Adaptive Fast XGBoost for Binary Classification

Fabiano Baldo[1], **Julia Grando**[1], **Kawan M. Weege**[1], **Gustavo M. Bonassa**[1]

[1]Departamento de Ciência da Computação
Programa de Pós-Graduação em Computação Aplicada
Universidade do Estado de Santa Catarina (UDESC)
Joinville – SC – Brazil

fabiano.baldo@udesc.br, juliagrando@gmail.com,

kawanweege@gmail.com, gustavo.bonassa1@gmail.com

Abstract. *Modern machine learning algorithms must be able to fast consume data streams, maintaining accurate results, even with the presence of concept drift. This work proposes AFXGB, an Adaptive Fast binary classification algorithm using XGBoost, focusing on the fast induction of labeled data streams. AFXGB uses an alternate model training strategy to achieve lean models adapted to concept drift. We compared AFXGB with other data stream classifiers using synthetic and real datasets. The results showed that AFXGB is four times faster than ARF and 22 times faster than AXGB, maintaining the same accuracy and with the fastest recovery from concept drifts, thus preserving long-term accuracy.*

1. Introduction

Due to the rapid development of computer technologies, an immense volume of data is generated in real-time by several people and systems. Adding this to the broad use of the Internet of Things and smart devices, we witness the shaping of the so-called Big Data, a mass of information characterized by the volume, variety, and speed with which the data is presented [1]. In this scenario, data is increasingly being used for machine learning, a technique that is applied in several real-world classification tasks [2, 3].

For some of these tasks, it is possible to process the data asynchronously, but many others require a real-time response [4], forcing the classification model to handle data in stream format. This means that current data instances should be classified at high speed, before the arrival of new instances, in order to avoid delays or data loss. In addition to the quickness of response, it should be taken into account that the data pattern can be dynamic, meaning that over time the relationship between classes and features can change [5, 6]. So besides the need of being fast, the model should be updated at a certain frequency to cope with concept drifts and maintain high accuracy.

Boosting algorithms are a popular method for classification problems, which combine weak learners' solutions iteratively to obtain a better overall prediction result [7, 8]. This combination of multiple learners is called ensemble learning. One of the most popular ensemble libraries is the eXtreme Gradient Boosting (XGBoost) [9] algorithm, which uses decision trees as weak learners, creating each individual tree in parallel, decreasing training time [10].

In this work, we propose the Adaptive Fast XGBoost (AFXGB) model as a binary classifier of data streams. We compare its speed and accuracy with other data stream

classifiers, showing that AFXGB has an overall faster time performance while preserving the accuracy and recovering earlier from concept drifts.

The paper is organized as follows: Section 2 presents an overview of data stream classifiers, Section 3 describes the proposed algorithm and datasets used for simulation, and on Section 4 we discuss the experiments and results. Finally, Section 5 highlights the conclusion and future work.

2. Related Works

There are some related works with an emphasis on data stream classification using the framework *XGBoost* and other kinds of decision trees. This section presents the summary of the literature review that was carried out.

Typical classification models are not fit for working with data streams because of the large volume of data and changes in concept over time, also called concept drifts. To avoid accuracy drops when concept drifts occur, the approach of using sliding-windows of data was developed [11]. The sliding-window model aids in the task of keeping the model up to date by discarding old data and considering only the last N elements for analysis and training [12].

The Hoeffding Tree Classifier [13] is an incremental decision tree algorithm capable of learning from data streams. It assumes that the distribution of classes does not change over time, using a small sample to choose an optimal splitting attribute. This strategy is supported mathematically by the Hoeffding bound, which quantifies the number of samples needed to estimate a statistical attribute within a certain precision. Using Hoeffding Trees as a basis, it was proposed an incremental decision tree inducer for data streams that can deal with concept drift called Hoeffding Adaptive Tree (HAT) [14]. It uses change detectors and estimator modules to adapt automatically to the data on the stream, not needing user-defined parameters.

