Online Linear Models for Edge Computing

Hadar Sivan[1](✉), Moshe Gabel[2], and Assaf Schuster[1]

[1] Technion - Israel Institute of Technology, Haifa 3200, Israel
{hadarsivan,assaf}@cs.technion.ac.il
[2] University of Toronto, Toronto, Canada
mgabel@cs.toronto.edu

Abstract. Maintaining an accurate trained model on an infinite data stream is challenging due to concept drifts that render a learned model inaccurate. Updating the model periodically can be expensive, and so traditional approaches for computationally limited devices involve a variation of online or incremental learning, which tend to be less robust.
The advent of heterogeneous architectures and Internet-connected devices gives rise to a new opportunity. A weak processor can call upon a stronger processor or a cloud server to perform a complete batch training pass once a concept drift is detected – trading power or network bandwidth for increased accuracy.
We capitalize on this opportunity in two steps. We first develop a computationally efficient bound for changes in any linear model with convex, differentiable loss. We then propose a sliding window-based algorithm that uses a small number of batch model computations to maintain an accurate model of the data stream. It uses the bound to continuously evaluate the difference between the parameters of the existing model and a hypothetical optimal model, triggering computation only as needed.
Empirical evaluation on real and synthetic datasets shows that our proposed algorithm adapts well to concept drifts and provides a better tradeoff between the number of model computations and model accuracy than classic concept drift detectors. When predicting changes in electricity prices, for example, we achieve 6% better accuracy than the popular EDDM, using only 20 model computations.

1 Introduction

Consider a computationally limited device like a wireless sensor or a router that receives an infinite stream of (occasionally) labeled samples, and applies machine learning to perform tasks such as gesture recognition or network attack detection, or employs it as part of a mobile healthcare application [12, 20]. Classic offline learning algorithms assume a fixed distribution of the data to make some guarantees about the accuracy of learned model. However, this is not always the case in data streams, where the underlying distribution may change over time. This is known as a *concept drift*. To maintain an accurate model, the device has to update the model whenever the concept changes, a computationally expensive task.

Concept drift has been widely studied. Algorithms designed for learning from data streams with concept drifts rely on two main strategies. *Incremental learning algorithms* [32, 31, 27, 24, 7] adapt to the new concept implicitly by updating the model periodically. They incrementally update the model using only the previous model and a single new sample from the stream rather than an entire batch of samples. However, for stochastic gradient descent, which is a popular learning method for incremental algorithms, the convergence rate is approximately linearly dependent on the condition number of the problem [6, p. 467]. Concept drifts such as changes in variable scaling or covariance structure can increase this condition number, causing slower adaptation to the new concept (since incremental algorithms process one sample at a time). Conversely, algorithms based on either *sliding windows* or *adaptive windows* use a batch of recent samples to compute the current model [30, 3]. Such algorithms are more immune to outliers since multiple samples are used simultaneously. They also explicitly forget irrelevant samples, as the computed model is based only on samples that appear in a recent window. Despite these advantages, sliding window algorithms are less computationally efficient than incremental algorithms. As such, they are difficult to use in settings with low-powered devices such as those used in edge computing and IoT (Internet of Things) [29, 20].

Connected devices in edge computing and IoT settings present a new opportunity to tradeoff communication or battery power for better accuracy. These often have limited computational power, but are connected to stronger machines. Smart cities, for example, are composed of many weak sensors which use a cloud server to perform learning tasks [2]. Thus, weak edge devices can occasionally call on stronger machines for heavy computational tasks such as batch learning.

However, many weak devices could flood the network and overwhelm the cloud. This gives rise to a tradeoff between accuracy and the network overhead (or required computations). A similar tradeoff exists in *heterogeneous architectures*, which incorporate power-efficient weak processors and power-hungry strong processors on the same device. These are common in edge computing settings where client or edge devices are often battery-powered [19]. The weak processor can wake the strong processor to perform computationally intensive tasks such as recomputing the model, but with higher power consumption. Algorithms must therefore be carefully designed to minimize model recomputations by efficiently detecting when they are necessary.

Our Contributions We present DRUiD (for **D**rift detecto**R** from bo**U**nded **D**istance): a novel sliding window algorithm designed for learning from data streams in edge computing and IoT settings. DRUiD is suitable for any linear model with convex differentiable loss, while supporting both classification and regression tasks.

