On the Impact of Industrial Delays when Mitigating Distribution Drifts: an Empirical Study on Real-world Financial Systems

Thibault Simonetto[1], Maxime Cordy[1], Salah Ghamizi[2], Yves Le Traon[1], Clément Lefebvre[3], Andrey Boystov[3], and Anne Goujon[3]

[1] University of Luxemourg {thibault.simonetto, maxime.cordy, yves.letraon}@uni.lu
[2] Luxembourg Institute of Science and Technology salah.ghamizi@list.lu
[3] BGL BNP Paribas {andrey.boytsov, anne.goujon}@bgl.lu

Abstract. An increasing number of financial software system relies on Machine learning models to support human decision-makers. Although these models have shown satisfactory performance to support human decision-makers in classifying financial transactions, the maintenance of such ML systems remains a challenge. After deployment in production, the performance of the models tends to degrade over time due to concept drift. Methods have been proposed to detect concept drift and retrain new models upon detection to mitigate the drop in performance. However, little is known about the effectiveness of such methods in an industrial context. In particular, their evaluation fails to consider the delay between the detection of the drift and the deployment of a new model. This delay is inherent to the strict quality assurance and manual validation processes that financial (and other critical) institutions impose on their software systems. To circumvent this limitation, we formalize the problem of retraining ML models against distribution drift in the presence of delay and propose a novel protocol to evaluate drift detectors. We report on an empirical study conducted on the transaction system of our industrial partner, BGL BNP Paribas, and two publicly available datasets: Lending Club Loan Data and Electricity. We release our tool and benchmark on GitHub [4]. We demonstrate for the first time how ignoring the delays in the evaluation of the drift detectors overestimates their ability to mitigate performance drift, up to 39.86% for our industrial application.

Keywords: ML, distribution-drift, real-world system, AI in finance

1 Introduction

Machine Learning (ML) is used in industry to leverage an increasingly large amount of data and reduce operational costs, develop new disruptive products, and deliver personalized services to their customers.

[4] Code available at https://github.com/serval-uni-lu/drift-robustness

Our industrial partner is an important actor in the financial sector, and has many operational benefits from automated services built using ML, including scalability at low costs and the ability to process large amounts of data efficiently.

However, the wide dissemination of ML technologies within industrial software systems is hindered by their high maintenance costs [1]. Our partner has observed that the effectiveness (e.g. prediction accuracy) of their ML systems declines over time due to changes in data distribution.

The usual solution to mitigate the effect of drifts on ML systems is to retrain the ML model periodically (periodic retraining) or continuously (online learning). In our partner's case, online learning is prohibited by stringent security policies, which prevent feeding models with live data without manual checks. Thus, our partner relies on periodic retraining. The critical questions they face are *how often* and *how* they should retrain their model.

Alternatively, research has developed *drift detectors* as a better means to decide when to trigger model retraining [14]. A drift detector is a statistics-based method that takes as input a stream of samples. After each sample, it returns whether the observed distribution has shifted or not compared to previous samples. Using drift detectors to trigger retraining at the most appropriate times can reduce the cost of periodic retraining and increase its effectiveness.

Although the use of periodic retraining and drift detectors has been intensively investigated in the literature [14,18,3], previous studies do not consider the industrial constraints facing ML systems in production. In particular, two types of delay inhibit the retraining process. First, *labeling delay* implies a temporal distance between the time at which a new data reaches the system and the time at which its ground truth label is retrieved. Second, *deployment delay* is inherent to the strict quality assurance and manual validation processes that financial institutions (and other critical institutions) impose on their software systems. Hence, there is a significant time gap between when a system is fully engineered (or updated) and when it is running in production. In the case of BGL BNP Paribas, this delay is typically 10 days for labeling, and 28 days for deployment.

In this paper, our objective is to uncover the capabilities of retraining strategies to mitigate the effect of drifts *in presence* of labeling and deployment delays. Therefore, we conduct an empirical study involving one real-world financial system of our partner, and one publicly available dataset. We cover 16 retraining scheduling methods (based on periodic retraining or drift detectors). We measure the capability of these strategies to retrain models efficiently (minimizing the number of retraining) and effectively (maximizing performance over time). We specifically investigate the impact that the aforementioned delay has on existing retraining practices. To summarize, our study brings three novel contributions:

- We formulate the problem of retraining against distribution drifts in presence of *labeling and deployment delays*.
- We propose a novel step-by-step protocol for ML practitioners to diagnose concept drifts with deployment delays and identify the best retraining strategies that fit their case.

– We report on an empirical study of the effectiveness and efficiency of retraining strategies. We notably shed light on the impact of window size of retraining, the importance of drift detectors tuning, and how the delay affects the Pareto-optimal retraining strategies.

Through our study, we highlight the importance that labeling and deployment delays have on the predictive maintenance of machine learning-based systems, and the scale at which these delays impact the solutions to combat distribution drifts in the real world. By providing a proper definition and evaluation protocol of this industry-relevant problem overlooked by the literature, we hope to inspire future research on designing effective and efficient solutions.

2 Background

2.1 Source of performance drift

In their survey [14], Lu et al. define concept drift (also defined as distribution shift [22]) as follows:

Given a time period $[0, t]$, the set of samples, denoted $S_{0,t} = \{d_0, \ldots, d_t\}$, where $d_i = (x_i, y_i)$ is one observation (or a data instance), x_i is the feature vector, y_i is the label and $S_{0,t}$ follows a certain distribution $F_{0,t}(X, y)$. Concept drift occurs at timestamp $t + 1$, if $F_{0,t}(X, y) \neq F_{t+1,\infty}(X, y)$, denoted $\exists t : P_t(X, y) \neq P_{t+1}(X, y)$.

