OPTWIN: Drift identification with optimal sub-windows

Mauro Dalle Lucca Tosi
mauro.dalleluccatosi@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Martin Theobald
martin.theobald@uni.lu
University of Luxembourg
Esch-sur-Azette, Luxembourg

ABSTRACT

Online Learning (OL) is a field of research that is increasingly gaining attention both in academia and industry. One of the main challenges of OL is the inherent presence of *concept drifts*, which are commonly defined as unforeseeable changes in the statistical properties of an incoming data stream over time. The detection of concept drifts typically involves analyzing the *error rates* produced by an underlying OL algorithm in order to identify if a concept drift occurred or not, such that the OL algorithm can adapt accordingly. Current concept-drift detectors perform very well, i.e., with *low false negative rates*, but they still tend to exhibit *high false positive rates* in the concept-drift detection. This may impact the performance of the learner and result in an undue amount of computational resources spent on retraining a model that actually still performs within its expected range. In this paper, we propose OPTWIN, our "OPTimal WINdow" concept drift detector. OPTWIN uses a sliding window of events over an incoming data stream to track the errors of an OL algorithm. The novelty of OPTWIN is to consider *both the means and the variances* of the error rates produced by a learner in order to split the sliding window into two provably optimal sub-windows, such that the split occurs at the earliest event at which a statistically significant difference according to either the t- or the f-tests occurred. We assessed OPTWIN over the MOA framework, using ADWIN, DDM, EDDM, STEPD and ECDD as baselines over 7 synthetic and real-world datasets, and in the presence of both sudden and gradual concept drifts. In our experiments, we show that OPTWIN surpasses the F1-score of the baselines in a statistically significant manner while maintaining a lower detection delay and saving up to 21% of time spent on retraining the models.

KEYWORDS

Concept drift, Drift detection, Data streams, Change detection

1 INTRODUCTION

Online Learning (OL) is an area of research which gained an increasing amount of attention both in academia and industry over the past years. OL is a sub-field of Machine Learning (ML) in which an underlying ML technique aims to constantly update its model's parameters from an incoming data stream under limited time and space constraints. Ideally, OL methods are trained in real time and thereby aim to maximize the prediction accuracy of their models based on bounded windows of data instances they have previously seen, and which they may see in the near future [15]. Common use-cases of OL include social-networks analyses, various kinds of IoT-related services and predictions, spam detection, online fraud detection (e.g., credit-card transactions), and many others.

The presence of *concept drifts* is one of the numerous challenges when processing data streams in real-time. Concept drifts are commonly defined as unforeseeable changes in the statistical properties of the incoming data stream over time [17]. Such a change may have different sources and types, and it may involve different adaptation strategies (cf. Section 2.1). In practice, concept drift is the key phenomenon that impairs the performance of pre-trained (and hence static) ML models over data streams. Take as an example the spam-detection use-case studied by Fdez-Riverola et al. [10]. Spammers are known to constantly adapt their spamming strategy (thus creating concept drifts intentionally) to overcome the spam filters. Therefore, if we train a spam filter based on a pre-defined dataset, it will perform as expected just until the spammers create a new form of spam that bypasses the pre-trained filter. Thus, if the pre-trained filter fails to detect a concept drift and, consequently, adapt itself accordingly, its performance may be drastically impacted.

To detect concept drifts, one may rely on one of the following two principal types of drift detectors (or on a combination of them): (1) *error rate-based*, which use the difference between the number or magnitude of historical prediction errors and new prediction errors to determine if a concept drift occurred [17]; and (2) *data distribution-based*, which use distance metrics to determine if the distribution from the historical data instances and the new data instances statistically changed [17]. We here highlight two fundamental differences between both approaches. First, as prediction errors already provide a form of aggregated measure for the performance of an ML model, it is usually computationally more expensive to track changes on features derived from the data stream (data distribution-based) rather than on the error rates (error rate-based). Second, error-based approaches need labels from the ML models, whereas data distribution-based ones do not. Note that, error-based approaches use labels from the ML problem to calculate the error rate of the ML models; thus, no concept drift-related labels are assumed to be available. This paper focuses on the drift identification of supervised ML problems, which already assume the availability of labels for training the ML models. Therefore, considering its lightweight performance compared to the data distribution-based approaches, we further focus on error rate-based methods.

The most popular among the error rate-based algorithms are ADWIN [5], DDM [11], and their variations as [3, 4]. ADWIN maintains a variable window W of the past error rates produced by the learner. Then, it checks, at each iteration, if there are two sub-windows $W = W_0 W_1$ whose absolute difference among their *mean*

This paper is published in 2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW), Utrecht, Netherlands, 2024, pp. 331-337, DOI: 10.1109/ICDEW61823.2024.00049.

values μ_{W_0}, μ_{W_1} of errors exceeds a given threshold ϵ. If it finds two such sub-windows, with $|\mu_{W_0} - \mu_{W_1}| \geq \epsilon$, a drift is flagged and W is shrunk to discard the staled data. DDM, on the other hand, tracks the error rate p of the learner by its *standard deviation* s. A drift is flagged at iteration i iff the current error rate p_i surpasses the minimum previously recorded error rate p_{min} by over 3 times its minimum standard deviation s_{min}, i.e., $p_i + s_i \geq p_{min} + 3\, s_{min}$. Both methods use the errors' standard deviations only to set their thresholds and guarantee the statistical significance of the errors' differences. Thus, they do not interpret changes in those standard deviations as potential concept drifts.

As opposed to previous approaches, we argue that also changes in the standard deviations of past prediction errors should trigger concept drifts. Take, as a simple example, a regressor that outputs the following errors at window W_0:

$$W_0 = \langle 0.3; 0.7; 0.7; 0.3; 0.3; 0.7; 0.5; 0.5 \rangle$$

While, at window W_1, it outputs the following errors:

$$W_1 = \langle 0.0; 1.0; 1.0; 0.0; 1.0; 0.0; 0.0; 1.0 \rangle$$

Current drift detectors like ADWIN would not consider this as a concept drift, as the error means μ_{W_0}, μ_{W_1} over both windows are equal to 0.5, whereas we—intuitively—understand that something changed and, therefore, a concept drift should be flagged.

In this paper, we propose the "OPTimal WINdow" drift detector (OPTWIN). It is also a sliding-window algorithm which analyzes the error rates (either binary or real-valued) produced by an underlying ML algorithm. It calculates the *optimal cut* of a sliding window W to divide it into two sub-windows W_{hist} and W_{new}. It then performs the well-known t- (for means) and f-tests (for standard deviations) to determine whether the two sub-windows exhibit a statistically significant difference in either their means or standard deviations, respectively. OPTWIN has the following two main features: (1) it can calculate the *optimal cut* based only on the length of W; (2) it uses both the errors' means and standard deviations to flag drifts. Furthermore, we provide detailed and rigorous guarantees of OPTWIN's performance in terms of its true-positive (TP), false-positive (FP) and false-negative (FN) rates, as well as in its drift-identification delay.