We develop a bound that estimates the difference between the last batch-computed model and the hypothetical model that could be computed from the current position of the sliding window. While most other algorithms monitor the error rate of the model to detect concept drifts, DRUiD monitors changes to the

model coefficients. By only recomputing models as needed, DRUiD reduces the number of model recomputations while maintaining high accuracy. We also show that our new bound is tighter than bounds in previous work [23] by recasting the mathematical proof of the bounds from prior work in simpler, geometric terms. Our reanalysis also points to a limitation on using previous bounds to infer the class predictions of the hypothetical model.

Evaluated on synthetic and real-world data sets, DRUiD provides more accurate predictions than other online learning methods. It also provides more accurate predictions than an equivalent method using previous bounds. For example, when predicting change in electricity price, DRUiD achieves 6% higher accuracy over existing work while recomputing only 20 times (roughly 0.04% of the stream length), or 2.5% higher accuracy with only 10 recomputations.

2 Related Work

We divide existing work on concept drift into algorithms which focus on accurately detecting concept changes, and incremental algorithms that implicitly adapt the learned model to the new concept.

Concept Drift Detection DDM by Gama et al. [13] monitors the classifier error rate by assuming that it decreases as the number of examples increases. If the error rate increases significantly the data is considered to have undergone concept drift. Similarly, EDDM by Baena-García et al. [1] detects concept drifts by monitoring the number of correct predictions between two consecutive classification errors.

Some adaptive algorithms also rely on sliding windows. FLORA2 by Widmer and Kubat [30] adjusts the window size to maintain model accuracy above a user-defined accuracy threshold. Harel et al. [15] use an adaptive sliding window to detect concept drifts: each window is split several times to different train and test sets, and the models built from each partition are expected to have similar accuracy. Otherwise, a concept drift is likely to have occurred. Multiple model computations make this approach unsuitable for systems with limited computational resources. Klinkenberg [18] suggests monitoring the values of three performance indicators – accuracy, recall and precision. If a concept drift is detected, the window is decreased to its minimal size, which is equal to one batch size. Klinkenberg and Joachims [17] presented an approach that selects an SVM window size such that the estimated generalization error on new examples is minimized. Such approaches require batch computation whenever the window is adjusted, or even to set its size in the first place, making them impractical when computational power is limited. ADWIN by Bifet and Gavaldà [3] adapts the window size by monitoring the difference in the mean value of the samples for every potential split of the window, and shrinking the window if this difference is too large. It is designed for one-dimensional samples and requires that the feature values be within a known range.

Our algorithm resembles concept drift detectors in that it considers both the features and the labels when detecting concept drifts. However, unlike most

concept drift detectors, which only monitor the predictions of the model, we can monitor changes to the model coefficients. As we show in our evaluation, this results in superior tradeoff of model computation and accuracy.

DILSQ by Gabel et al. [10] is a distributed sliding window algorithm that triggers model recomputation when the Euclidean distance between models is too large. Though similar to DRUiD in that respect, DILSQ focuses on reducing the network overhead using geometric monitoring techniques [28, 11], rather than trading accuracy, and is limited to least squares regression models.

Incremental Learning These algorithms can implicitly adapt to concept drifts. One such example is SGD, which applies first-order updates to the model [31]. Other incremental algorithms use second-order optimization, such as AROW by Crammer et al. [7] and NAROW by Orabona and Crammer [24]. Although not explicitly designed for concept drift detection, they may be adapted for this task. Our proposed algorithm can use any incremental learner internally when a concept drift is suspected, then use batch learning from the sliding window once enough samples from the new concept have been obtained.

3 Problem Definition and Notations

Consider a stream of data, where only some of the samples are labeled. The labeled data arrives as tuples $\{x_i, y_i\}$, while $y_i \in \{-1, 1\}$ for classification problems or $y_i \in \mathbb{R}$ for regression problems. Unlabeled samples have $y_i = null$.

We focus on sliding windows with a fixed or time-based window size. When a new labeled sample arrives, the window is updated – older samples in the window are removed and the new sample is added. We define W to be the sliding window at time t, and let $\mathcal{D}$ be the set of indices of labeled samples inside W.

Let $f(x, \beta) = x^T\beta$ be a linear function and let the loss function $\ell(\cdot, \cdot)$ be a differentiable and convex function with respect to the second argument. The model β_t^* is the optimal solution for the following optimization problem:

$$\beta_t^* = \arg\min_{\beta \in \mathbb{R}^d} C \sum_{i \in \mathcal{D}} \ell(y_i, f(x_i, \beta)) + \frac{1}{2}\|\beta\|^2, \quad C > 0. \tag{1}$$

Given an objective function of the form $a \sum_{i \in \mathcal{D}} \ell(y_i, f(x_i, \beta)) + b\|\beta\|^2$, choosing $C = \frac{a}{2b}$ will bring it to the standard form (1).