Based on Random Forests, an algorithm was proposed for classifying evolving data streams called Adaptive Random Forest (ARF) [2]. The algorithm includes an effective resampling method and adaptive operators that can handle different types of concept drifts with simple optimizations for different datasets. ARF can also be configured to use active concept drift identification through ADWIN [15]. ADWIN (ADaptive WINdowing) is an algorithm that detects changes in data concept statistically and raises a warning to the classification model. The authors also implemented a version of ARF with parallelism called ARF[M] and compared it with a version without parallelism called ARF[S] and observed that ARF[M] is 3 times faster and has no performance loss during classification. ARF has been compared with state-of-the-art algorithms and it was seen that it achieves good classification performance in both delayed and immediate settings.

A model for classifying data streams with concept drifts using *XGBoost* called AXGB (Adaptive eXtreme Gradient Boosting) [16] was also proposed. The method consists of creating new XGBoost models and adding them to an ensemble whenever a dynamic sliding window reaches its maximum size. The dynamic window size doubles in each iteration, growing exponentially. The maximum number of XGBoost classifiers in the ensemble is fixed, but the ensemble is updated with new XGBoost classifiers to ensure consistency with the current data concept. Since an XGBoost classifier is a tree based ensemble, AXGB can be considered an ensemble of ensembles, which has a longer response

time when compared to techniques that use fewer classifiers. The update of the AXGB ensemble can be done in two ways: I. Push, the ensemble looks like a queue, when new models are created they are appended to the ensemble, and older models are removed and II. Replacement, where older models are replaced by the new models. The authors performed several experiments and concluded that the Replacement strategy had better performance regarding performance, training time, and memory usage.

3. Proposal: Adaptive Fast XGBoost

This work proposes a fast classifier algorithm of data streams using the XGBoost library, as an adaptation of AXGB [16], called AFXGB (Adaptive Fast XGBoost).

By default XGBoost was not developed to deal with data streams, so it presents some limitations, the first one is that it does not handle concept drift. To make XGBoost adaptable to concept drifts we can use a feature of XGBoost which allows us to save the current model. Thereby, whenever the sliding window reaches its maximum size, the trained model previously saved is loaded and updated with the new training batch, and then saved again. This feature allows incremental training and constant update of the model.

The second limitation of XGBoost when handling data streams is that it does not forget outdated models, since they are not excluded during incremental training. Therefore, the ensemble of classifiers will have its complexity increased as long as new data arrives, and thus the training and prediction times will get higher, making the ensemble performance impractical. To overcome this problem, this work proposes a strategy of training two classifier models alternately, similarly to SUN algorithm [17].

Figure 1 shows the strategy of alternate model training. First, an incrementally trained model is used to classify the input data until it reaches the maximum lifetime defined by a parameter. In parallel with the training of this model, a new classifier starts to be trained before the first classifier exceeds its lifetime. However, during this period, only the oldest model is used to classify the input data and it will be replaced only when the new model reaches an adequate training level. This base training period before the model becomes effective is also a set parameter. So when the older model reaches its lifetime it is discarded and the new model becomes active. Figure 1 shows in dark blue the model currently in use to classify the input data and in light blue the training initialization of the new model that will replace the old one.

The parameters of maximum classification and base training of the classifiers are based on the number of filled data windows. Each time that the window reaches its max-