According to this definition, concept drift can be defined as the change in the joint probability of X and y at time t. The joint probability $P_t(X, y)$ can be decomposed as $P_t(X, y) = P_t(X) \times P_t(y|X)$, therefore, the concept drift can have three sources. We describe the three sources of concept drift and explain how each source can influence the performance of the model.

– $P_t(X) \neq P_{t+1}(X)$ while $P_t(y|X) = P_{t+1}(y|X)$, that is, only the input space distribution $P_t(X)$ changes and the relation $P_t(y|X)$ remains unchanged. This kind of drift does not affect the true decision boundary. Therefore, it may or may not affect the performance of the model depending on the difference in the distributions and the generalization capability of the model. It remains interesting to study drifts in the $P(X)$ distribution as it is observable at prediction time and is model-agnostic. As an example, the extension of the ML system usage to a new type of client can cause this kind of drift.
– $P_t(y|X) \neq P_{t+1}(y|X)$ while $P_t(X) = P_{t+1}(X)$, that is the distribution of input $P_t(X)$ remains unchanged, but the true decision boundary updates. Therefore, the decision boundary learned by the model is outdated and this causes a drop in accuracy in the region where the boundary has changed. For example, the changing economic context in which a financial system evolves can cause this type of drift.
– $P_t(X) \neq P_{t+1}(X)$ and $P_t(y|X) \neq P_{t+1}(y|X)$, that is, both the input data distribution and the decision boundary changes. In many real-world applications, this type of drift occurs, and we observe a drop in model accuracy.

2.2 Drift detectors

A drift detector is a method that observes a stream of data over time and determines for every new data point if the current distribution of the data has changed compared to a reference data set. We survey the literature for available drift detectors and identify three types of detectors: data-based detectors [23,11,19], error-based detectors [4,8,2,6,20,16], and predictive detectors [21,10]. We select drift detectors that are scalable to handle our datasets, which contain more than a million examples. Our second criterion is the availability of the implementation - or the implementation details - that allows us to reproduce the detectors presented in their respective paper. We identified three data-based, seven error-based, and two predictive-based detectors. The intuition behind each detector and their tunable parameters are given in Appendix A.

2.3 Domain Generalization

The domain generalization (DG) problem was first formally introduced by Blanchard et al. [5]. Unlike other related learning problems such as domain adaptation or transfer learning, DG considers the scenarios where target data is *inaccessible* during model learning. Hence, adaptation to distribution shift falls under the umbrella of domain generalization. [24] introduced a categorization of techniques commonly used to address the DG challenge, including domain alignment training, synthetic data augmentation, and self-supervised learning. Our work stems from the need of our industrial partner to improve the monitoring of the deployed models and to effectively trigger their well established retraining procedures. While we believe that DG techniques could also improve the performance of the models against distribution shift, all these approaches are orthogonal to our investigations for an efficient retraining schedule under delay.

2.4 Delays in time series evaluation

Masud et al. [15] study the problem of novel class detection while considering the true label delay constraints. They show how delaying the classification of incoming samples helps to detect new classes before the true label is available, hence providing a more accurate prediction. Poenaru-Olaru et al. [18] investigated recently the reliability of data and error-based drift detectors. However, the experimental protocol used does not consider label and deployment in production delays, which is the core of our study. In [17], Plasse et al. introduced a taxonomy to describe the labeling delay mechanism. They showed how delayed labels can be used to pre-update classifiers in real-world applications. However, the study didn't tackle validation delays and their impact on the model's deployment or the drift monitoring process. Žliobaité [25] analyzed the factors that allow concept drift detection before the label is available. However, no experiments are conducted on the impact of the finding on the performance of ML predictions, which is the aim of our novel protocol.

3 Problem

Without loss of generality, we consider a classification problem defined on a n dimensional feature space $\mathcal{X} \subseteq \mathbb{R}^n$ and a binary label space $\mathcal{Y} = \{0, 1\}$. We assume that the samples come as a time series S where each sample $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is indexed by a discrete time parameter t_i, such that t_i represents the time at which the input x_i reached the system. The *labeling delay* of x_i is the time between t_i and the moment x_i receives its true label y_i. For simplicity, we assume that this delay δ_l is constant across the inputs.

Let $h_{t_j} : \mathcal{X} \to \mathcal{Y}$ be the classification model trained at time t_j. Then h_{t_j} can only be trained on inputs $\{x_i\}$ such that $t_i + \delta_l \leq t_j$. Furthermore, the *deployment delay* of h_{t_j} is the time needed to deploy it in production. As before, we assume that this delay δ_d is constant. Thus, any model h_{t_j} can only make predictions on inputs that arrive in the system after it is deployed, that is, on inputs $\{x_k\}$ such that $t_j + \delta_d \leq t_k$.