The optimization problem could be classification or regression, where for classification the linear classifier is $\hat{y} = \operatorname{sgn}(f(x, \beta_t^*))$ while for regression the linear model is $\hat{y} = f(x, \beta_t^*)$.

For simplicity, we define a compact notation for the loss function for a specific sample. Let $\ell_i := \ell(y_i, f(x_i, \beta))$ be the loss with respect to sample $\{x_i, y_i\}$. Then $\nabla\ell_i(\beta^*)$ is the gradient of ℓ_i with respect to β at the point β^*.

This work focuses on the problem of maintaining high model accuracy over a stream of data with concept drift. The naïve approach would be to compute a new optimal solution β_t^* for every window update, which is infeasible if the computational power is limited. Instead, we aim to understand when concept drift occurs and to compute a new model only then.

4 Bounding Model Differences

Consider two sliding windows, W_1 and W_2, where W_1 is the sliding window at some previous time t_1 and W_2 is the window at current time t_{current}. If the concept we are trying to model has not changed, we expect the two models computed from two windows to be similar. For example, the Euclidean distance between the models is expected to be small. When the concept has changed, the opposite is expected.

We first develop a bound that estimates the difference between the last computed model and the model based on the current sliding window, without actually computing it. In Section 4.3 we propose algorithm that uses this bound to monitor that difference: it computes a new model only when the estimated difference is large.

4.1 Bound the Distance Between Models

Let β_1 and β_2 be the models trained on the labeled samples in the windows W_1 and W_2. We define the difference between two models as the Euclidean distance between the two model vectors: $\|\beta_1^* - \beta_2^*\|$. We propose a bound for this distance that can be computed without knowing β_2^*. Monitoring this bound over the stream helps the algorithms detect changes in the concept, thus preventing unnecessary computations while maintaining accurate models.

Theorem 1. *Let P be an optimization problem over a window W with sample indices $\mathcal{D}$, of the standard form (1):*

$$P : \beta_p^* = \arg\min_{\beta \in \mathbb{R}^d} C_p \sum_{i \in \mathcal{D}} \ell_i + \frac{1}{2}\|\beta\|^2,$$

with its associated constant C_P. Let β_1^* be the optimal solution of P_1 over previous window W_1 containing the labeled samples with indices $\mathcal{D}_1$, let β_2^* be the solution of P_2 over current window W_2 containing labeled samples $\mathcal{D}_2$, let C_1 be the associated constant of P_1 and let C_2 be the associated constant of P_2. Let $\mathcal{D}_\mathcal{A}$ be the set of indices of labeled samples added in W_2: $\mathcal{D}_\mathcal{A} = \mathcal{D}_2 \setminus \mathcal{D}_1$. Similarly, let $\mathcal{D}_\mathcal{R}$ be the set of indices of samples removed in W_2: $\mathcal{D}_\mathcal{R} = \mathcal{D}_1 \setminus \mathcal{D}_2$. Finally, let Δg be

$$\Delta g := \sum_{i \in \mathcal{D}_\mathcal{A}} \nabla \ell_i(\beta_1^*) - \sum_{i \in \mathcal{D}_\mathcal{R}} \nabla \ell_i(\beta_1^*).$$

Then the distance between β_1^* and β_2^* is bounded by: $\|\beta_1^* - \beta_2^*\| \leq 2\|r\|$, where

$$r = \frac{1}{2}\left(\beta_1^* - \frac{C_2}{C_1}\beta_1^* + C_2 \Delta g\right). \tag{2}$$

Theorem 1 bounds the difference between computed models for any convex differentiable loss given the difference of their training sets. For example, we can apply Theorem 1 to L_2-regularized logistic regression, as defined in Liblinear [9],

Table 1. Objective functions, losses, and associated bound parameter r.

Model	Objective Function	Loss	r
L_2-reg LR	$\min_\beta C\sum_i \log\left(1+\exp(-y_i x_i^T\beta)\right)+\frac{1}{2}\|\beta\|^2$	$\log\left(1+\exp(-y_i x_i^T\beta)\right)$	$\frac{C}{2}\Delta g$
L_2-reg SVM	$\min_\beta C\sum_i \left(\max\{0,1-y_i x_i^T\beta\}\right)^2+\frac{1}{2}\|\beta\|^2$	$\left(\max\{0,1-y_i x_i^T\beta\}\right)^2$	$\frac{C}{2}\Delta g$
Ridge Reg.	$\min_\beta \sum_i \left(y_i - x_i^T\beta\right)^2+\alpha\|\beta\|^2$	$\left(y_i - x_i^T\beta\right)^2$	$\frac{1}{4\alpha}\Delta g$

$C_1 = C_2 = C$. Assigning this in (2) gives $r = \frac{C}{2}\Delta g$. For L_2-regularized MSE loss, the constants C_1 and C_2 depend on the number of samples in the windows, and thus may differ if the size of the windows W_1 and W_2 is different. Table 1 lists r for several important optimization problems.