To assess OPTWIN, we used the popular MOA framework [6] and compared OPTWIN to all major drift-detection frameworks provided in the literature: ADWIN [5], DDM [11], EDDM [3], STEPD [22] and ECDD [23]. Using MOA, we compared the *precision*, *recall* and *F1-score*, as well as the *delay* of all drift detectors over both sudden and gradual drifts. Furthermore, we also trained a Naive Bayes (NB) classifier on MOA that adapts itself based on the drift detectors and compared its results on various synthetic datasets (STAGGER [25], RANDOM RBF [7], and AGRAWAL [2]) and real-world ones (Covertype and Electricity) [8, 13]. Furthermore, we likewise assessed OPTWIN on a Neural Network (NN) use-case. Specifically, we pre-trained a Convolutional NN (CNN) with the CIFAR-10 [16] image dataset and simulated an end-to-end OL scenario with concept drifts by using OPTWIN and ADWIN as detectors.

Our results show that OPTWIN reliably identifies both sudden and gradual drifts across all datasets. Furthermore, OPTWIN is the drift detector with the best F1-score when compared to the baselines. This is due to its higher precision, which indicates a low FP rate. With respect to the drift-identification delay, there was no drift detector that was superior to the others in a majority of the datasets. In terms of run-time, OPTWIN in combination with the CNN training pipeline is 21% faster than a similar ADWIN pipeline due to OPTWIN's lower FP rate, which leads to a significantly reduced amount of retraining iterations. Therefore, our experiments indicate that OPTWIN identifies concept drifts with a similar delay as other drift detectors but maintains a higher precision and recall in the drift detection, which reduces the overall run-time of OL pipelines that trigger a re-training of their models upon each detected concept drift.

2 BACKGROUND & RELATED WORKS

We next formally introduce the problem of concept drift and review its principal characteristics (cf. Section 2.1). We also discuss the most important related approaches for concept-drift detection from the literature, which serve as inspiration and baselines of our work throughout this paper (cf. Section 2.2).

2.1 Background

A concept drift may exhibit different characteristics whose understanding is essential for a fast and reliable detection. Consider an unbounded data stream that receives *features* $x_i \in X$ and *labels* $y_i \in Y$ in the form of pairs of instances (x_i, y_i). At *time* t, these instances follow a certain distribution $P_t(X, Y)$. A concept drift is the change of this distribution over time. Formally, a concept drift occurs at time $t + 1$ iff $P_t(X, Y) \neq P_{t+1}(X, Y)$ [12, 17, 18]. The underlying data distribution $P_t(X, Y)$ may also be decomposed and expressed as a product of probabilities $P_t(X, Y) = P_t(X) \times P_t(Y|X)$ via the well-known *chain rule of probability*. Thus, a concept drift may come from two *sources*: (1) $P_t(X) \neq P_{t+1}(X)$, which is known as a *virtual drift* because it does not impact the decision boundaries of the learner; and (2) $P_t(Y|X) \neq P_{t+1}(Y|X)$, which is known as an *actual drift* because it directly impacts the learner's accuracy [12, 17, 18]. It is also possible to have both drift sources simultaneously [17].

Another important characteristic of a concept drift is its *type*. Concept drifts can be (1) *sudden*; (2) *incremental*; (3) *gradual*; and (4) *reoccurring* (see Figure 1) [12, 17]. Sudden drifts occur when the probability distribution changes completely within a single step. Incremental drifts occur when the distribution $P_t(X, Y)$ changes incrementally until its convergence. Gradual drifts occur when the new distribution gradually replaces the old one. Reoccurring drifts occur when distributions can reoccur after some time.

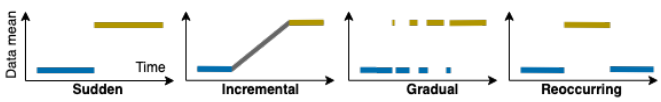

Figure 1: Concept drift types.

To recover from concept drifts, one may adapt the learner according to two main strategies: (1) *active drift adaptation*, which triggers drift adaptation only after a concept drift is flagged by the drift detector; and (2) *passive drift adaptation*, in which the learner is constantly adapting itself, independently of the detection of drifts [14]. The active drift-adaptation strategy depends on the correct identification of the drifts but avoids spending computational resources and time on (re-)training a learner that has already converged. Therefore, we further focus on the active drift-adaptation strategy in the following.

There are many sub-strategies for active drift adaptation. The selection of the appropriate one depends not only on the type and source of the drifts but also on the use-case and the adopted learner. A common strategy is to train a new model with the latest data points whenever a drift is identified. Another strategy is to adjust the current learner instead of training a new one from scratch. Moreover, it is also possible to have an ensemble of learners, which is primarily useful for adapting to reoccurring drifts [17].

The most popular procedure to evaluate learning algorithms that handle concept drifts is the *prequential procedure*. Prequential is an evaluation scheme in which each data point is used for testing before training the learning algorithm. Therefore, it is not necessary to know when a drift occurred to perform the evaluation, which is useful when using real-world datasets that do not have labeled drifts [17]. Regarding the metrics used to evaluate the drift detector, one may refer to the following ones: TPs, FPs and FNs (and hence *precision*, *recall* and *F1-score*) in the detection rates, as well as the *delay* of the detection [9, 17].

2.2 Related Works

ADWIN [5] (for "ADaptive WINdowing") is an algorithm with a sliding window W that takes as input a confidence level $\delta \in (0, 1)$ and an unbounded sequence of real values $X = \langle x_1, x_2, ..., x_i, ... \rangle$ from a data stream. Each input x_i is expected to be in $[0, 1]$ and is stored in a finite window $W \subset X$. The idea of the algorithm is that a concept drift occurs whenever there are two subsequent sub-windows of W with an average difference in their mean above ϵ_{cut}, which is guaranteed to yield a *confidence level* of δ. If a drift is detected, a part of W is dropped, and the algorithm is ready to resume the drift detection. ADWIN has guaranteed bounds on all error distributions. However, the observed errors usually tend to follow a normal distribution. Thus, they tightened their ϵ_{cut} accordingly and evaluated the algorithm taking those new bounds. Furthermore, as ADWIN checks multiple sub-windows of W, its complexity is $O(\log |W|)$ per iteration (where an iteration is ideally triggered for each new element that arrives from the stream).

DDM [11] (for "Drift Detection Method") also receives as parameter a value δ related to the confidence level of the algorithm, and a sequence of data instances X which are available one by one from a data stream. However, the data stream is expected to follow a binomial distribution, as its values represent the number of errors in a sample of n instances. Therefore, it is primarily suited for classification problems rather than for regression. The main idea of DDM is that, as the number of samples increases, the number of errors from a learner will decrease. Therefore, if the error rate from the learner increases above a certain threshold, it means that a concept drift occurred. Thus, DDM tracks the minimal error rate p_{min} from X and its respective standard deviation s_{min}. Then, DDM flags a drift at step $i \geq 30$ iff $p_i + s_i \geq p_{min} + \delta s_{min}$. When a drift is detected, the algorithm is reset. One advantage of DDM is its lightweight nature due to storing only p_{min} and s_{min}. However, DDM is known for its decreased performance when detecting slow gradual changes.

EDDM [3] (for "Early Drift Detection Method") is similar to DDM but uses the distance between errors to identify concept drifts. Therefore, EDDM is suited only for binary input data, which does not allow it to be used for regression problems. They assume that, as the number of examples increases, the distance between errors will increase accordingly. Therefore, if this distance decreases more than a threshold, a concept drift is flagged. Thus, EDDM tracks the maximum distance between errors p'_{max} and its respective standard deviation s'_{max}. Then, after analyzing at least 30 errors, EDDM flags a drift at step t if $(p'_t + 2\,s'_t)/(p'_{max} + 2\,s'_{max} < \delta)$, with δ again representing a simple form of confidence level for the detected concept drifts. According to the authors, EDDM improves DDM's performance when identifying gradual concept drifts while maintaining good performance on sudden drifts.