We define a *retraining schedule* as an ordered sequence $sched = \{t_1 \dots t_n\}$ that determines when a model should be re-trained. For example, periodic retraining uses a sequence in which all elements are exactly separated by a constant period p, that is, $t_{j+1} = t_j + p$. By contrast, drift detectors decide the schedule on the fly based on their statistical analysis of the data and the model. A retraining schedule determines a sequence of models $H = \{h_{t_1} \dots h_{t_j} \dots h_{t_n}\}$. Then, any input example x_i observed at time t_i is predicted by the latest available model: The predicted label for x_i is given by $\hat{y}_i = h_{t_*}(x_i)$ where $t_* = \max\{t_k \in sched \text{ s.t. } t_k + \delta_{prod} \leq t_i\}$.

The evaluation of retraining schedule is two-folds: *effectiveness* and *efficiency*. Effectiveness is measured as the negative of the prediction error made by the sequence of models H calculated by an arbitrary scoring function $score(\mathcal{Y}, \hat{Y})$, with $\hat{Y} = \{\hat{y}_i\}$. Efficiency measures the overall cost of model retraining. For simplicity, in our study we assume this cost to be constant across models and compute it as the number $n = |H|$ of retraining. This assumption matches the context of our partner, where the cost of deploying models in production largely surpasses the computational cost to retrain the model and the data labeling cost.

4 Methodology

We propose a novel protocol to thoroughly evaluate retraining scheduling techniques under realistic industrial constraints (labeling and deployment delays). These constraints have been overlooked by previous studies. We do so, moreover, while carefully and empirically considering alternative design decisions that affect model performance (incl. hyperparameter tuning and training window size).

Our protocol starts from an initial model m_0 trained on an initial training set $S_{train} =]d_0, d_{N_{train}}]$, which contains the first N_{train} example of the time series S. The protocol evaluates this model on the remaining samples of the time series $S_{test} = [d_{N_{train}}, d_{|S|}[$ using an arbitrary score function, which takes as input the prediction of the model and the true label.

4.1 Model hyperparameter tuning

The initial part of our protocol investigates whether hyperparameter tuning can improve the baseline model and if repeating this tuning at each retraining improves the model's effectiveness.

We select the best hyperparameter tuning strategy across three strategies. "No tuning" uses the hyperparameters provided by our industrial partner. "Initial tuning" involves training the baseline model from scratch on the training set S_{train} with hyperparameter tuning, then keeping these hyperparameters fixed during evaluation. "Re-tuning" means tuning the hyperparameters each time the model is retrained, based on the data available at that time.

Hyperparameter tuning is performed using K-fold validation with time series splits and Bayesian search, aiming to maximize the average score function over the folds. Time series splits are preferred over random or stratified K-fold splits as they better evaluate the model's generalization to future samples, which is particularly useful for handling data drift and improving model robustness. We identify the **best tuning strategy** and reuse it for the rest of our protocol.

4.2 Training window size

We next need to use an appropriate window size is, i.e. how many of the most recent data model retraining should use. Training with the maximum amount of data does not always produce the best model, according to the dilemma between model plasticity (learning new information) and stability (retaining previous knowledge) [13,7].

We thus compare the effectiveness achieved by different window sizes that ranges from a fraction of the data up to all the available data. We select the best window size with periodic retraining schedule with different period ranging from a period as short as the label delay up to a period corresponding to a single retraining across the entire time series. We consider scenarios without and with delays. An evaluation without delays measures maximal potential performance improvements of alternative window sizes, whereas an evaluation with delays measures the improvement in a realistic context. Thanks to this step, we select the **best window size** and use it to evaluate retraining schedules.

4.3 Drift detector evaluation

We evaluate drift detectors and their parameters in a realistic scenario. We start from an initial model m_0 trained on the training set S_{train} with the best tuning strategy and window size (as explained before). Once a new sample $d_i = x_i, t_i, y_i$ arrives in the system, we use the initial model m_0 to predict a label $\hat{y}_i$ for x_i. We feed our drift detectors with the feature x_i, the time t_i, the label y_i and/or the prediction $\hat{y}_i$, depending on the detector type. If the detector does not detect a drift, we do nothing and wait for the next samples to arrive and be processed.

If we detect a drift, we retrain a model with the latest data $]d_{i-window}, d_i]$. In practice, we use a pre-trained model for evaluation such that d_i is rounded

up. The newly trained model will become available for prediction after a delay $\delta = \delta_l + \delta_d$ with regard to t'_i. Between the detection of drift in t_i and the model being available in time $t_i + \delta$, we continue to predict the samples with m_0.

Whether we continue to detect drift or not depends on the type of drift detector. If the drift detector is data-based, we continue to detect drift. Indeed, these detectors do not rely on the model and can continue to trigger model retraining during this time window. On the other hand, error-based and predictive-based detectors observe model properties (e.g. error rate, uncertainty). Therefore, after detecting a drift, we wait for the new model and distribution to be available.

After the first drift has occurred, for the remainder of the sample $d_j, j > i$ we use the latest available model to make the prediction and follow the same process as in m_0. A model m_i is available at time t_{m_i} if and only if it has been trained with data older than $t_m - \delta_l - \delta_d$.

With these steps, we can evaluate the **drift detectors** effectiveness (according to the score function) and efficiency (number of retraining).

4.4 Comparing drift detectors and periodic retraining

To evaluate the periodic retraining strategy, we use the same protocol as for the drift detectors and instantiate a drift detector that is equivalent to periodic retraining. Note that this detector behaves like a data drift detector and can detect drift before the latest available model is used. To compare the efficiency (number of models) and effectiveness (ML metric) of periodic retraining schedules, we vary the retraining intervals to identify the optimal Pareto front.