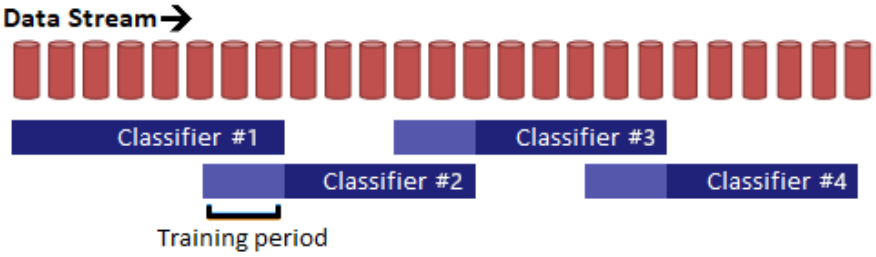

Figure 1. Alternating classifiers method

Algorithm 1: Adaptive Fast XGBoost alternate training

Input: (x, y) ϵ Data Stream
Data: *w* = maximum number of samples on the window, M = XGBoost classifier model, W = window, SW = sliding window of data, *life_time* = lifetime of each classifier, *training_time* = training time of each classifier, count = how many times the window has restarted

```
1  Adds (x, y) to window (W)
2  if |W| > w then
3      SW = updates sliding window with data from W
4      Adds (x, y) on buffer to train the classifier
5      if M <> NULL then
6          if training_time ≥ (life_time - count) then
7              # Checks if it is inside the training time of next classifier
8              nextM = trains next XGBoost classifier
9          if count ≥ life_time then
10             # resets counter and starts to use the new classifier
11             count = 0
12             M = nextM
13         M = loads the model;
14         M' = trains classifier with W using M and adds to M;
15         M = M';
16         Saves new M model recently trained;
17     else
18         M = Trains classifier with W;
19         Saves new M model recently trained;
20     W = W - W';
21     count = count + 1 # counts how many times the window has restarted
22 returns M;
```

imum capacity, the data is used to do the incremental train of the current classifier and removed from the window. Using small-sized windows makes the algorithm train more frequently, which slows down the processing time, but also updates the model's concept at a higher frequency. Thus, according to the proposed strategy, when the sliding window fills up with a number p of times, defined as a parameter, a new classifier starts to be trained, and when the window fills $q|(q > p)$ times, the new classifier replaces the older one and starts to be used.

3.1. Implementation

A pseudo-code for the proposed algorithm can be viewed in Algorithm 1. The algorithm starts with the instances arriving from the stream and added to the window (line 1). Then the algorithm checks whether the window has reached its maximum capacity (line 2). If so, then the sliding window is updated with the arriving data. After this, the data is added to the *buffer* to train the classifier (line 4).

Then algorithm checks if a classifier has already been built, if not (line 17), a new XGBoost classifier is trained and its model is saved for later updates (lines 18 and 19).

If there is already a trained classifier, there are two routines: I. checks if the replacement model should be trained (line 6) and II. checks if the current model should be replaced (line 9).

Routine I (line 6) works by evaluating if training_time is greater or equal than the life_time minus count, which is the current round. The training_time is a fixed parameter, so this will cause the replacement model to be trained a pre-set number of times before replacing the current model. This also guarantees that the training of the replacement model happens by the end of the current model's lifetime, meaning that the replacement model is trained with the latest data. Next, routine II (line 9) checks if the current round is greater or equal to the current model's lifetime. If so, the counter is reset and the model is replaced.

During lines 13 to 16, the model is loaded, trained with data from the window, and saved. Finally, the window is updated and the counter is incremented to record another training round (lines 20 and 21) and the trained model is returned (line 22).

3.1.1. Parameters

- **Learning Rate (eta)**: a value between 0 and 1. The learning rate applies a weighting factor to new trees added to the XGBoost model. When next to 0 the algorithm will make fewer corrections, thus resulting in more trees and slower processing time.
- **Maximum depth (*max_depth*)**: the maximum depth which the tree can reach. Increasing this number created a more complex model and increases memory consumption.
- **Maximum window size (*max_window_size*)**: maximum size of the window which stores the data from the data stream. The algorithm updated the models only when the maximum window size is reached.
- **Classifier lifetime (*life_time*)**: Lifetime of each alternate classifier. This value is based on the number of times that the sliding window was reset.
- **Training time (*training_time*)**: training time of each alternate classifier. This value is based on the number of times that the sliding window was reset.
- **Number of estimators (*n_estimators*)**: number of classifiers added to the model at each training step.

4. Results assessment

This section presents the results of tests performed with the AFXGB algorithm. The 4.1 section outlines the methodology that was used for the tests and the 4.2 section details the results obtained.

4.1. Testing methodology

To evaluate the proposed algorithm, the AFXGB was compared with another 3 data stream classification algorithms: Adaptive Random Forest (ARF) [2], Adaptive eXtreme Gradient Boosting (AXGB) [16] and Hoeffding Adaptive Tree (HAT) [14]. The addressed aspects were: speed, accuracy, and ability to adapt to new concepts. The evaluation

method selected is the *Prequential Evaluation* with batch size equal to 1, where each data is analyzed sequentially as they arrive in the model.

Experiments were made in 7 scenarios with real and synthetic data. We used two variations of Agrawal and SEA synthetic datasets to simulate abrupt (A) and gradual (G) concept drifts. The drifts were positioned every 125k instances and the window of change for gradual drifts is 20k. The Hyperplanes dataset is also synthetic and contains fast incremental drifts. Two real-world datasets were used, Airlines and Electricity Market. The summary of the datasets is shown in Table 1.

