2.2675**	15.9225	2.3780	3.6790	3.3039	5.9094	5.0714	4.5717	3.7817
Hyper(I)	48.2369	126.0124	50.6482	**48.0818**	50.9431	79.8120	54.4025	65.4014	48.3118
RBF(A)	17.2946	57.5298	23.0847	**13.9592**	17.9073	23.2324	20.4353	14.8843	15.6089
RBF(G)	17.3575	58.0759	22.9520	14.9155	18.7157	23.1230	20.5220	**14.5837**	15.9212
RBF(I)	29.3239	38.9952	29.9269	28.3527	29.2988	**28.0289**	28.0496	28.4256	28.6819
Avg. rank	5.47	8.88	5.06	**2.59**	4.76	6.00	5.29	4.00	2.94
Avg. rank real	7.00	8.86	5.57	2.86	4.86	5.86	4.86	3.29	**1.86**
Avg. rank synth.	4.40	8.90	4.70	**2.40**	4.70	6.10	5.60	4.50	3.70

against concept drifts. We conclude that resetting the base models has a positive effect in the predictive performance. Based on the experiments, we notice that a reactive strategy (based on a drift detector) may not produce the best results all the time. Simpler reset strategies such as periodically replacing members of the ensemble with new models trained on different windows can also boost performance in the ensemble. Another relevant observation was that random subspaces and random patches were not as effective for regression as when they were applied for classification.

For future works, we are considering a further analysis of ensembles of k-Nearest Neighbors for regression, and how to minimize the impact in the computational resources caused by an unbounded growth of the Hoeffding Tree algorithms.

References

[1] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM CSUR*, 2017.

[2] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *ACM SIGKDD*, 2001.

[3] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfahringer, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, 2017.

[4] H. M. Gomes, J. P. Barddal, L. E. B. Ferreira, and A. Bifet, "Adaptive random forests for data stream regression." in *ESANN*, 2018.

[5] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[6] T. K. Ho, "Random decision forests," in *International Conference on Document Analysis and Recognition*, vol. 1. IEEE, 1995.

[7] G. Louppe and P. Geurts, "Ensembles on random patches," in *ECML*. Springer, 2012, pp. 346–361.

[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] E. Ikonomovska, J. Gama, and S. Džeroski, "Learning model trees from evolving data streams," *Data mining and knowledge discovery*, vol. 23, no. 1, pp. 128–168, 2011.

[10] P. Domingos and G. Hulten, "Mining high-speed data streams," in *ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

[11] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, 1963.

[12] E. Ikonomovska, J. Gama, B. Zenko, and S. Dzeroski, "Speeding-up hoeffding-based regression trees with options," in *ICML*. Citeseer, 2011, pp. 537–544.

[13] E. Almeida, C. Ferreira, and J. Gama, "Adaptive model rules from data streams," in *ECML*. Springer, 2013, pp. 480–492.

[14] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi, "Test of page-hinckley, an approach for fault detection in an agro-alimentary production system," in *Asian Control Conference*, vol. 2, 2004.

[15] J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. D. Sousa, "Ensemble approaches for regression: A survey," *Acm computing surveys (csur)*, vol. 45, no. 1, p. 10, 2012.

[16] E. Ikonomovska, J. Gama, and S. Džeroski, "Online tree-based ensembles and option trees for regression on evolving data streams," *Neurocomputing*, vol. 150, 2015.

[17] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *SIAM international conference on data mining*, 2007, pp. 443–448.

[18] T. R. Hoens, N. V. Chawla, and R. Polikar, "Heuristic updatable weighted random subspaces for non-stationary environments," in *ICDM*. IEEE, 2011.

[19] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *ICDM*. IEEE, 2019.

[20] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *PKDD*, 2010, pp. 135–150.

[21] P. L. Bühlmann, "Bagging, subagging and bragging for improving some prediction algorithms," in *Research Seminar für Statistik, Eidgenössische Technische Hochschule (ETH)*, vol. 113, 2003.

[22] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 249–260.

[23] J. Montiel, A. Bifet, V. Losing, J. Read, and T. Abdessalem, "Learning fast and slow: A unified batch/stream framework," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1065–1072.

[24] A. Bifet, "Classifier concept drift detection and the illusion of progress," in *ICAISC*. Springer, 2017, pp. 715–725.

[25] Z.-H. Zhou and Y. Yu, "Adapt bagging to nearest neighbor classifiers," *Journal of Computer Science and Technology*, vol. 20, no. 1, pp. 48–54, 2005.

[26] T. K. Ho, "Nearest neighbors in random subspaces," in *IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition*. Springer, 1998.

[27] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *Journal of Machine Learning Research*, vol. 19, no. 72, 2018.

[28] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *The Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.