For each detector, we tune its parameters using Bayesian search to minimize the number of models used and maximize the ML effectiveness on the test set. To avoid information leakage, we split the S_{train} set into K-fold of (s_{train}, s_{val}) using the time series split. For each fold, we evaluate the effectiveness and efficiency of the drift detector parameters using the protocol of Section 4.3. For each detector, we select the parameters on the Pareto front of efficiency and effectiveness.

We evaluate the effectiveness and efficiency of the kept parameters of all drift detectors on the complete dataset. We build the Pareto front of all the detectors' efficiency and effectiveness and compare them with the effectiveness and efficiencies we obtain with periodic retraining schedules.

These steps lead to the **best drift detector** and its best parameters.

5 Experiments

Below, we outline our empirical study protocol and evaluate each step's impact.

5.1 Dataset, model and metrics

We apply our empirical study to the transaction system of BGL BNP Paribas. The objective of the ML model is to classify a transaction as accepted or refused based on the recent transaction of a particular client. The dataset contains

1,093,587 labeled inputs from transactions that occurred over a 5.6-year period. The timestamps associated with inputs are precise for a single day.

The classifier previously developed by our partner is a random forest. To comply with the regulation, our partner must have the capacity to interpret the automated decision made by the model; and tree-based models are interpretable by design and, therefore, we use a random forest architecture like our partner. Due to the sensitivity of the system, we did not work with the real model in production but have created a baseline model with the support of our partner's instructions. Therefore, we built a random forest classifier with 100 estimators up to 8-level deep. We trained the baseline model with 400,000 samples, following our partner's recommendations. The minimum period for periodic retraining corresponds to the average label delay, which is 5,293. For simplicity, we round it down to 5,000 in our experiments.

We use Matthew's correlation coefficient (MCC) to score the prediction of our models which is well suited for unbalanced datasets. It is important to note that, in our partner's case, even small differences in MCC correspond to a significant business impact. For example, during our experiments, we noticed that a difference in 0.01 MCC corresponds to 3,000 transactions on average.

Figure 1 (blue curve) shows the performance of the baseline model over time, without retraining. This reveals that the model performance is not stable over time and tends to degrade after a certain point due to distribution drifts. On the first 20,000 samples, our model has an MCC of 0.5595. Later, the MCC score ranges between 0.5169 and 0.6099, which is a significant difference business-wise and alerts our partner. We observe a peak performance on the batch $[740, 760[$. After this peak, performance tends to degrade until an MCC of 0.5443 for the last batch of our evaluation. The orange curve shows the performance of the best model in our study; this shows the potential benefits that appropriate retraining can produce.

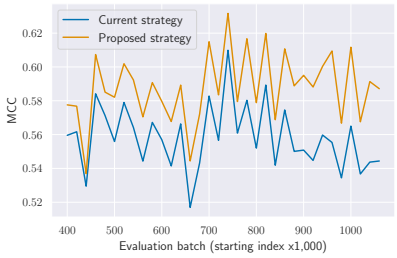


Fig. 1: Evolution of ML effectiveness (MCC) with time (in batches of 20,000 inputs).

Additionally, we evaluate the LCLD dataset [12]. The task of the ML model is to classify loan requests as accepted or refused based on information provided by the client (e.g. purpose of the loan), publicly available data (e.g. credit score), and computed features (e.g. installment). The dataset contains 1,124,606 labeled inputs from transactions that occurred over a 5-year period.[5]

5.2 Results

Tuning the hyperparameters of the initial model has a positive impact We first investigate the benefits of re-tuning the model hyper-parameters over

[5] Code available at `https://github.com/serval-uni-lu/drift-robustness`

time. We consider three different scenarios: 1) the baseline model of our partner is used throughout ("no tuning"); 2) we tune the model based on a time split[6] ("initial tuning"); and 3) the model is re-tuned each time it is re-trained ("re-tuning"). Since we want to assess the potential of model tuning to get better models over time, we ignore labeling and deployment delays at this stage.

Table 1: Impact of training strategy on ML effectiveness.

Dataset	Model hyperparameters	Retraining period 20,000	50,000	100,000	200,000	400,000
BGL	No tuning	0.5678	0.5656	0.5648	0.5627	0.5616
	Initial tuning	**0.5887**	**0.5877**	**0.5864**	**0.5855**	**0.5842**
	Re-tuning	0.5882	0.5867	0.5862	0.5848	0.5837
LCLD	No tuning	0.2691	0.2688	0.2681	0.2678	0.2669
	Initial tuning	0.2742	0.2737	0.2725	**0.2723**	**0.2709**
	Re-tuning	**0.2751**	**0.2744**	**0.2734**	0.2711	0.2685

Table 1 shows that initial model tuning yields significant improvements but that re-tuning at each retraining is not necessary. For our partner model, initial tuning always has a higher MCC than no-tuning and re-tuning regardless of the retraining period. For LCLD, initial tuning is more effective for large retraining periods while re-tuning is more effective with frequent retraining. However, the MCC gains are always under 0.001 between initial and re-tuning. For our study, this means that we may proceed with a careful **initial tuning and skip re-tuning the model each time we retrain it**.

