Adaptive Extreme Gradient Boosting Regressor for Evolving Data Streams

Hélder Vieira, up201503395
Samuel Aduroja, up202009191
Vânia Guimarães, up200505287

Data Stream Mining 2021/2022

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Outline

1) Motivation
2) Purpose of project
3) Theoretical description of Adaptive Extreme Gradient Boosting
4) AXGB implementation
5) Results
6) Conclusion

XGBoost - Extreme Gradient Boosting.

- XGBoost is a specific **implementation of the Gradient Boosting** method
- Power of XGBoost lies on:
  - Accuracy with introduction of a regularization term
  - Speed
  - Efficient memory usage
  - Scalability
- These features are important to deal with stream data, because resources like running time and space memory are limited.

AXGB – Adaptive Extreme Gradient Boosting

Purpose of project: Implementation of Adaptive Extreme Gradient Boosting proposed in 2020 by Jacob Montiel et all. [1]

Modifications:

- adjustments to solve regression problems rather than classification on evolving data streams;
- rewrite from scratch several functions;
- replace ADWIN detector with Page-Hinkley test;

[1] J. Montiel, R. Mitchell, E. Frank, B. Pfahringer, T. Abdessalem, and A. Bifet, 'Adaptive XGBoost for Evolving Data Streams', arXiv:2005.07353 [cs, stat], May 2020, Accessed: Dec. 05, 2021. [Online]. Available: http://arxiv.org/abs/2005.07353

AXGB – Adaptive Extreme Gradient Boosting

- K number of weak learners are needed to build full ensemble model.
- The proposed method creates new members of the **ensemble** from "**mini-batches**" of data as new data becomes available;
- The size of the windows/buffer, is predefined (W).
- First weak learner is trained over this mini-batch of data.
- When a new chunk of data arrives, the model make a prediction.
- The errors are computed and the next weak learner is trained on these residuals.

AXGB – Objective Function

- Each new weak learner improves the model most by minimizing the objective function:

$$L(\emptyset) = \sum_i \; l(\hat{y}_i, y_i) + \sum_k^K \Omega(f_k)$$

$l(\hat{y}_i, y_i)$ is the differentiable loss function

$\Omega(f) = \gamma T + \frac{1}{2}\lambda||\theta||^2$ is the regularization term which controls the complexity of trees, preventing the chance of overfitting.

AXGB – Dynamic Window Size

- When ensemble model is empty, we can create new weak learners based on small windows. At each iteration W_{min} increases exponentially until it reaches a pre-defined maximum size.