Proof. The proof of Theorem 1 proceeds in three steps: (i) use the convexity of the objective function to get a sphere that contains β_2^*; (ii) use the convexity of the objective function again to express the sphere's radius as a function of Δg; and (iii) bound the distance between β_1^* and β_2^* using geometric arguments.

(i) Sphere Shape Around β_2^*: This step adapts the proof in [23] to the canonical form and simplifies it. Recall that β_2^* is the optimal solution of (1), so according to the first-order optimality condition [6], $C_2\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_2^*)+\beta_2^*=0$. This could be written as

$$\beta_2^* = -C_2\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_2^*). \tag{3}$$

ℓ_i is convex and differentiable, and therefore its gradient is monotonic non-decreasing (see Lemma 1 in [8] for the proof of this feature of convex function):

$$(\nabla\ell_i(\beta_2^*)-\nabla\ell_i(\beta_1^*))^T(\beta_2^*-\beta_1^*)\geq 0. \tag{4}$$

By summing (4) over all $i\in\mathcal{D}_2$, opening brackets and rearranging the inequality, we obtain:

$$\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_2^*)^T(\beta_2^*-\beta_1^*)\geq\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_1^*)^T(\beta_2^*-\beta_1^*). \tag{5}$$

Multiplying both sides of (5) with C_2 ($C_2>0$), and using (3), gives

$$\beta^{*T}_2(\beta_2^*-\beta_1^*)+C_2\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_1^*)^T(\beta_2^*-\beta_1^*)\leq 0. \tag{6}$$

Denote $r:=\frac{1}{2}\left(\beta_1^*+C_2\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_1^*)\right)$, and observe that we can write: $\beta_1^*-r=\frac{1}{2}\left(\beta_1^*-C_2\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_1^*)\right)$. Completing the square of (6), we have:

$$\|\beta_2^*-(\beta_1^*-r)\|^2=\underbrace{\beta^{*T}_2(\beta_2^*-\beta_1^*)+C_2\sum_{i\in\mathcal{D}_2}\nabla\ell_i(\beta_1^*)^T(\beta_2^*-\beta_1^*)}_{\leq 0,\text{ due to (6)}}+\|r\|^2.$$

Then from (6) we have $\|\beta_2^* - (\beta_1^* - r)\|^2 \leq \|r\|^2$. Denoting $m := \beta_1^* - r$, we can rewrite it as: $\beta_2^* \in \Omega$, where $\Omega := \left\{\beta \,\middle|\, \|\beta - m\|^2 \leq \|r\|^2\right\}$. Thus the new optimal solution β_2^* is within a sphere Ω with center m and radius vector r.

(ii) Express the Radius Vector as a Function of Δg: β_1^* is the optimal solution of (1). Then by the first-order optimality condition: $C_1 \sum_{i\in\mathcal{D}_1} \nabla\ell_i(\beta_1^*) + \beta_1^* = 0$. This implies $\sum_{i\in\mathcal{D}_1} \nabla\ell_i(\beta_1^*) = -\frac{\beta_1^*}{C_1}$. From the fact that $\mathcal{D}_2 = \mathcal{D}_1 + \mathcal{D}_\mathcal{A} - \mathcal{D}_\mathcal{R}$, and the definition of Δg:

$$\sum_{i\in\mathcal{D}_2} \nabla\ell_i(\beta_1^*) = \sum_{i\in\mathcal{D}_1} \nabla\ell_i(\beta_1^*) + \underbrace{\sum_{i\in\mathcal{D}_\mathcal{A}} \nabla\ell_i(\beta_1^*) - \sum_{i\in\mathcal{D}_\mathcal{R}} \nabla\ell_i(\beta_1^*)}_{\triangleq\Delta g} = -\frac{\beta_1^*}{C_1} + \Delta g.$$

Substituting this into the definition of r above, we obtain (2).

(iii) Upper Bounds to $\|\beta_1^* - \beta_2^*\|$: We observe that both β_1^* and β_2^* are inside or on the surface of the sphere Ω. For β_1^* this follows since Ω is centered at $m = \beta_1^* - r$ with radius vector r. For β_2^* this property is obtained from the definition of Ω.