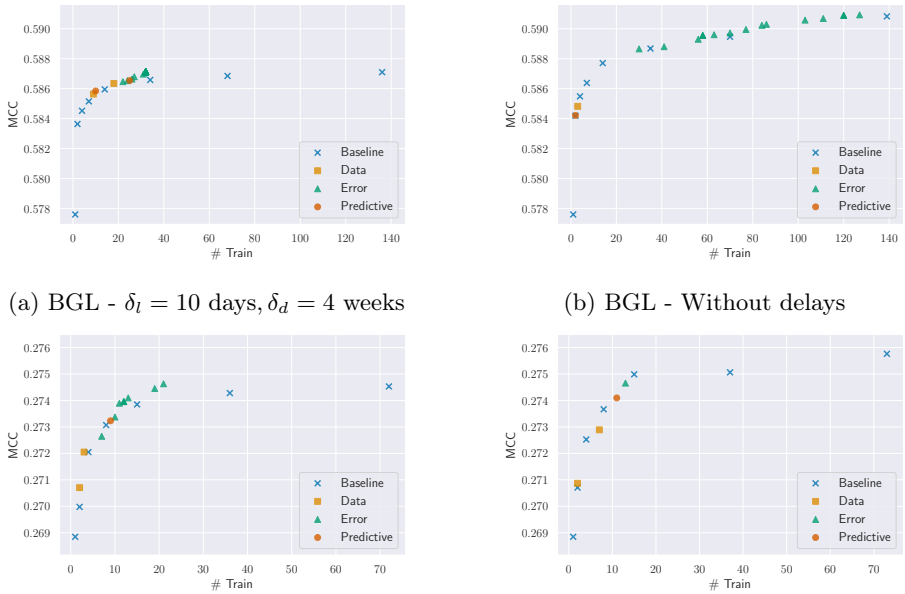
Training with all available data is counterproductive. We evaluate the impact of the retraining window on the ML effectiveness when using the simple periodic retraining strategy. As before, we study this impact in the ideal scenario without delay to measure the maximum potential gains and in a realistic scenario with our partner delays.

Table 2: Impact of ML effectiveness on window size. Average over periodic retraing with periods $p = \{5, 10, 20, 50, 100, 200, 400\} \times 10^3$.

Delay Window	BGL No	BGL Yes	LCLD No	LCLD Yes
50k	0.5737	0.5723	0.2726	0.2721
100k	0.5819	0.5804	0.2735	0.2729
200k	0.5867	0.5849	**0.2741**	**0.2732**
400k	**0.5875**	**0.5857**	0.2732	0.2726
All	0.5860	0.5841	0.2733	0.2730

Table 2 reveals that using 400k (respectively 200k) for our partner use case (respectively, LCLD) is the best retraining strategy on average. Breaking down

[6] The baseline model of our partner is tuned on non-time-sensitive k-fold validation.



(a) BGL - $\delta_l = 10$ days, $\delta_d = 4$ weeks

(b) BGL - Without delays

(c) LCLD - $\delta_l = 10$ days, $\delta_d = 4$ weeks

(d) LCLD - Without delays

Fig. 2: Pareto front of drift detectors, no retraining VS periodic retraining.

the results for each period on our partner use case reveals that using the 400k most recent samples to retrain is always among the 2 best solutions without delay and the best solution with delays, independently of the retraining period. We also observe that using all the available data is only the third best solution – for all periods except 50k – and is therefore not the optimal solution. Consequently, we empirically set the retraining window used in our experiments at 400k examples for our partner and 200k for LCLD.

Periodic retraining and error-based detectors together offer a flexible compromise between effectiveness and efficiency. We investigate the effectiveness and efficiency of drift detectors and periodic retraining schedules in scenarios with delays of BGL BNP Paribas, comparing their efficiency (number of retrainings) and effectiveness (MCC) in Figure 2 and Table 4 In Figure 2, each data point is a particular method setting, i.e. a scheduling method (periodic retraining or drift detector) with given parameter values. All scheduling methods (periodic and detectors) appear on the Pareto front, including the no-retraining strategy due to its inherent efficiency. We excluded drift detector schedules equivalent to "no retraining" or "always retraining". Figure 2a indicates that for a retraining budget above 22, error-based detectors are as effective as periodic retraining with fewer retrainings. For instance, the HDDM-W detector performs better with 32 retrainings compared to the most effective periodic

Table 3: For each schedule, the number of parameter settings (A) / (B). (A) are the settings on the efficiency/effectiveness Pareto front across all methods. (B) are the settings on the Pareto front local to the method. For varying delays, we report the number of Parameters on the Pareto front with delay δ_d that remains on the Pareto front for delays $\delta_d/2$ and $2\delta_d$. $\delta_l = 10$ days, $\delta_d = 4$ weeks.