STEPD [22] (for using the "Statistical Test of Equal Proportions") is a lightweight drift detector that also considers a sequence of data instances X as input. The idea of STEPD is that the recent accuracy of a learner shall be similar to its overall accuracy, otherwise a concept drift occurred. As the name suggests, STEPD checks the equality-of-proportions hypothesis test [20] by comparing the most recent values from X (kept in a sliding window W) with the other values in X, thus comparing $X_{0:W}$ and $X_{W:|X|-1}$. It also considers that the data follows a normal distribution, and with a confidence level of δ, it can determine if the null hypothesis is rejected and a concept drift is flagged. In this case, the algorithm is reset. The value of W selected by the authors was 30.

ECDD [23] is the acronym for "Exponentially Weighted Moving Average" (EWMA) for concept-drift detection. It is a drift-detection method based on EWMA that checks the misclassification rate of its underlying learners. ECDD receives a sequence of data instances X as input, which it assumes to follow a Bernoulli distribution. ECDD is suited only for classification problems, as it only accepts binary data from the data stream. Other than X, ECDD takes as inputs ARL_0, which is the time one expects between detecting false positives; and λ, which dictates how much weight new data has compared to the old one. Despite being computationally expensive to calculate, ARL_0 can be approximated before starting to process the data stream. ECDD then uses pre-calculated polynomials, ARL_0 and λ to update its estimators. With its updated estimators, ECDD detects if a concept drift occurred or not. Finally, if a drift is detected, the algorithm is reset.

We note that all algorithms described above additionally have a form of warning detection which can be used to assist in the concept drift adaptation. The warning is usually a simple relaxation of the drift threshold which can be used to start training a new learner before the actual concept drift is flagged.

3 OPTWIN – OPTIMAL WINDOW CONCEPT DRIFT DETECTOR

OPTWIN is the new concept-drift detector that we propose in this paper. It is an error rate-based drift detector, which tracks the error rates produced by an OL learner in a sliding window W. Its name stands for "OPTimal WINdow" due to its calculation of the optimal split of the sliding window. We consider this split "optimal" because it is detected for the first element in W at which a statistically significant difference between either the means or the variances of the error rates (based on the common t- and the f-tests) among two sub-windows of W occurs. Specifically, W keeps growing until either a concept drift is detected or a maximum user-defined size w_{max} is reached. Based on the sliding-window size $|W|$ and a predefined confidence level δ, OPTWIN calculates the optimal point to divide W into "historical" (W_{hist}) and "new" (W_{new}) data points.

3.1 Definition of concepts

- δ: confidence level defined by the user, $\delta \in (0, 1)$.
- W: sliding window of error rates.
- w_{proof}: minimum size of W to identify drifts of ρ magnitude.
- w_{min}: minimum W size, with $w_{min} = 30$.
- w_{max}: maximum W size, with $w_{max} \in [w_{min}, \infty)$.
- μ_W: mean of W.
- σ_W: standard deviation of W.
- ν: optimal splitting percentage of W, with $\nu \in (0, 1)$.
- ν_{split}: optimal splitting point of W, with $\nu_{split} = \lfloor \nu\,|W| \rfloor$
- W_{hist}: historical errors from W, defined as $W_{0:\,\nu_{split}}$.
- W_{new}: new errors from W, defined as $W_{\nu_{split}:\,|W|-1}$.
- ρ: robustness parameter, i.e., the ratio that $\mu_{W_{new}}$ has to vary in relation to $\sigma_{W_{hist}}$ to count as a concept drift, with $\rho\ in(0, \infty)$.
- ρ_{temp}: temporary ρ, defined by solving Equation 1, with $\nu = 0.5$.
- $X = x_1, x_2, ..., x_i, ...$: data-stream of real numbers x_i.

3.2 Setup & Parameters

Without loss of generality, we consider a typical OL scenario, in which OPTWIN receives as input a sequence of real numbers $x_1, x_2, ..., x_i, ...$ from a possibly unbounded data-stream X. Thus, the algorithm is assumed to access only one element x_i at time i from the stream, and buffers previously seen elements in a *sliding window* $W \subset X$ of consecutive events. Moreover, OPTWIN requires as parameters (1) δ, the *confidence level* for the concept-drift detection; (2) w_{max}, the *maximum size* of the sliding window W; and (3) ρ, a parameter we refer to as *robustness*, which we define as the minimum ratio by which $\mu_{W_{new}}$ has to vary in relation to $\sigma_{W_{hist}}$ in order for OPTWIN to consider this variation as a concept drift.

3.2.1 Assumptions.

- There is no concept drift within the first w_{min} data instances from X (which is needed to initialize OPTWIN).
- A concept drift occurs when the means or the standard deviations within two sub-windows of W are statistically different.
- Within any two sub-windows of W, the values produced by the test statistic of the unequal-variance t-test [24] (henceforth called "t_value") follow a t-distribution.
- Within any two sub-windows of W, the values produced by the test statistic of the f-test [19] (henceforth called "f_value") follow an f-distribution.

Our first assumption mitigates the negative impact of outliers when a few data points are available. The second assumption points out when we expect OPTWIN to identify concept drifts. The third and the fourth assumptions are generally required to guarantee the validity of the t- and f-tests, respectively, but do not impose limitations on the values ingested from the data stream in practice.

3.3 Algorithm

Algorithm 1 OPTWIN

Input parameters:
- δ – confidence level
- ρ – robustness
- w_{max} – max window size

Global variables:
$W = \langle\,\rangle$ – sliding window
$w_{min} = 30$ – min window size
$\eta = 1e^{-5}$ – avoids division by 0

procedure AddElement(x_i)
  $W \leftarrow W \cup x_i$
  if $|W| < w_{min}$ **then**
    return False
  else if $|W| \geq max\_lenght$ **then**
    $W \leftarrow W - W_0$
  $\nu \leftarrow$ OptimalCut($|W|, \rho, \delta^{\frac{1}{4}}$) cf. Equation (1)
  $\nu_{split} \leftarrow \lfloor \nu\,|W| \rfloor$
  $W_{hist} \leftarrow W_{0:\nu_{split}}$
  $W_{new} \leftarrow W_{\nu_{split}:|W|-1}$
  // *f-test*
  if $\frac{(\sigma_{W_{new}}+\eta)^2}{(\sigma_{W_{hist}}+\eta)^2} > f\_ppf(\delta^{\frac{1}{4}}, \nu|W|-1, (1-\nu)|W|-1)$ **then**
    reset()
    return True
  // *t-test* – cf. Equations (1) and (2)
  else if $t\_value(W_{hist}, W_{new}) > t\_ppf(\delta^{\frac{1}{4}}, df)$ **then**
    reset()
    return True

As seen in Algorithm 1, OPTWIN first needs to be initialized by creating an empty sliding window W. Analogously to other drift detectors, it collects w_{min} many first elements from the stream (usually within 30 to 50) to guarantee that it has a minimum amount of data to output statistically relevant drift detections. It also defines a small constant $\eta = 1e^{-5}$, which avoids a division by 0 when added to the standard deviations during the calculation of the f-test.