Table 1. Datasets characteristics. Concept drifts: Abrupt (A) and Gradual (G).

Dataset	# Instances	# Features
Agrawal-A	500000	9
Agrawal-G	500000	9
Hyperplanes	500000	10
SEA-A	500000	3
SEA-G	500000	3
Airlines	500000	31
Electricity Market	45312	6

The Prequential Evaluation simulation was executed five times for each algorithm and dataset to obtain the average accuracy and time. The hyperparameters for AFXGB were selected empirically based on numerous simulations, and for the other models, we used the parameters proposed by the authors in their respective works. The parameters were kept the same throughout all experiments and are the following:

- **ARF**: Drift Detection = ADWIN, Grace Period = 50, No. Estimators = 20
- **HAT**: Grace Period = 200, Split Confidence = 1×10^{-7}
- **AXGB**: Learning Rate = 0.3, Max Depth = 6, Max Window Size = 1000, Min Window Size = 1, No. Estimators = 30
- **AFXGB**: Learning Rate = 0.3, Max Depth = 6, Max Window Size = 1000, Min Window Size = 1, Lifetime = 25, Training time = 15, No. Estimators = 30

4.2. Analysis of the results

We present in Table 2 the average accuracy of the 5 executions of the considered models for each dataset. The best accuracy is highlighted, and the corresponding ranking of the model for that dataset is next to the accuracy. The model rank is defined by ordering all models' accuracy for that dataset from highest to lowest, this meaning that the smallest the ranking is better.

In the following sections, we analyze the accuracy, runtime, and concept drift recovery results.

4.2.1. Accuracy analysis

It is noticed that the average accuracy of all algorithms varies between 0.89 and 0.87 success rate. Because these results are very close, a statistical assessment of the results is

Table 2. Average Accuracy and rankings for the four algorithms considered in the study over 7 data sets.

Dataset	ARF	HAT	AXGB	AFXGB
Agrawal-A	0.9659 (4)	0.9808 (3)	0.9868 (2)	**0.9872** (1)
Agrawal-G	0.955 (4)	0.9649 (3)	0.9724 (2)	**0.9786** (1)
Hyperplanes	0.8485(4)	**0.8609** (1)	0.857 (2)	0.852 (3)
SEA-A	**0.9956** (1)	0.9879 (2)	0.986 (4)	0.9871 (3)
SEA-G	**0.9923** (1)	0.986 (2)	0.9857 (4)	0.9859 (3)
Airlines	**0.6722** (1)	0.64 (2)	0.6327 (3)	0.6312 (4)
Electricity Market	**0.863** (1)	0.8133 (2)	0.7273 (3)	0.7257 (4)
Avg. acc.	0.8989	0.8905	0.8782	0.8782
Avg. rank	2.2857	2.1429	2.8571	2.7143

needed [18]. To find if there is a performance difference between the algorithms, we start applying the Friedmann test. Performing Friedman with a significance of 95% resulted in a $p\text{-}value$ of 5.42×10^{-133}, which rejects the null hypothesis. This means we found a significant difference and need to apply a pair-wise $post\text{-}hoc$ test for multiple comparisons [19].

The $post\text{-}hoc$ test applied was a Nemenyi test using the algorithms' average ranking. This test determines a Critical Difference (CD), meaning that two algorithms whose ranking difference is greater than the critical difference are considered significantly different. The Nemenyi resulted in a CD of 0.79 and pair-wise comparisons are shown in Figure 2. Since the difference of average ranks between all models is smaller than the value of CD, we can conclude that their performance difference is not significant.

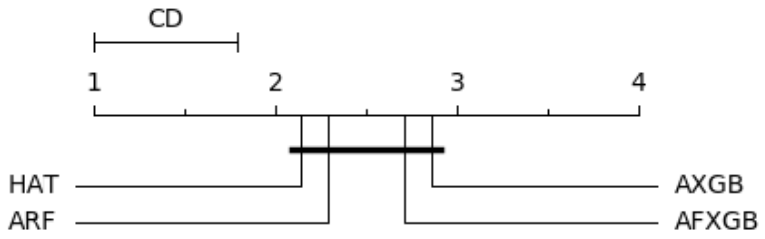

Figure 2. Nemenyi test of model's average rankings.