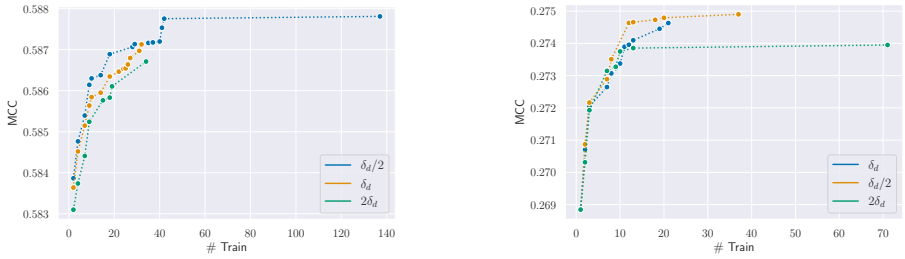
		BGL				LCLD			
Type	Schedule	No	δ_l δ_d	δ_l $\delta_d/2$	δ_l $2\delta_d$	No	δ_l δ_d	δ_l $\delta_d/2$	δ_l $2\delta_d$
Baseline	No detection	1 / 1	1 / 1	1	1	1 / 1	1 / 1	1	1
	Periodic	4 / 7	4 / 7	4	3	5 / 7	1 / 7	0	1
Data-based detector	Statistical test	0 / 25	0 / 25	0	0	0 / 9	0 / 9	0	0
	Divergence	1 / 4	2 / 6	2	2	1 / 4	1 / 5	2	1
	PCA-CD	0 / 3	0 / 4	0	0	1 / 2	1 / 2	1	1
Error-based detector	ADWIN (CE)	3 / 9	1 / 4	0	0	1 / 3	3 / 6	2	1
	ADWIN (PE)	1 / 9	1 / 3	0	0	0 / 5	0 / 5	0	0
	DDM	0 / 4	0 / 3	0	0	0 / 5	0 / 2	0	0
	EDDM	0 / 3	0 / 4	0	0	0 / 6	0 / 6	0	0
	HDDM-A	5 / 6	10 / 12	0	10	0 / 19	0 / 4	1	1
	HDDM-W	2 / 2	17 / 17	1	16	0 / 1	0 / 17	0	0
	KSWIN (CE)	1 / 8	0 / 7	0	0	0 / 7	2 / 5	3	0
	KSWIN (PE)	1 / 3	1 / 3	0	0	0 / 4	2 / 4	0	0
	Page-Hinkley (CE)	2 / 8	1 / 3	0	0	0 / 1	0 / 2	0	0
	Page-Hinkley (PE)	0 / 3	0 / 2	0	0	0 / 1	0 / 1	0	1
Predictive-based detector	Uncertainty	1 / 14	1 / 11	1	0	0 / 3	0 / 5	0	0
	Aries ADWIN	0 / 4	1 / 4	0	0	1 / 5	1 / 3	0	1

strategy requiring 136 retrainings. Periodic retraining is most effective for up to seven retrainings, with no clear advantage for either method between seven and 22 retrainings. Similarly, for LCLD, data-drift detectors and periodic retraining are most effective for fewer retrainings, and error-based detectors for higher numbers.

In the second column of Table 4, for each method, the right number shows the number of parameter settings that are Pareto-optimal *within* this method (i.e., the best settings of this particular method); the left number shows the number of these settings that are Pareto-optimal *across* all methods. The statistical test and PCA-CD detectors fail to reach the Pareto front for Partner, while the divergence detector succeeds. Error-based detectors DDM, EDDM, and Page-Hinkley (PE) also fail for both datasets, but predictive-based detectors reach the Pareto front.

5.3 Generalization study

Not considering delay overestimates the effectiveness/efficiency trade-off of retraining schedules. To emphasize the importance of considering the



(a) BGL (b) LCLD

Fig. 3: Pareto front of retraining schedule with different deployment delays.

delay in the evaluation of retraining schedules, we compare the drift detector and periodic retraining schedules with and without delay. We find that not considering delay overestimates the effectiveness/efficiency trade-off of retraining schedules. Indeed, comparing Figure 2a (respectively 2c) with Figure 2b (respectively 2d), we observe that the Pareto front lies higher when there is no delay. For example, on our partner's case the most effective strategy without delays has an MCC of 0.5909 for Page-Hinkley (CE), while with delays, the most effective strategy has an MCC of 0.5871 for HDDM-W.

The relative ranking between the methods also changes. Table 4 compares the number of method settings (for each method) that are on the Pareto front, in the cases without delays (left column) and with delays (right column). We see that the scheduling method settings on the Pareto front are different in the two cases. For instance, for our partner use case, KSWIN (CE) had one setting on the Pareto front without delay, but this setting disappears from the front when there is a delay. Hence, the optimal drift detection method settings without delay do not remain optimal if delays occur.

Change in deployment delay has an opposite effect on the effectiveness/efficiency Pareto front. We study the impact of varying delays on the effectiveness and efficiency of retraining scheduling methods by simulating scenarios where the deployment delay is increased or decreased after the methods are tuned and running. We tune the detectors based on the previously used delays δ_l and δ_d and then evaluate them when δ_d is halved or doubled. Figure 3 compares the Pareto fronts in the cases where δ_d is halves, unchanged and doubled. We observe that the $\delta_d/2$ front dominates the δ_d front, which itself dominates the $2\delta_d$ one. This indicates that a reduction in deployment delay (compared to the delay considered when tuning the drift detectors) yields improvement, whereas an augmented delay incurs a loss in the effectiveness/efficiency trade-off.

Changes in deployment delays disrupt the relative ranking of the retraining scheduling methods. Table 4 shows the number of schedules

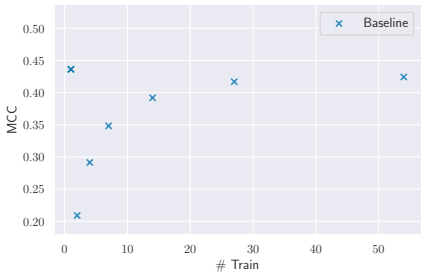
(method settings) that are on the Pareto front for the original delay δ_d (second column (A)) and how many of these exact schedules remain on the front for delays $\delta/2$ (third column) and 2δ (fourth column). A retraining schedule generalizes if it remains on the Pareto front in spite of the delay change. For our partner use case, we observe that only three methods generalize to the halved and doubled delays: periodic retraining, HDDM-W, and divergence. The uncertainty detector only generalizes to a reduction of the delays. All other methods do not generalize. For HDDM-W, the parameters that generalize when augmenting the delay are different than the ones when reducing the delay. On the LCLD dataset, only Divergence and ADWIN generalize to the halved and doubled delays.