Second, the AddElement procedure is called once to process each data element received from the data stream. The procedure starts by inserting the most recent data element x_i into W and by checking whether W has already reached the minimum amount of elements to detect a possible concept drift. If so, it also checks if W increased above w_{max} and removes the oldest element from W if necessary. Up to this point, OPTWIN performs standard operations to maintain its sliding window bounded between w_{min} and w_{max} elements.

Then, OPTWIN calculates ν, which is the optimal splitting point of W into W_{hist} and W_{new}, by solving Equation 1 for the highest

value of ν in terms of δ', ρ and $|W|$:

$$\rho = t\_ppf(\delta', df)\sqrt{\frac{1}{\nu|W|} + \frac{f\_ppf(\delta', \nu\,|W|-1, (1-\nu)\,|W|-1)}{(1-\nu)\,|W|}} \tag{1}$$

where ρ is the aforementioned, user-defined robustness parameter. During this calculation, OPTWIN uses t_ppf and f_ppf, which are the Probability Point Functions (PPF) of the t- and the f-distributions, respectively[1].

To calculate t_ppf, we apply a confidence of $\delta' = \delta^{\frac{1}{4}}$, which considers the application of the two tests to calculate ν in Equation 1 and the two tests on Lines 11 and 14 in Algorithm 1. Equation 2 depicts our calculation of the degrees of freedom df, which is needed as another parameter for the t-distribution:

$$df = \frac{(\frac{1}{\nu|W|} + \frac{f\_ppf}{(1-\nu)|W|})^2}{\frac{(\frac{1}{\nu|W|})^2}{(\nu|W|)-1} + \frac{(\frac{f\_ppf}{(1-\nu)|W|})^2}{((1-\nu)|W|)-1}} \tag{2}$$

To calculate f_ppf, on the other hand, we take $\nu|W|-1$ and $(1-\nu)|W|-1$ as the degrees of freedom for the f-distribution and again apply the same confidence value δ' also here.

Finally, OPTWIN performs the t- and f-tests (in any order) to compare both sub-windows and determine if their means and standard deviations, respectively, belong to the same distribution. If not, a concept drift is flagged and the algorithm is reset. Otherwise, the AddElement method is called for the next element from the data stream in an iterative manner.

The usage of the t- and f-tests to identify significant changes in the means and standard deviations among series of data values is well-established. However, OPTWIN's novelty comes from the combination of both tests to calculate ν, and thereby identify where to optimally divide this series of values to perform both tests. In short, by solving Equation 1 in terms of ν, we determine the minimum size of W_{new} (the optimal splitting point of W) that statistically guarantees the identification of any concept drift with a robustness of at least ρ when using the t- and f-tests, thereby achieving lower drift-detection delays than other approaches. To calculate ν, other than the confidence level δ, the only values that shall be inputted by the user are w_{max} and ρ. Regarding ρ, when selecting a small value, one may expect to identify smaller drifts in exchange of a higher drift-detection delay. On the other hand, with higher values of ρ, one can expect a smaller detection delay in exchange of missing smaller drifts. In practice, ρ shall be set based on the magnitude and frequency of drifts expected. However, determining the correct ρ to each use case should not be a difficult task, since different ρ's tend to produce similar results (as seen in Section 4). Regarding w_{max}, with a higher value, one may expect smaller drift-detection delays in exchange for more memory usage. In practice, when using $\rho = 0.1$, we did not observe a significant variation in $|W_{new}|$ even when increasing w_{max} to more than 25,000 elements.

One important point to note is that we can only calculate ν as the optimal splitting point if W is large enough. Otherwise, we set ν to the middle of W until it grows to the minimum size needed to solve Equation 1. Thus, if it exists, it is defined as the highest root of Equation 1. Otherwise, it is set to $\nu = 0.5$.

Below, we present Theorem 3.1, our main theoretical result. It gives guarantees in terms of OPTWIN's false positive (FP) and false negative (FN) bounds for identifying concept drifts.

Theorem 3.1.

- ***False Positive Bound.*** *At every step, if μ_W and σ^2_W remain constant within W, OPTWIN will flag a concept drift at this step with a confidence of at most 1-δ.*
- ***False Negative Bound*** *(for mean drift with large enough W). For any partitioning of W into two sub-windows W_{hist} W_{new}, with $|W| \geq w_{proof}$ and W_{new} containing the most recent elements, if $\mu_{hist} - \mu_{new} > \rho\,\sigma_{hist}$, then, with confidence δ, OPTWIN flags a concept drift in at most $|W| - \nu_{split}$ steps.*
- ***False Negative Bound*** *(for mean drift with small W). For any partitioning of W into two sub-windows W_{hist} W_{new}, with $w_{min} \leq |W| < w_{proof}$ and W_{new} containing the most recent elements, if $\mu_{hist} - \mu_{new} > \rho_{temp}\,\sigma_{hist}$, then, with confidence δ, OPTWIN flags a concept drift in at most $\frac{|W|}{2}$ steps.*
- ***False Negative Bound*** *(for standard-deviation drift with any W). For any partitioning of W into two sub-windows W_{hist} W_{new}, with $|W| \geq w_{min}$ and W_{new} containing the most recent elements, if $\frac{\sigma^2_{new}}{\sigma^2_{hist}} > f\_ppf(\delta', \nu\,|W|-1, (1-\nu)\,|W|-1)$, then, with confidence δ, OPTWIN flags a concept drift in at most ν_{split} steps.*

3.3.1 Proof of Theorem 3.1.

- **Part 1:** In each iteration of OPTWIN, we apply the t- and the f-tests to identify the concept drifts. Therefore, the false-positive bounds of OPTWIN are directly drawn from those tests. Thus, with $\delta' = \delta^{\frac{1}{4}}$ as the confidence for each of the four tests, we identify a false-positive drift with a confidence of at most $1-\delta$.
- **Part 2:** Considering an unequal-variance t-test [24] by comparing W_{hist} and W_{new}, we have:

$$t\_value = \frac{\mu_{hist} - \mu_{new}}{\sqrt{\frac{\sigma^2_{hist}}{|W_{hist}|} + \frac{\sigma^2_{new}}{|W_{new}|}}} \tag{3}$$

The nominator on the right represents the difference observed between the means of W_{hist} and W_{new}. If this difference is large enough, we can statistically guarantee that both sets have distributions with different means. To simplify the equation, we represent this difference in terms of σ_{hist}. Therefore, we set $\mu_{hist} - \mu_{new} = \rho\,\sigma_{hist}$ with ρ being a user-defined robustness parameter. Thus, by applying this substitution and passing the denominator to the other side of the equation, we have:

$$\rho\,\sigma_{hist} = t\_value\sqrt{\frac{\sigma^2_{hist}}{|W_{hist}|} + \frac{\sigma^2_{new}}{|W_{new}|}} \tag{4}$$

Considering ν as the splitting point of W into W_{hist} and W_{new}, we have $|W| = \nu\,|W_{hist}| + (1-\nu)\,|W_{new}|$, and therefore:

$$\rho\,\sigma_{hist} = t\_value\sqrt{\frac{\sigma^2_{hist}}{\nu\,|W|} + \frac{\sigma^2_{new}}{(1-\nu)\,|W|}} \tag{5}$$

[1] We omit the arguments of f_ppf in the following equation to improve readability.