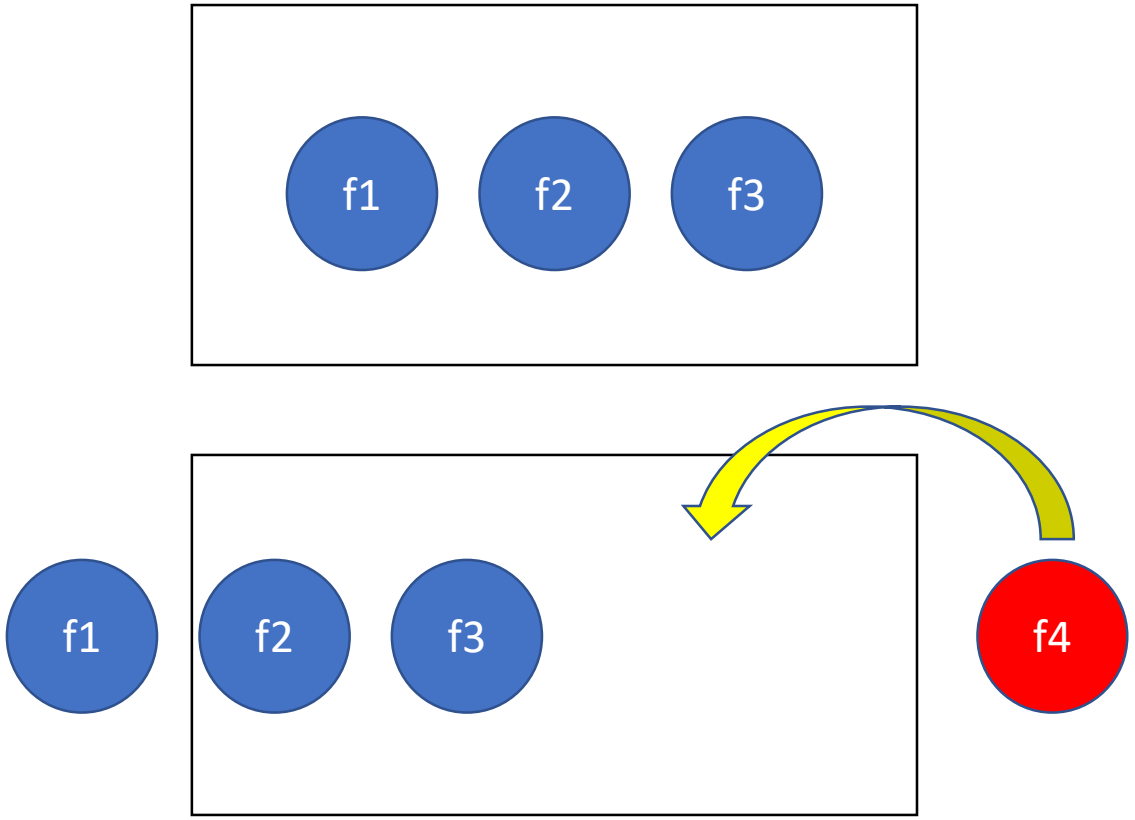
- Dynamic window size is a strategy to accelerate the process of construct all the entire model.

AXGB – Update Strategy

- When the ensemble reaches the pre-defined maximum (K) size there are two possible strategies: **Push** or **Replace** to continue training and evolving the model;

- Note that in both strategies we have to wait K iterations to get a completely new ensemble;