5.4 Generalization beyond financial domain

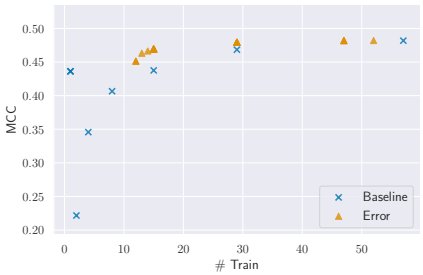
Table 4: For each schedule, the number of parameter settings (A) / (B). (A) are the settings on the efficiency/effectiveness Pareto front across all methods. (B) are the settings on the Pareto front local to the method. For varying delays, we report the number of Parameters on the Pareto front with delay δ_d that remains on the Pareto front for delays $\delta_d/2$ and $2\delta_d$. $\delta_l = 1$ day, $\delta_d = 10$ days.

Type	Detector	Delay: No	Delay: $\delta_l = 1$ day, $\delta_d = 9$ days	Delay: $\delta_d/2$	Delay: $2\delta_d$
Baseline	No detection	1 / 1	1 / 1	1	1
	Periodic	1 / 7	1 / 7	2	1
Data-based detector	Statistical test	2 / 9	6 / 7	4	7
	Divergence	1 / 4	0 / 5	1	0
	PCA-CD	0 / 0	0 / 0	0	0
Error-based detector	ADWIN (CE)	0 / 10	3 / 4	0	2
	ADWIN (PE)	0 / 4	3 / 4	0	0
	DDM	1 / 3	0 / 3	0	0
	EDDM	0 / 4	0 / 4	0	0
	HDDM-A	0 / 23	7 / 7	1	0
	HDDM-W	1 / 2	0 / 2	1	2
	KSWIN (CE)	0 / 5	3 / 5	0	0
	KSWIN (PE)	0 / 9	1 / 2	1	0
	Page-Hinkley (CE)	0 / 2	0 / 2	2	0
	Page-Hinkley (PE)	1 / 1	0 / 1	1	1
Predictive-based detector	Uncertainty	1 / 4	4 / 4	3	4
	Aries ADWIN	1 / 4	1 / 3	0	0

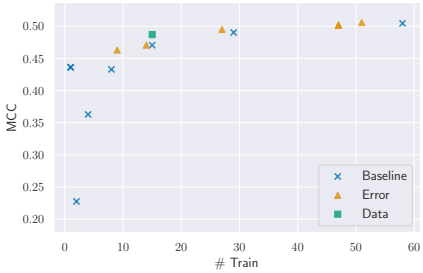
Electricity dataset Electricity [9] is a widely used dataset in the distribution shift on tabular data literature as shown in [14]. The classification task is to



(a) Electicity - $\delta_l = 10$ days, $\delta_d = 4$ weeks



(b) Electicity - $\delta_l = 1$ day, $\delta_d = 9$ days



(c) Electicity - Without delays

Fig. 4: Pareto front of drift detectors, no retraining and periodic retraining schedules on Electricity dataset.

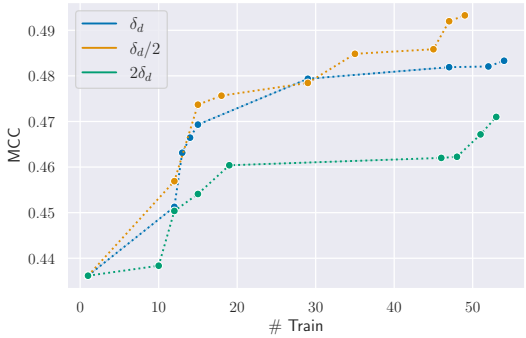


Fig. 5: Pareto front of retraining schedule with same parameters and different deployment delays.

determine at any point in time whether the electricity price is going up or down. The six features include the day of the week, the current price, the electricity demand, as well as the price, demand, and transfer of the adjacent geographical region. This dataset is smaller than the financial datasets, spans over a shorter

period, and is drawn from another domain. Electricity contains 45,312 labeled inputs that occur over 943 days with exactly one input recorded every 30 minutes. The dataset is precise to 30 minutes. Hence, the labeling and deployment delays may differ in a real-world system. The minimum period of retraining corresponds to the average label delay, which is 480 samples. We trained the baseline model with one year of data. We round up to the next multiple of 480 to facilitate model reuse during the experiments and obtain 17760 inputs.

Results We started with the same delays as for financial datasets ($\delta_l = 10$ days, $\delta_d = 4$ weeks). With these settings, we observe in Figure 4a that none of the schedules, including retraining every 10 days, can outperform the baseline. Not retraining and keeping the original model is the most effective strategy for such delays. This confirms the importance of considering the delay in the evaluation of retraining schedules. We also consider a scenario with shorter delays. We use a total delay of 10 days corresponding to our previous label delay. We assume that at any given time the electricity price of the previous day is available. Hence, we split this total delay into 1 day for labeling and 9 days for production. With these settings, we observe that error-based detectors are the best trade-off between accuracy and efficiency for 12 or more retrains. Below 12 retrains, not retraining remains the best strategy.