By our definition of a concept drift, σ_{new} and σ_{hist} must follow the same distribution (otherwise a drift occurred). Furthermore, we also consider that they follow the f-distribution, which is usually the case for standard deviations. Thus, consider the definition of the test statistic for the f-test [19] as shown below:

$$f\_value = \frac{\sigma^2_{new}}{\sigma^2_{hist}} \tag{6}$$

We can now determine σ_{new}'s upper bound in terms of σ_{hist} and by defining f_factor as the highest value that the f_value can have while maintaining the null hypothesis of the f-test with confidence δ'.

$$\sigma^2_{hist}\, f\_factor \geq \sigma^2_{new} \tag{7}$$

In this case, the f_factor can be calculated using the f_ppf, the PPF of the F-curve, by taking a confidence of δ' and $|W_{new}| - 1$ and $|W_{hist}| - 1$ as the degrees of freedom for the numerator and the denominator, respectively (cf. Equation 6). Thus, we obtain:

$$f\_factor = f\_ppf(\delta', \nu\,|W| - 1, (1-\nu)\,|W| - 1) \tag{8}$$

Therefore, to later simplify Equation 5, we substitute σ_{new} with its upper bound as calculated by the f-test on Equation 7:

$$\rho\,\sigma_{hist} = t\_value\sqrt{\frac{\sigma^2_{hist}}{\nu\,|W|} + \frac{\sigma^2_{hist}\, f\_factor}{(1-\nu)\,|W|}} \tag{9}$$

Next, we can simplify the equation by removing σ_{hist} from both sides as follows:

$$\rho = t\_value\sqrt{\frac{1}{\nu\,|W|} + \frac{f\_factor}{(1-\nu)\,|W|}} \tag{10}$$

Then, we calculate the t_value in Equation 11 analogously by using the PPF of the T-curve.

$$t\_value = t\_ppf(\delta', df) \tag{11}$$

We consider a confidence level of δ' and the degrees of freedom df for the unequal-variance t-test (cf. Equation 12).

$$df = \frac{(\frac{\sigma^2_{hist}}{|W_{hist}|} + \frac{\sigma^2_{new}}{|W_{new}|})^2}{\frac{(\frac{\sigma^2_{new}}{|W_{hist}|})^2}{|W_{hist}|-1} + \frac{(\frac{\sigma^2_{new}}{|W_{new}|})^2}{(|W_{new}|)-1}} \tag{12}$$

By considering Equation 12 in our setting with $|W_{hist}| = \nu\,|W|$ and $|W_{new}| = (1-\nu)\,|W|$ and our upper bound for $\sigma_{new} \leq \sigma_{hist}\, f\_factor$ (cf. Equation 7), we reach the formerly presented Equation 2.

Finally, by replacing t_value on Equation 10 by the one calculated on Equation 11, we reach Equation 1 introduced in Section 3.3. □

We remark that Equation 1 only depends on δ', ρ, $|W|$ and ν. Recall that δ' is $\delta^{\frac{1}{4}}$ which is provided by the user; ρ is also given by the user; and $|W|$ increases linearly until w_{max}. We can then calculate the highest value ν based on each $|W|$ that solves Equation 1.

Therefore, considering the split $W = W_{hist}\, W_{new}$ with $W_{hist} = W_{0:\nu\,|W|}$ and $W_{new} = W_{\nu\,|W|:|W|-1}$, by applying the t-test, we guarantee that if μ_{hist} and μ_{new} diverge more than $\rho\,\sigma_{hist}$, we identify this divergence with a confidence of δ. Moreover, considering our OL setting, it takes no more than $(1-\nu)|W|$ iterations for W to be divided into $W_{hist}\, W_{new}$ after the drift occurred.

- **Part 3:** By setting $\nu = 0.5$ in Equation 10, we have:

$$\rho_{temp} = t\_value\sqrt{\frac{1}{0.5\,|W|} + \frac{f\_factor}{0.5\,|W|}} \tag{13}$$

  With $\nu = 0.5$ and ρ_{temp} as calculated above, we can guarantee that if μ_{hist} and μ_{new} diverge more than $\rho_{temp}\,\sigma_{hist}$, we identify the divergence with a confidence of δ (as guaranteed by the test procedure). Similarly to Part 2 of the proof, we achieve this in at most $(1-\nu)\,|W|$ iterations, in this case, $\frac{|W|}{2}$ iterations. □

- **Part 4:** This bound is directly obtained from the f-test. By simply applying the f-test to compare two sub-windows of W, we guarantee the given confidence value, in this case δ. Again, it takes no more than $(1-\nu)\,|W|$ iterations for W to be divided into $W_{hist}\, W_{new}$ after the drift occurred. □

3.4 Implementation & Analysis

We implemented OPTWIN via a combination of Java and Python scripts as extensions of the MOA [6] (Java) and the River [21] (Python) libraries. We pre-calculated the values of ν, t_ppf, and f_ppf based on a fixed confidence value of $\delta = 0.99$ and for $w_{max} = 25,000$, thus $30 \leq |W| \leq 25{,}000$. All pre-calculated variables were stored in lists indexed by the $|W|$ from which they were calculated. This is possible because those variables, as seen in Equation 1, do not depend on the data distribution. Thus, it is not necessary to calculate them in real-time. On the other hand, we need to store the sliding window W plus the other three lists of floating point values of size up to w_{max} in memory, which takes a maximum of $w_{max} * 4 * 4$ bytes. Thus, for $w_{max} = 25{,}000$, OPTWIN would require only around 390 KB of memory.

Furthermore, the full calculation of the means and standard deviations from W_{hist} and W_{new} can be avoided. Instead of calculating them from scratch, one only needs to update them incrementally. Moreover, as W is bounded by w_{max}, we can use a circular array to make insertions at the end of the array, deletions from the beginning of the array, and then look up each value in $O(1)$ time. Therefore, the AddElement procedure has an overall computational complexity of $O(1)$ per element ingested from the data stream (assuming a constant cost for the numerical operations involved in resolving Equation 1 to ν). Our analytical approach thus provides great potential runtime gains over the iterative search procedure applied, e.g., by ADWIN which requires $O(\log |W|)$ computations per iteration.

By default, OPTWIN's Algorithm 1 tracks concept drifts that either increase or decrease the means and standard deviations of the variables tracked. However, in an OL scenario, we usually want to update the learner only when the number of errors increases. Therefore, in our implementation, we check if also $\mu_{new} \geq \mu_{hist}$ along with the statistical tests on Lines 11 and 14 of Algorithm 1. In doing so, we consider that a drift occurred only if μ_{new} is higher than μ_{hist} (i.e., the learner decreased in performance).

4 EXPERIMENTS & RESULTS

In this section, we report our detailed experiments to asses OPTWIN. We compared OPTWIN to the most-commonly used baselines for concept-drift detection: ADWIN, DDM, EDDM, STEPD, and ECDD. We performed most of the experiments using the common MOA [6] framework which is a Java-based data stream simulator. In addition, we performed experiments on Python to asses OPTWIN on a Neural Network (NN) use case, which would not be possible via MOA.

4.1 MOA Experiments

We compared 3 configurations of OPTWIN with the default configurations of our baselines. All OPTWIN configurations had $\delta = 0.99$, $w_{max} = 25{,}000$, and pre-computed values for v, t_ppf, and f_ppf (as described on Section 3.4). The difference among the OPTWIN configurations is only on ρ, which we varied between 0.1, 0.5, and 1.0 to better understand how our robustness parameter affects OPTWIN's performance in practice.