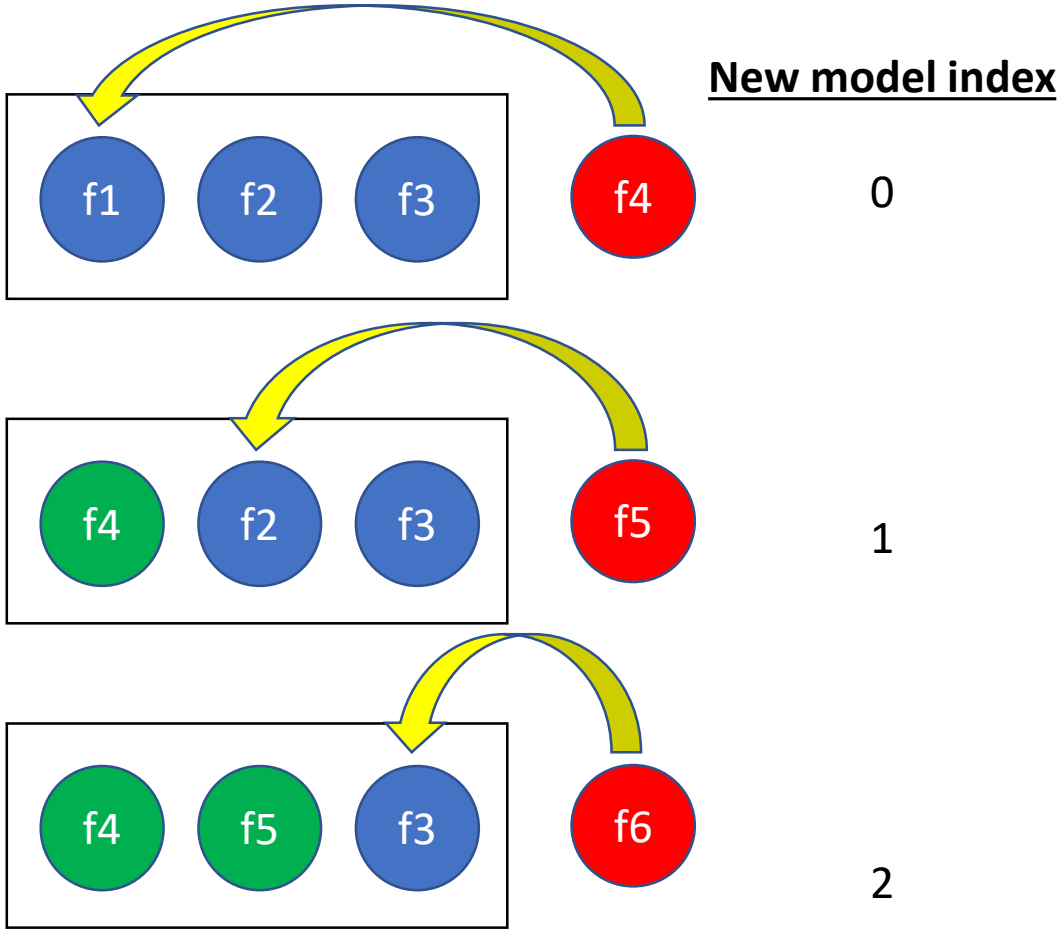
AXGB - Update: Push



The ensemble is full. To keep it updated **Push** can be applied.

When the buffer is full a new booster (f4) is trained using the residuals from the models in the ensemble (f1, f2, f3). The oldest model f1 is "pushed" out.

AXGB - Update: Replace



The ensemble is full. When the buffer reaches its capacity an index is initialized at 0. However, f4 is not trained using the residuals from the previous models.

In the next update, the new booster is trained using the residuals from the models that are on the left from the current index. f5 is trained with f4 residuals.

Finally, f6 is trained using residuals from f4 and f5.

AXGB – Drift Detection

Ensemble methods naturally adapt to concept drift. However, the process can be slow in case of fast drifts.

Active approach helps the model learn new concept quickly.

Model may be reacting to a false alarm.

Increase computational resources and time.

AXGB – Drift Detection

- **Trade-off** between computational costs and gains in performance.
- User can choose to use or not **Page – Hinkley test** as drift detector.

- When **Page – Hinkley test** detects change of concept:
  - Window size is reset. The size of new buffer will have W_{min} samples.
  - In Replace strategy, the new model index is also reset; all weak learners are trained over data from new concept.

Page-Hinkley test

- At each time stamp, we compare the error by MAE.
- PH compute the cumulative sum of differences of the observed error and their mean until the present moment:

$$m_t = \sum_{t=1}^{T} (x_t - \bar{x}_T + \alpha)$$

- Detection is declared when $m_t - m_{min} > \lambda$
- α represents the magnitude of accepted change
- λ expresses tolerable false alarm rate.

AXGB – Implementation

- Implementation of AXGB was made using scikit-multiflow package, a data stream software written in Python and XGBoost python package.

- The open-source code is available on github:

https://github.com/hmcvieira/Data-Stream-Mining/tree/main/HW1/AXGBregression

- It contains the following files:
  - *axgb_regression.py:* script that implements the regression algorithm
  - *axgb_regression_test.py:* contains an example execution script
  - *README:* briefly describes the file structure

AXGB – Execution demo

- Now, we will run a live example to show our script execution.