4.2.2. Runtime analysis

Since it was proven that the accuracy has no significant statistical difference, we focus on comparing the simulated models runtimes. Table 3 presents the average training time, average testing time, and average total time on minutes of all simulations on all datasets. These times correspond to the execution of the entire simulation, not just a single instance.

It is apparent that ARF and HAT have larger training times and smaller testing times, while AXGB and AFXGB are the opposite. This happens due to XGBoost characteristic, where the training of classifiers can happen in parallel, but testing needs to

Table 3. Average times of complete simulations for the algorithms considered in the study.

	Time (minutes)		
Model	**Avg. training**	**Avg. testing**	**Avg. Total**
ARF	71.05	6.55	77.60
HAT	2.90	0.47	3.37
AXGB	0.21	378.54	378.76
AFXGB	0.31	16.26	16.57

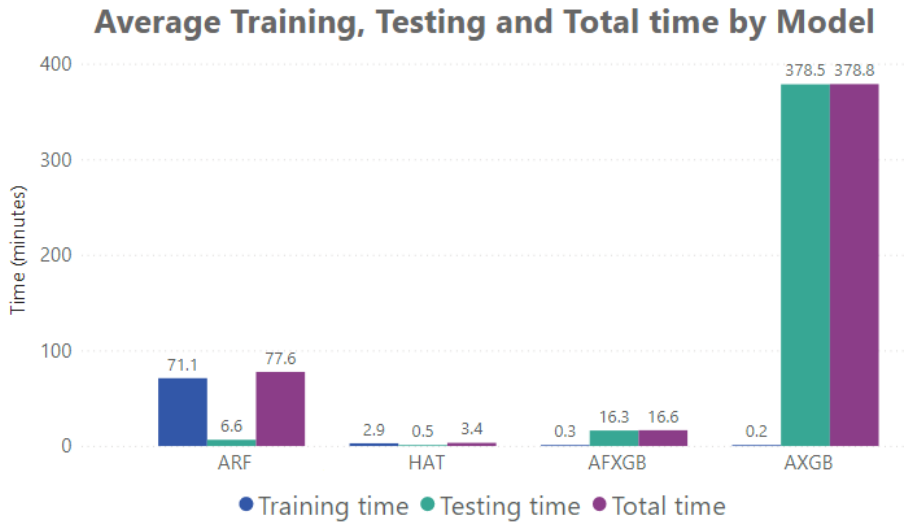

Figure 3. Average training, testing, and total running time of all models.

calculate and consolidate all classifiers' results, resulting in a larger testing time. Also, AXGB and AFXGB have more trees in the ensemble than the others due to the No. estimators used. Each training phase AXGB and AFXGB train 30 models, while ARF trains 20. This means that our ensemble grows faster, thus spending more time to run and consolidate results during testing. Even though, our model is able to train much faster and has a better overall time than ARF.

The average running times are compared in Figure 3. It is noticeable that AXGB has the largest running time, exceeding AFXGB's execution by 22 times. This is expected because of AXGB multiple ensemble characteristic, which was adapted to alternate model training strategy on AFXGB, thus reducing overall model complexity and running time. AFXGB's runtime could be reduced even further by using a smaller lifetime, thus decreasing the model's build-up by resetting it more frequently, but this needs to be balanced with possible accuracy reduction.

It is important to notice that for non-stationary data streams the training times are extremely relevant, since training the model must be done constantly to keep it up to date in the occurrence of concept drifts. In summary, it is not enough for data stream models to be fast only during training or testing, they must be fast in the sum of both times.

This means that ARF time performance is worse than AFXGB, possibly because of the concept drift detection algorithm, and the fastest one is HAT due to the algorithm's

simplicity.

4.2.3. Concept drift recovery analysis

To evaluate the ability of the models on recovering from a concept drift we analyzed the accuracy of each model over time after each drift. We selected simulations of Agrawal-A and Agrawal-G to show the model's behavior on both abrupt and gradual concept drifts. The behavior was similar to other simulations with concept drifts.