We performed two types of experiments on MOA. The first one uses the "Concept Drift" interface, in which MOA creates a stream of data points (either binary or non-binary) and produces a concept drift that can be sudden or gradual. We later refer to those experiments according to their data input and their drift type.

The second type of experiment uses the "Classification" interface. It generates data streams based on synthetic datasets (STAGGER, RANDOM RBF, and AGRAWL) [2, 7, 25] and real-world ones (Electricity and Covertype) [8, 13]. The idea of these experiments is to train a classifier that is reset every time a concept drift is detected by a drift detector. We chose MOA's built-in Naive Bayes (NB) classifier for its simplicity, which facilitates the analysis of the drift-detection results. For the synthetic datasets, we generate data streams with 100,000 data points with drifts occurring every 20,000 data points (either sudden or gradual). For the real-world data sets, the drifts are already present and have an unknown location on the stream.

Table 1 presents the comparison among drift detectors on the above-mentioned configurations. We repeated each experiment 30 times and compared their average TP, FP and FN rates to compute their micro-average precision, recall, and F1-score, along with their average drift-detection delay (in terms of the number of streamed elements between the occurrence of a known concept drift and its identification by the detector). Note that we did not include in Table 1 the "Classification" experiments on real-world datasets nor the ones with gradual concept drifts. For the real-world datasets, it is not possible to calculate the above-mentioned metrics without knowing the drift's locations. For the gradual drifts, we observed a divergence between the starting and ending points of those drifts in the MOA documentation and in practice. Therefore, we did not include those configurations in our comparison.

Based on the results in Table 1, we can generally observe a high F1-score for OPTWIN when compared to the baselines. In fact, with $\rho \leq 0.5$, OPTWIN's average F1-score is over 95%, followed by OPTWIN with $\rho = 1.0$ achieving 89,2%, DDM with 89,7%, ADWIN reaching 67% and the other detectors less than 50%. We assert this due to OPTWIN's low FP rate which produces a high precision and F1-score, respectively. We further compared the F1-scores of all configurations of OPTWIN with the two drift detectors that can be used for regression problems (ADWIN and STEPD), which showed OPTWIN to be superior based on a one-tailed Wilcoxon signed-rank test with $\alpha = 0.05$ in a statistically significant manner.

Regarding the drift-detection delay, OPTWIN with $\rho = 0.5$ is the drift detector with the smallest delay, taking on average 121 iterations to detect a concept drift; followed by ADWIN, ECDD, and OPTWIN at $\rho = 0.1$ taking 132, 132 and 156 iterations each; the

Experiment	Drift Detector	Delay	FP	P	R	F1
	ADWIN	280	16.33	43%	100%	60%
	DDM	365	0.37	88%	100%	93%
gradual	EDDM	148	6.33	56%	100%	71%
binary	STEPD	180	8.00	23%	100%	37%
drift	ECDD	**117**	5.37	27%	100%	42%
	OPTWIN $\rho = 0.1$	328	0.30	88%	100%	94%
	OPTWIN $\rho = 0.5$	278	0.10	96%	100%	98%
	OPTWIN $\rho = 1.0$	317	0.07	97%	100%	**99**%
	ADWIN	145	0.00	100%	100%	**100**%
gradual	STEPD	41	150	10%	100%	18%
non-binary	OPTWIN $\rho = 0.1$	**2.00**	0.00	100%	100%	**100**%
drift	OPTWIN $\rho = 0.5$	**2.00**	0.00	100%	100%	**100**%
	OPTWIN $\rho = 1.0$	**2.00**	0.00	100%	100%	**100**%
	ADWIN	46.33	1.87	35%	100%	52%
	DDM	64.50	0.20	83%	100	91%
sudden	EDDM	26.77	6.57	13%	100%	23%
binary	STEPD	**0.40**	6.67	71%	100%	83%
drift	ECDD	5.47	2.67	27%	97%	42%
	OPTWIN $\rho = 0.1$	75.13	0.00	100%	100%	**100**%
	OPTWIN $\rho = 0.5$	28.17	0.00	100%	100%	**100**%
	OPTWIN $\rho = 1.0$	18.33	0.33	75%	100%	86%
	ADWIN	25.00	2.00	33%	100%	50%
sudden	STEPD	21.00	32.00	3%	100%	6%
non-binary	OPTWIN $\rho = 0.1$	**1.00**	0.00	100%	100%	**100**%
drift	OPTWIN $\rho = 0.5$	**1.00**	0.00	100%	100%	**100**%
	OPTWIN $\rho = 1.0$	**1.00**	0.00	100%	100%	**100**%
	ADWIN	31.00	0.30	93%	100%	96%
	DDM	6.33	0.17	96%	100%	98%
sudden	EDDM	40.01	0.00	100%	100%	**100**%
STAGGER	STEPD	17.39	31.40	11%	99%	20%
	ECDD	0.72	0.47	90%	100%	94%
	OPTWIN $\rho = 0.1$	0.76	0.27	94%	100%	97%
	OPTWIN $\rho = 0.5$	**0.72**	0.43	90%	100%	95%
	OPTWIN $\rho = 1.0$	**0.72**	0.60	87%	100%	93%
	ADWIN	169.67	6.00	33%	75%	46%
	DDM	536.33	2.00	60%	75%	67%
sudden	EDDM	**53.55**	11.00	21%	75%	33%
RANDOM	STEPD	281.25	25.00	14%	100%	24%
RBF	ECDD	71.75	174.00	2%	100%	4%
	OPTWIN $\rho = 0.1$	315.00	0.00	100%	75%	**86**%
	OPTWIN $\rho = 0.5$	187.00	0.00	100%	75%	**86**%
	OPTWIN $\rho = 1.0$	1,931.50	1.00	67%	50%	57%
	ADWIN	**229.40**	4.23	49%	100%	65%
	DDM	1,875.87	0.90	78%	80%	79%
sudden	EDDM	5,370.47	17.20	12%	60%	20%
AGRAWAL	STEPD	838.03	23.47	14%	99%	25%
	ECDD	465.90	153.57	3%	100%	5%
	OPTWIN $\rho = 0.1$	371.62	0.63	86%	100%	**93**%
	OPTWIN $\rho = 0.5$	350.89	0.77	83%	93%	88%
	OPTWIN $\rho = 1.0$	630.63	0.27	93%	88%	91%

Table 1: Statistics of drift identification on synthetic datasets.

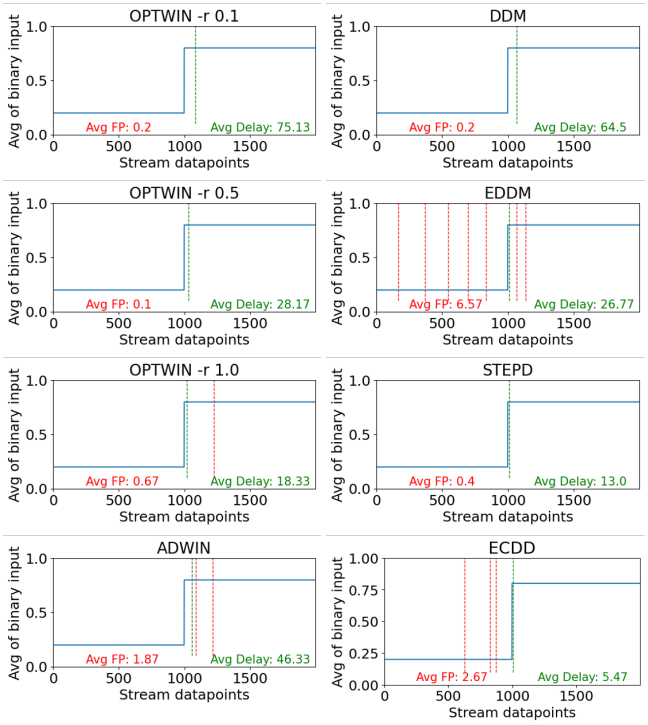

Figure 2: Sudden binary drift detection with average FP rates compared to drift-detection delays.