- The execution script has the following parts:

1. Initialize the desired models;
2. The dataset/stream must be loaded, we created an artificial one;
3. Setup and run the prequential evaluator;

AXGB – Execution Demo

- Algorithm was tested in an artificial dataset, created by *RegressionGenerator* method, available on scikit-multiflow.
  1) 2,000 samples, 15 features, 8 informative features
  2) 3,000 samples, 15 features, 10 informative features
  3) 5,000 samples, 15 features, 12 informative features

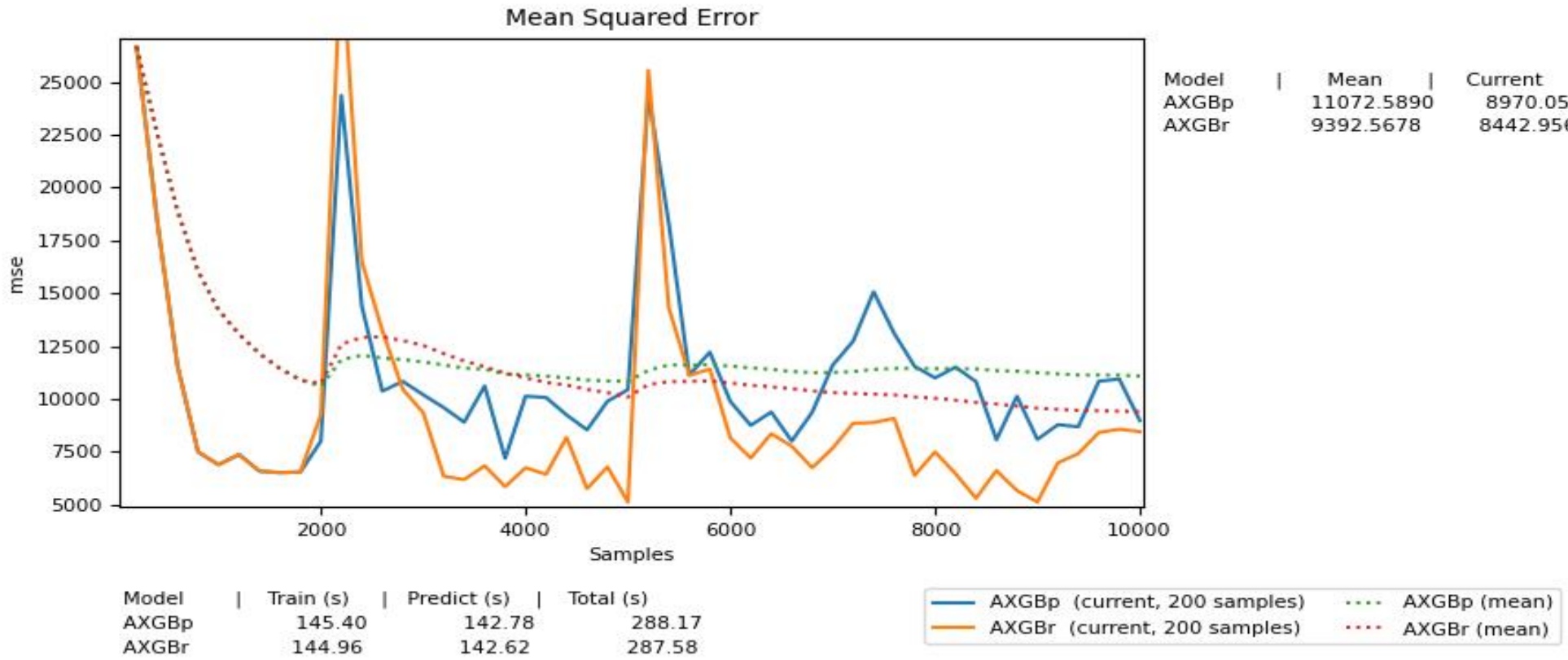
- We have 2 **concept drifts**: at position 2,000 and 5,000.