When comparing the occurrences on the Pareto front of each schedule strategy between the scenario with and without delays on Table 4 for Electricity, we confirm that the optimal schedule varies. KSWIN, HDDM-A, and ADWIN become relevant for Electricity. Similarly, changing the delay has an impact on which schedule strategy remains optimal. ADWIN and KSWIN (CE) are no longer on the Pareto front when halving the delay for Electricity and HDDM-A is no longer on the front when we double the delay.

Similarly to finance use cases, doubling or halving the delay has a significant impact on the Pareto-front of drift detectors. In Figure 5, the MCC decreases by more than 5% between scenarios $\delta_d/2$ and $2\delta_d$.

6 Conclusion

In this paper, we studied the performance of ML models with drift detector triggered retrain in the presence of delays. We considered industrial use cases where the label arrives ten days after the prediction and the model goes through a four-week validation phase before deployment. We evaluated 15 detectors in two use cases and different delay scenarios. Our results show that drift detectors and scheduling strategies are particularly affected by realistic delays.

Acknowledgments

This project is Supported by the Luxembourg National Research Fund, grant BRIDGES/2022/IS/17437536.

References

1. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software Engineering for Machine Learning: A Case Study. In: Proceedings - ICSE-SEIP 2019. pp. 291–300 (2019)
2. Baena-García, M., Campo-Ávila, J., Fidalgo-Merino, R., Bifet, A., Gavald, R., Morales-Bueno, R.: Early drift detection method (01 2006)
3. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the 7th SIAM International Conference on Data Mining. pp. 443–448 (2007)
4. Bifet, A., Gavaldà, R.: Learning from Time-Changing Data with Adaptive Windowing. In: Proceedings of the 2007 SIAM International Conference on Data Mining. pp. 443–448. Society for Industrial and Applied Mathematics (Apr 2007)
5. Blanchard, G., Lee, G., Scott, C.: Generalizing from several related classification tasks to a new unlabeled sample. In: NeurIPS (2011)
6. Frias-Blanco, I., Campo-Avila, J.d., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A., Caballero-Mota, Y.: Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. IEEE Transactions on Knowledge and Data Engineering **27**(3), 810–823 (Mar 2015)
7. Gama, A., Bifet, A., Barcelona, R.: A survey on concept drift adaptation. ACM Comput. Surv **46** (2014)
8. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with Drift Detection. vol. 8, pp. 286–295 (Sep 2004)
9. Harries, M.B.: Splice-2 comparative evaluation: Electricity pricing (1999), https://api.semanticscholar.org/CorpusID:151207670
10. Hu, Q., Guo, Y., Xie, X., Cordy, M., Ma, L., Papadakis, M., Traon, Y.L.: Aries: Efficient Testing of Deep Neural Networks via Labeling-Free Accuracy Estimation (Feb 2023)
11. Inc., E.A.: Evidently ai: Data drift algorithm (2021)
12. Kaggle: All Lending Club loan data (2019)
13. Lim, C.P., Harrison, R.: Online pattern classification with multiple neural network systems: an experimental study. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **33**(2), 235–247 (May 2003)
14. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under Concept Drift: A Review. IEEE Transactions on Knowledge and Data Engineering **31**(12), 2346–2363 (Dec 2019)
15. Masud, M., Gao, J., Khan, L., Han, J., Thuraisingham, B.M.: Classification and novel class detection in concept-drifting data streams under time constraints. IEEE Transactions on knowledge and data engineering **23**(6), 859–874 (2010)
16. Page, E.S.: Continuous Inspection Schemes. Biometrika **41**(1/2), 100–115 (1954)
17. Plasse, J., Adams, N.: Handling delayed labels in temporally evolving data streams. In: 2016 IEEE International Conference on Big Data (Big Data). pp. 2416–2424. IEEE (2016)
18. Poenaru-Olaru, L., Cruz, L., van Deursen, A., Rellermeyer, J.S.: Are Concept Drift Detectors Reliable Alarming Systems? – A Comparative Study (Nov 2022)
19. Qahtan, A.A., Alharbi, B., Wang, S., Zhang, X.: A PCA-Based Change Detection Framework for Multidimensional Data Streams. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 935–944. ACM, Sydney NSW Australia (Aug 2015)

20. Raab, C., Heusinger, M., Schleif, F.M.: Reactive Soft Prototype Computing for Concept Drift Streams. Neurocomputing **416** (Apr 2020)
21. Shaker, M.H., Hüllermeier, E.: Aleatoric and Epistemic Uncertainty with Random Forests (Jan 2020)
22. Storkey, A., et al.: When training and test sets are different: characterizing learning transfer. Dataset shift in machine learning **30**(3-28), 6 (2009)
23. Van Looveren, A., Klaise, J., Vacanti, G., Cobb, O., Scillitoe, A., Samoilescu, R., Athorne, A.: Alibi detect: Algorithms for outlier, adversarial and drift detection (2019)
24. Zhou, K., Liu, Z., Qiao, Y., Xiang, T., Loy, C.C.: Domain generalization: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence p. 1–20 (2022). https://doi.org/10.1109/tpami.2022.3195549, http://dx.doi.org/10.1109/TPAMI.2022.3195549
25. Žliobaité, I.: Change with Delayed Labeling: When is it Detectable? In: 2010 IEEE International Conference on Data Mining Workshops. pp. 843–850 (Dec 2010)