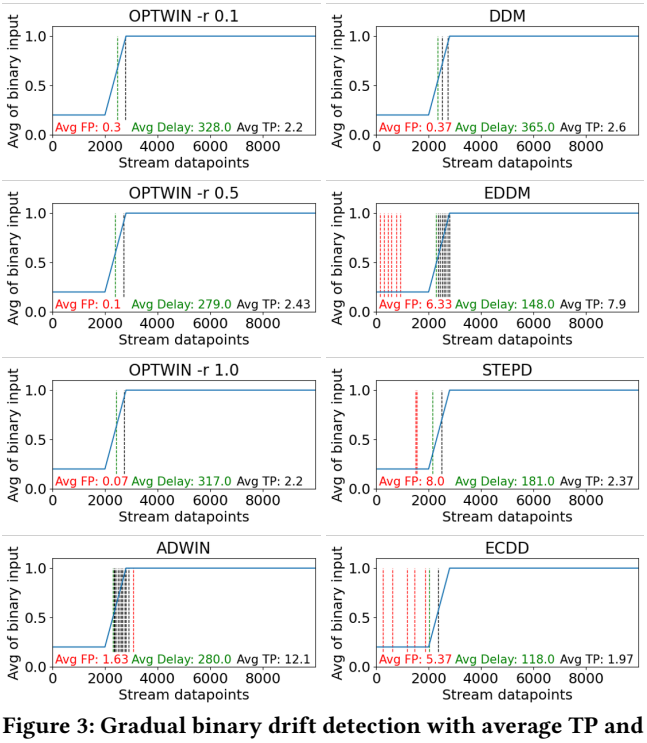

Figure 3: Gradual binary drift detection with average TP and FP rates compared to drift-detection delays.

other detectors took on average between 198 and 1127 iterations to detect a drift, with OPTWIN at $\rho = 1.0$ taking 414 on average. Observe that, the higher the FP rate of a drift detector, the higher are also its chances of identifying a drift earlier. Therefore, when analyzing ECDD's average drift delay of 132 iterations, we have to take into consideration that it had an average of 67 FPs per run, in contrast to an average of less than 0.4 for any of the OPTWIN configurations. Furthermore, considering that DDM, EDDM, and ECDD depend on binary input data, they were not included in the experiments using the "non-binary" datasets.

We can better visualize the difference between the drift detectors in Figures 2 and 3, which represent one of the 30 runs of the "sudden binary drift" and the "gradual binary drift" configurations, respectively (we selected the runs with results closest to the ones on the observed average). In Figure 2, we can see the relatively high FP rate of the EDDM, ADWIN and ECDD detectors when compared to OPTWIN, DDM and STEPD. Moreover, we can see that OPTWIN's detection delay decreases as ρ increases. In Figure 3, we cannot observe the same effect of ρ in OPTWIN's delay detection. Nevertheless, we can still observe the high FP rate of EDDM, ECDD and ADWIN compared to the other drift detectors. We note that STEPD's high FP rate comes from a few runs with dozens of FPs.

In Figure 4, we can once again see OPTWIN and DDM identifying all the drifts with a low FP rate. However, in this scenario, ADWIN also produces only a few FPs, which we noticed to occur immediately after correctly identifying a concept drift. This is due to the time it takes to adapt its window size and remove data points from before the concept drift. In this use case, ECDD and STEPD produce almost "random guesses" on the location of the concept drifts.

4.2 Classification Experiments

In the "Classification" experiments (still using MOA as platform), we can analyze the average accuracy achieved by the NB classifier when varying the drift detectors (cf. Table 2). The idea is that the better a drift detector's performance, the better the classifier can adapt to the concept drifts, thus generating a higher prediction accuracy. However, the fast and accurate detection of the drifts did not always traverse to a better accuracy for the classifier. We can observe this by comparing Tables 1 and 2, in which experiments using drift detectors that produced a low F1-score in Table 1 achieved good accuracy in Table 2. Moreover, the drift detectors with the best accuracy on real-world data sets were the ones that detected more drifts, which is why ECDD achieved such good accuracy (being the detector that produced the higher amount of FPs). For example, ECDD detected 426 drifts on the Electricity dataset—more than twice as many as all other drift detectors.

4.3 Neural Network Experiments

To further explore OPTWIN's behavior in a regression scenario, we compared it with ADWIN for identifying drifts from the loss of a CNN. We chose ADWIN as our baseline because it was the drift detector with the best F1-score among the ones that do not require binary inputs (thus excluding DDM, EDDM and ECDD). To simplify the reader's understanding of the generation of the concept drift, instead of training a regression problem on a CNN, we here focused on image classification by swapping the labels of the images. Thus,

Drift Detector	Sudden drift			Gradual drift			Real-world datasets	
	STAGGER	Random RBF	AGRAWAL	STAGGER	Random RBF	AGRAWAL	Electricity	Covertype
No drift detector	66.31	58.69	57.94	66.31	58.69	57.93	73.36	60.52
ADWIN	99.89	69.74	**70.22**	98,78	69,48	69,89	80.01	82.49
DDM	**99.96**	68.18	65.53	98,67	68,19	65,80	81.18	88.03
EDDM	99.87	65.99	65.46	**98,83**	66,92	65,31	84.83	86.08
STEPD	98.74	**70.92**	69.48	98,78	**70,49**	69,13	84.49	87.53
ECDD	**99.96**	68.64	67.03	98,71	68,52	66,80	**86.76**	**90.16**
OPTWIN $\rho = 0.1$	**99.96**	69.65	70.11	98,72	69,54	**69,94**	79.99	83.29
OPTWIN $\rho = 0.5$	**99.96**	69.77	69.91	98,51	69,48	69,27	83.30	85.63
OPTWIN $\rho = 1.0$	**99.96**	68.67	69.44	98,36	68,48	68,61	82.96	86.31

Table 2: Accuracy of NB on synthetic and real-world datasets.