AXGB – Execution Demo

Hyper-parameters of our model:

- n_estimators = 20
- learning_rate = 0.3
- max_depth = 6
- max_window_size = 100
- min_window_size = 10
- detect_drift = True/False
- threshold = 5100

AXGB – Results with Page-Hinkley test



Model	Mean	Current
AXGBp	11072.5890	8970.0527
AXGBr	9392.5678	8442.9565

Model	Train (s)	Predict (s)	Total (s)
AXGBp	145.40	142.78	288.17
AXGBr	144.96	142.62	287.58

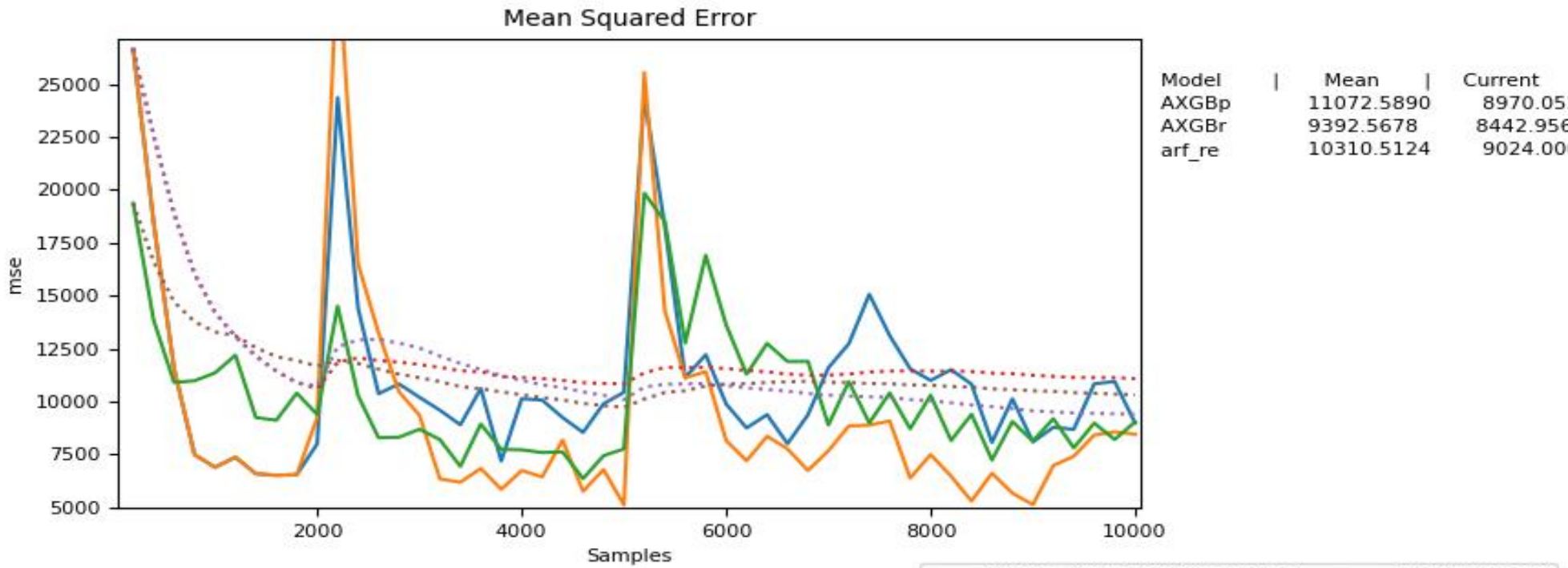
AXGB – Results without Drift Detector

Model	Train (s)	Predict (s)	Total (s)
AXGBp	2.18	137.14	139.32
AXGBr	1.48	136.93	138.42

AXGB – Conclusions

- AXGB[p] is slightly slower because it uses more ensemble members to make predictions and to train new learners;
- Drift detection has a negative impact on the running time;
- Total time spent by methods with drift detector more than doubled.
- Tuning hyper-parameters of Page-Hinkley test is not easy.
- False alarms may reduce the accuracy.
- Hyper-parameters setting like number of estimators, window size, learning rate and so on, have a significant impact on the performance.