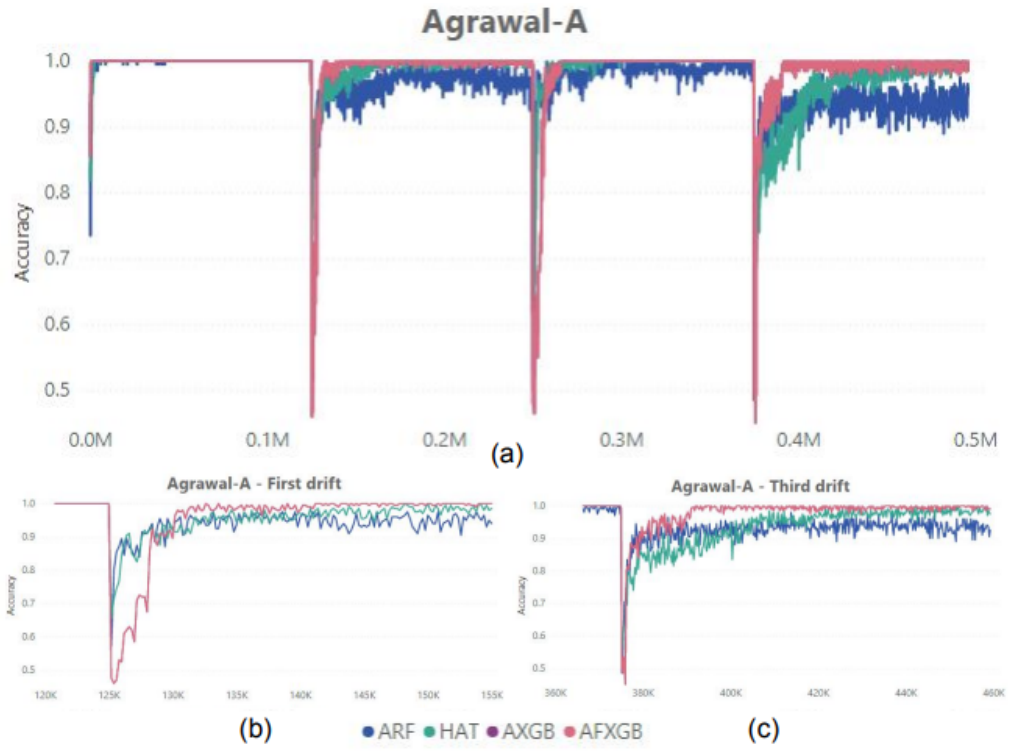

Figure 4. (a) Accuracy over time for ARF, HAT, AXGB, and AFXGB models on Agrawal-A dataset. (b-c) A highlight of 1st and 3rd concept drifts of the simulation.

In Figure 4a we observe the models accuracy for the simulation of Agrawal-A dataset, where AFXGB increases its accuracy faster than ARF and HAT. This can be perceived in more detail in Figure 4b and 4c, where the first and third drifts are highlighted. It is seen that HAT loses accuracy over time, having a worse recovery each drift. ARF is fast in detecting the concept drift, this being expected because of its use of ADWIN active drift detection, but it is not so quick to adapt to the new concept. AXGB and AFXGB have similar behavior.

On the first Agrawal-A drift depicted by Figure 4b, we can see that AFXGB recovery took a longer time to happen. It is possible that there was a coincidence of the drift happening right after AFXGB's maximum lifetime reach and model change. This means that the new model was trained with old concept data and because of this, it took a long time to adapt to the new concept.

This can be further observed in Table 4, where the accuracy of all models is dis-

played for five data points related to the first concept drift of Agrawal-A. On data 125k the drift has not happened yet, so all models are at maximum accuracy, then on the following 125.2k and 125.4k data points, AXGB and AFXGB have the largest accuracy drops, but can recover on 128.4k and surpass ARF and HAT by point 130.2k. In this scenario, it took about 5.2k of data volume for achieving full concept drift recovery.

Table 4. Accuracies of models on data points before and after the first concept drift for the 4th simulation of Agrawal-A dataset.