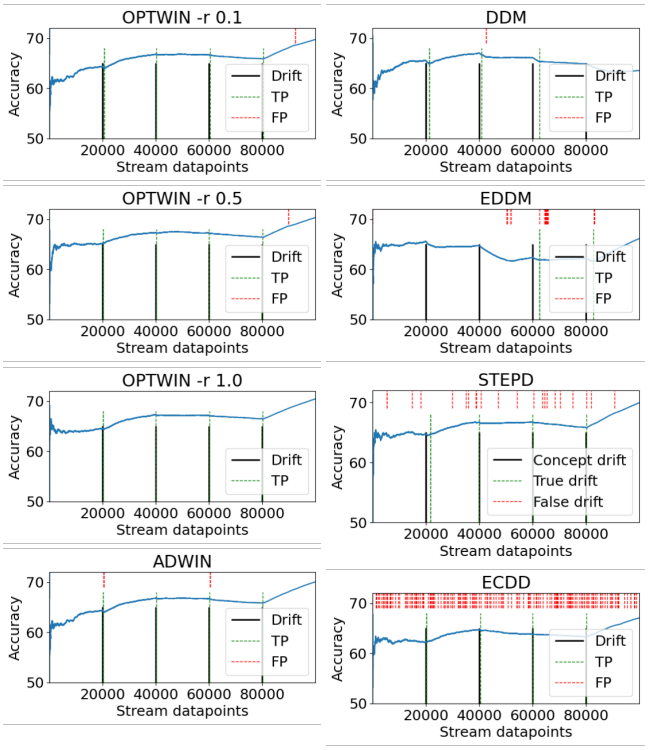

Figure 4: TP and FP rates for drift detection on the AGRAWAL dataset with sudden concept drifts.

we provoked 4 concept drifts by swapping the labels of two classes of images every 20% of the simulated data stream. For example, after 62,480 iterations, we swapped the labels between images from "cats" to "horses". Nevertheless, as the variable tracked by the drift detectors is the loss of the CNN, the type of the problem or the network architecture should not affect the experiment.

We pre-trained an image classification model [1] using the CIFAR-10 [16] data set during 100 epochs, achieving an average of 89% training accuracy over 3 different runs. Then, we simulated an OL scenario with concept drifts. Our data stream was formed by batches of 32 images from the CIFAR-10 dataset; with a total of 312,400 data points (equivalent to 100 epochs). We simulated our 4 concept drifts by swapping the labels of two classes every 62,480 iterations (the equivalent of 20 epochs). At every iteration, the model classified the 32 images and outputted the loss of the batch. We inputted this loss into the drift-detection algorithm. If a drift

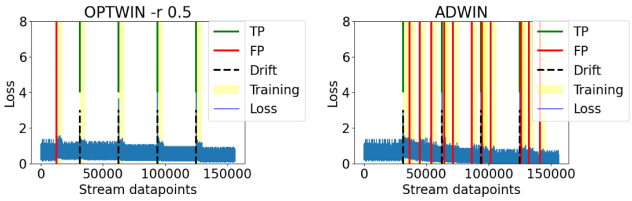

Figure 5: Sudden drift detection over the loss of a NN.

was detected, the next 9,372 batches of images (the equivalent of 3 epochs) were used for fine-tuning the model (thus adapting it to the concept drift). Therefore, the goal was for the drift detector to identify the 4 concept drifts that we simulated, triggering the fine-tuning of the model for a total of 12 epochs (3 epochs per drift).

In Figure 5, we compare OPTWIN and ADWIN under the setting described above. First, we note that ADWIN's high FP rates made it retrain the model for much longer than OPTWIN. In comparison, ADWIN identified 15 concept drifts (with 11 FPs), thus triggering model fine-tuning for 61,562 iterations which took in total 945 seconds. In contrast, OPTWIN identified just 5 drifts (with 1 FP), thus triggering the model fine-tuning for 23,430 iterations which took 781 seconds. We note that OPTWIN's running time per iteration is superior to ADWIN, $1e^{-5}$ against $6e^{-6}$ seconds. However, OPTWIN still ends up speeding up the OL pipeline whenever retraining is triggered by the concept-drift detection (21% faster in this use case). This is because the training of a learner is usually computationally more expensive than the drift detection. Thus, with fewer FPs, the total training time can be reduced substantially.

5 CONCLUSION

In this paper, we presented OPTWIN, a novel concept-drift detector that uses a sliding window of errors to identify concept drifts with a low FP rate. OPTWIN's novelty relies on the assumption that changes also in the variances of the elements ingested from a data stream can be an indication of a concept drift. This assumption lets it optimally divide its sliding window and apply the t- and f-tests to determine whether a concept drift occurred. To assess OPTWIN, we compared it with 5 popular drift detectors in 11 different experiments. As a result, OPTWIN achieved higher F1-scores in most of our experiments. In fact, OPTWIN had the best F1-score (with statistical significance) while maintaining a similar drift-detection delay compared to the drift detectors suited for both classification and regression problems. Moreover, OPTWIN could speed up the overall OL pipeline of a CNN by 21 (compared to ADWIN) due to

its low FP rate. To conclude, we conjecture that OPTWIN's characteristics can enable even higher speed-ups in scenarios where the drift identification triggers the re-training of complex models.

ACKNOWLEDGMENTS

This work is funded by the Luxembourg National Research Fund under the PRIDE program (PRIDE17/12252781).

REFERENCES

[1] 2022. Convolutional Neural Network (CNN); Tensorflow Core. https://www.tensorflow.org/tutorials/images/cnn

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. 1993. Database mining: A performance perspective. *IEEE transactions on knowledge and data engineering* 5, 6 (1993), 914–925.

[3] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno. 2006. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, Vol. 6. 77–86.

[4] Roberto SM Barros, Danilo RL Cabral, Paulo M Gonçalves Jr, and Silas GTC Santos. 2017. RDDM: Reactive drift detection method. *Expert Systems with Applications* 90 (2017), 344–355.

[5] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 443–448.

[6] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *J. Mach. Learn. Res.* 11 (2010), 1601–1604. https://doi.org/10.5555/1756006.1859903

[7] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavalda. 2009. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 139–148.

[8] Jock A Blackard and Denis J Dean. 1999. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture* 24, 3 (1999), 131–151.

[9] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. 2006. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*. Citeseer, 1–24.

[10] Florentino Fdez-Riverola, Eva Lorenzo Iglesias, Fernando Díaz, José Ramon Méndez, and Juan M Corchado. 2007. Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications* 33, 1 (2007), 36–48.

[11] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Brazilian symposium on artificial intelligence*. Springer, 286–295.

[12] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[13] Michael Harries, New South Wales, et al. 1999. Splice-2 comparative evaluation: Electricity pricing. (1999).

[14] Moritz Heusinger, Christoph Raab, and Frank-Michael Schleif. 2022. Passive concept drift handling via variations of learning vector quantization. *Neural Computing and Applications* 34, 1 (2022), 89–100.

[15] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. 2021. Online learning: A comprehensive survey. *Neurocomputing* 459 (2021), 249–289.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[17] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2346–2363.

[18] Ning Lu, Jie Lu, Guangquan Zhang, and Ramon Lopez De Mantaras. 2016. A concept drift-tolerant case-base editing technique. *Artificial Intelligence* 230 (2016), 108–133.

[19] Mohammad Mahbobi and Thomas K Tiemann. 2016. *Introductory business statistics with interactive spreadsheets-1st Canadian edition*. Campus Manitoba. 55–63 pages.

[20] Adam Massey and Steven J Miller. 2006. Tests of hypotheses using statistics. *Mathematics Department, Brown University, Providence, RI* 2912 (2006), 1–32.

[21] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, et al. 2021. River: machine learning for streaming data in python. *The Journal of Machine Learning Research* 22, 1 (2021), 4945–4952.

[22] Kyosuke Nishida and Koichiro Yamauchi. 2007. Detecting concept drift using statistical testing. In *International conference on discovery science*. Springer, 264–269.

[23] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. 2012. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters* 33, 2 (2012), 191–198.

[24] Graeme D Ruxton. 2006. The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test. *Behavioral Ecology* 17, 4 (2006), 688–690.

[25] Jeffrey C Schlimmer and Richard H Granger. 1986. Incremental learning from noisy data. *Machine learning* 1, 3 (1986), 317–354.

Received 01 February 2023