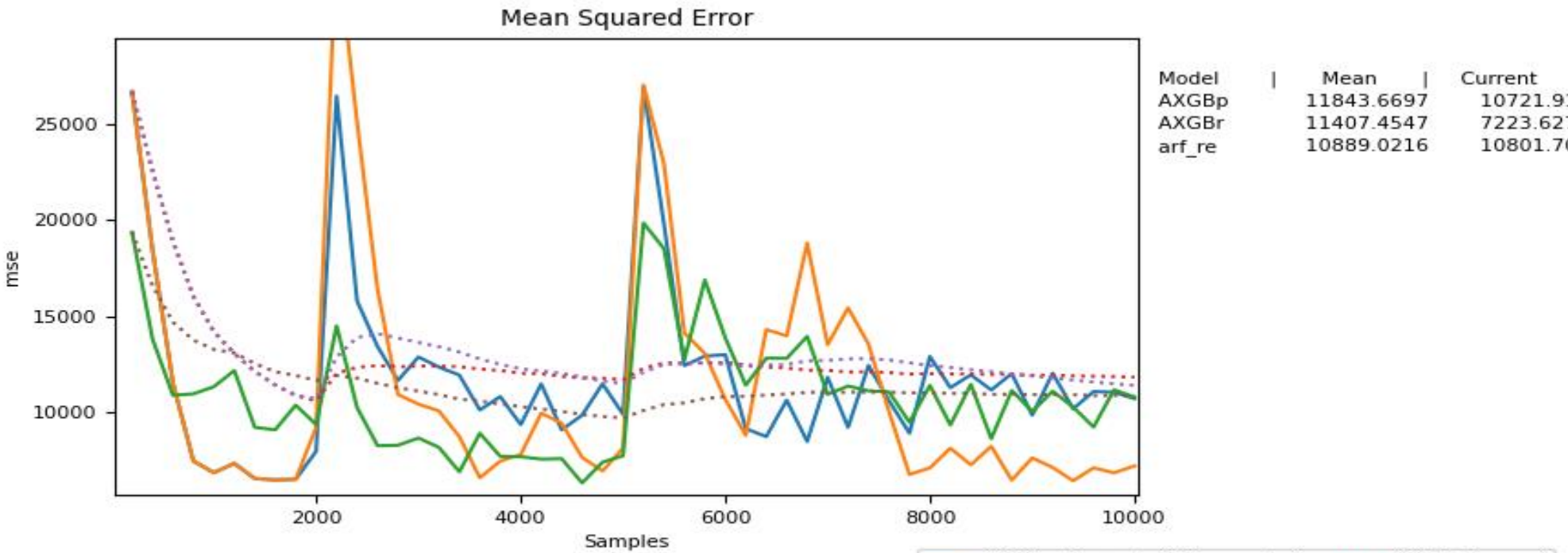
AXGB vs ARF – Results with Page-Hinkley test



Model	Mean	Current
AXGBp	11072.5890	8970.0527
AXGBr	9392.5678	8442.9565
arf_re	10310.5124	9024.0005

Model	Train (s)	Predict (s)	Total (s)
AXGBp	207.44	205.34	412.79
AXGBr	205.09	202.40	407.48
arf_re	366.76	39.76	406.52

AXGB vs ARF – Results without Drift Detector



Model	Mean	Current
AXGBp	11843.6697	10721.914
AXGBr	11407.4547	7223.6276
arf_re	10889.0216	10801.706

Model	Train (s)	Predict (s)	Total (s)
AXGBp	3.62	241.48	245.09
AXGBr	2.44	239.53	241.97
arf_re	374.86	45.62	420.48

AXGB vs ARF - Conclusion

Comparing AXGB against ARF, without drift detector:

- Mean MSE and MSE at 10,000 position is similar for all three models
- ARF took almost twice as long to train and to predict

Comparing AXGB against ARF, with drift detector:

- Mean MSE and MSE at 10,000 position is similar for all three models
- ARF was faster but the difference is negligible

The results support the quality and feasibility of our algorithm implementation.