# data	ARF	HAT	AXGB	AFXGB
125k	1.00	1.00	1.00	1.00
125.2k	0.57	0.66	0.48	0.48
125.4k	0.8	0.72	0.46	0.46
128.4k	0.94	0.93	0.93	0.93
130.2k	0.95	0.92	0.97	0.97

In comparison, Table 5 shows the accuracy of all models for five data points related to the third concept drift of Agrawal-A. Once again, all models are at the highest accuracy before the drift on point 375k and suffer the largest drops by following points 375.2k and 376k. Then by 376.2k all models recover and by 389k AXGB and AFXGB surpass other models, with the data volume for recovery being 14k.

Table 5. Accuracies of models on data points before and after the third concept drift for the 4th simulation of Agrawal-A dataset.

# data	ARF	HAT	AXGB	AFXGB
375k	0.99	1.00	1.00	1.00
375.2k	0.49	0.5	0.5	0.5
376k	0.71	0.68	0.45	0.44
376.2k	0.77	0.73	0.79	0.77
389k	0.87	0.87	0.95	0.95

Looking upon gradual concept drifts, Figure 5a shows the models accuracy for Agrawal-G dataset simulation. Since the concept drift happens progressively, ARF was not able to perform the fast concept drift detection as in the abrupt concept drift. As seen on 5b and 5c, all models adapted gradually to the concept drift, but AFXGB again shows a faster and better recovery than HAT and ARF. Once more, AXGB and AFXGB have equivalent behavior.

5. Conclusion and future work

In this work, we proposed a Fast Adaptive XGboost binary classification model (AFXGB) to handle data streams. AFXGB uses an alternate model training strategy to reduce the model piling up complexity and adapt faster to concept drifts. Comparing its accuracy, speed, and ability to adapt to concept drift with other models in different scenarios, including real and synthetic datasets with different kinds of concept drifts, it was seen that AFXGB can achieve the same level of accuracy as other models as proven statistically. Meanwhile, it is also faster than the more robust models, being 4 times faster than ARF and 22 times faster than AXGB.

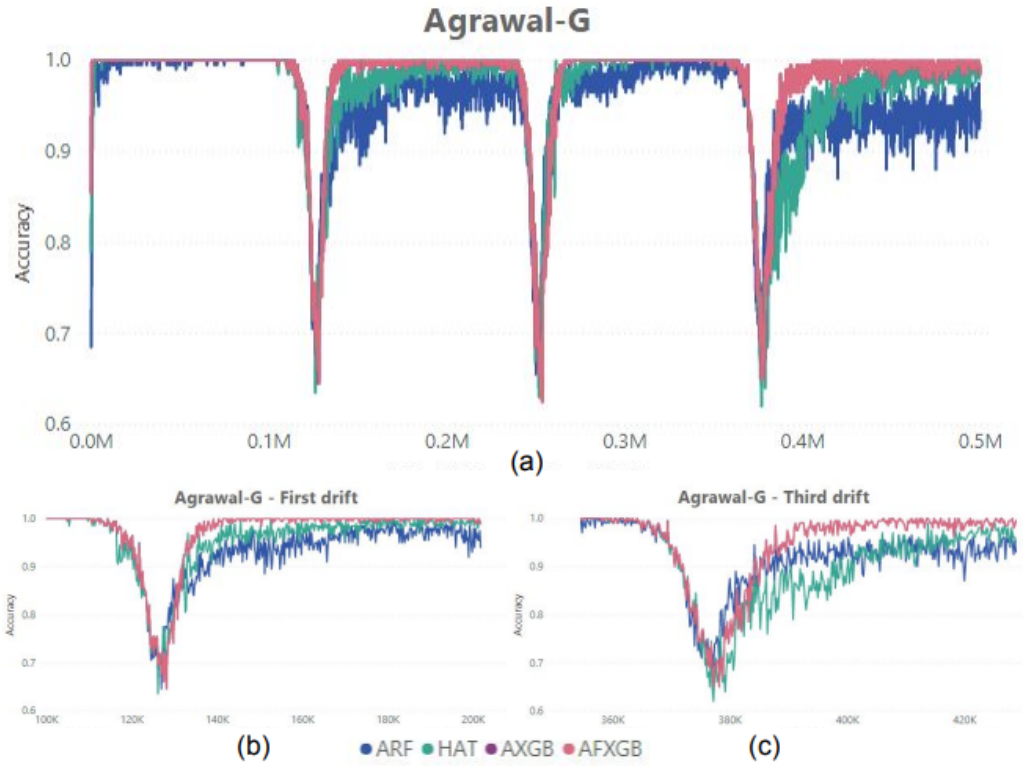

Figure 5. (a) Accuracy over time for ARF, HAT, AXGB, and AFXGB models on Agrawal-G dataset. (b-c) A highlight of 1st and 3rd concept drifts of the simulation.

Regarding concept drifts adaptation, the AFXGB model presented a faster and more consistent adaptation than ARF and HAT. It was seen that it needs less volume of data to come back from the accuracy drop and it is able to keep high long-term accuracy since old concepts are forgotten when the model is reset and substituted. Based on what was evaluated, the contribution of this work is related to AFXGB's earlier recovery from concept drifts, without using any concept drift detection algorithm and keeping a low runtime overall. For future works, we intend to explore the application of concept drift detectors for AFXGB to enhance the concept drift adaptation feature. Also, we want to explore a sliding window size reset with every substitution of the model, so the adaptation from concept drifts can happen with less volume of data and consequently. With these new features, we hope that the overall method accuracy improves.

Acknowledgment

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

[1] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big IoT data analytics: Architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[2] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream

classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, Oct 2017. [Online]. Available: https://doi.org/10.1007/s10994-017-5642-8

[3] K. K. Wankhade, S. S. Dongre, and K. C. Jondhale, "Data stream classification: a review," *Iran Journal of Computer Science*, vol. 3, no. 4, pp. 239–260, Dec 2020. [Online]. Available: https://doi.org/10.1007/s42044-020-00061-3

[4] H. M. Gomes, J. P. Barddal, L. Boiko Ferreira, and A. Bifet, "Adaptive random forests for data stream regression," 04 2018.

[5] R. D. Baruah, P. Angelov, and D. Baruah, "Dynamically evolving fuzzy classifier for real-time classification of data streams," in *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2014, pp. 383–389.

[6] H. Guo, H. Li, Q. Ren, and W. Wang, "Concept drift type identification based on multi-sliding windows," *Information Sciences*, vol. 585, pp. 1–23, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025521011439

[7] H. Binder, O. Gefeller, M. Schmid, and A. Mayr, "The evolution of boosting algorithms," *Methods of Information in Medicine*, vol. 53, no. 06, pp. 419–427, 2014. [Online]. Available: https://doi.org/10.3414\%2Fme13-01-0122

[8] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Comput. Surv.*, vol. 50, no. 2, mar 2017. [Online]. Available: https://doi.org/10.1145/3054925

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[10] R. Santhanam, S. Raman, N. Uzir, and S. Banerjeeb, "Experimenting XGBoost algorithm for prediction and classification of different datasets," *International Journal of Control Theory and Applications*, vol. 9, no. 40, 2016.

[11] P. B. Dongre and L. G. Malik, "A review on real time data stream classification and adapting to various concept drift scenarios," in *2014 IEEE International Advance Computing Conference (IACC)*, 2014, pp. 533–537.

[12] M. Datar and R. Motwani, *The Sliding-Window Computation Model and Results*. Boston, MA: Springer US, 2007, pp. 149–167. [Online]. Available: https://doi.org/10.1007/978-0-387-47534-9_8

[13] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 97–106. [Online]. Available: https://doi.org/10.1145/502512.502529

[14] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII*, N. M. Adams, C. Robardet, A. Siebes, and J.-F. Boulicaut, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 249–260.

[15] A. Bifet and R. Gavaldà, *Learning from Time-Changing Data with Adaptive Windowing*, pp. 443–448. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.42

[16] J. Montiel, R. Mitchell, E. Frank, B. Pfahringer, T. Abdessalem, and A. Bifet, "Adaptive XGBoost for evolving data streams," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[17] X. Wu, P. Li, and X. Hu, "Learning from concept drifting data streams with unlabeled data," *Neurocomputing*, vol. 92, pp. 145–155, 2012, data Mining Applications and Case Study. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231212001270

[18] B. Calvo and G. Santafé Rodrigo, "scmamp: Statistical comparison of multiple algorithms in multiple problems," *The R Journal, Vol. 8/1, Aug. 2016*, 2016.

[19] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.