This implies that the maximum distance between β_1^* and β_2^* is obtained when β_1^*, β_2^* are on the surface of the sphere at two opposite sides of the sphere's diameter, which has length $2\|r\|$, yielding the upper bound in Theorem 1. □

Improved Tightness The bound in Theorem 1 is tighter than the previous bound [23, Corollary 2] by a factor of $\sqrt{d}$, and in fact does not depend on the number of attributes d. The proof is technical and omitted for space reasons. The intuition is that [23] relies on summing d bounds on the coefficients of $\beta_1^* - \beta_2^*$.

Tikhonov Regularization We can extend our approach to Tikhonov Regularization with an invertible Tikhonov matrix A [6]. The objective function remains convex with respect to the weights, and the canonical form (1) changes to: $\beta^* = \arg\min_{\beta\in\mathbb{R}^d} C \sum_{i\in\mathcal{D}} \ell_i + \frac{1}{2}\|A\beta\|^2$. By the first-order optimality condition, $C_2 \sum_{i\in\mathcal{D}} \nabla\ell_i(\beta^*) + A^T A\beta^* = 0$. Repeating the steps for the proof of Theorem 1 starting from (3), we obtain: $r = \frac{1}{2}\left(\beta_1^* - \frac{C_2}{C_1}\beta_1^* + C_2(A^T A)^{-1}\Delta g\right)$. The only change to r is the addition of $(A^T A)^{-1}$ before Δg.

4.2 Bounding the Predictions of the New Model

To compare to previous work [23], we describe an alternative measure for the difference between two models: the difference in the prediction of the two models for a given sample. We describe upper and lower bounds for the prediction of β_2^* for a new sample. As before, we can compute these bounds without computing β_2^*, using the predictions from β_1^*.

Using the observation from Section 4.1 that β_2^* is within a sphere Ω with center m and radius vector r, we can obtain lower and upper bounds on applying β_2^* to a new sample x:

Lemma 1. *Let β_1^*, β_2^* and r be as in Theorem 1, and let x be a sample. Then the upper and lower bounds on the prediction of β_2^* for x are:*

$$L(x^T\beta_2^*) := \min_{\beta\in\Omega} x^T\beta = x^T\beta_1^* - x^T r - \|x\|\|r\| \tag{7a}$$

$$U(x^T\beta_2^*) := \max_{\beta\in\Omega} x^T\beta = x^T\beta_1^* - x^T r + \|x\|\|r\|. \tag{7b}$$

The proof follows by applying Theorem 1, then expressing β as $m+u$, where m is the center of the sphere Ω, u is parallel to x, and $\|u\| = \|r\|$. See Okumura et al. [23] for an alternative derivation of these bounds in a different form.

Lemma 1 could be used for concept drift detection in classification problems: if the upper and lower bounds (7) agree on the sign, then the classification of β_2^* is known [23]. The frequency of the disagreement between the bounds could be another indication for the quality of β_1^*; as the deviation of the current model from the older model increases due to concept drift, we expect more frequent sign disagreement as well.

However, it turns out that the bounds only agree on the class of a new sample when β_1^* and β_2^* also agree. Since both β_1^* and β_2^* are inside or on the surface of the sphere Ω, then from the definition of $L(x^T\beta_2^*)$ and $U(x^T\beta_2^*)$ we have that $L(x^T\beta_2^*) \le x^T\beta_1^*, x^T\beta_2^* \le U(x^T\beta_2^*)$. Hence, if $\operatorname{sgn}\left(x^T\beta_1^*\right) \neq \operatorname{sgn}\left(x^T\beta_2^*\right)$, then necessarily $\operatorname{sgn}\left(L(x^T\beta_2^*)\right) \neq \operatorname{sgn}\left(U(x^T\beta_2^*)\right)$.

The above implies that if the class of a sample is different under β_1^* and β_2^*, it cannot be determined from the bounds (7), and instead the bounds disagree on the sign (this limitation also applies to the bounds from [23]). Moreover, it is still possible that the bounds disagree even if the classifiers do agree on the classification. Therefore, this method for evaluating the quality of β_1^* is more sensitive to the data distribution than the bound in Section 4.1.

4.3 The DRUiD Algorithm

DRUiD is a sliding window algorithm suitable for both classification and regression problems. For every new sample that arrives, DRUiD: (a) computes β_1^*'s prediction of the new sample and updates the sliding window; (b) bounds the difference $\|\beta_1^* - \beta_2^*\|$ using Theorem 1; and (c) if the difference is too large, recomputes β_1^* from the current window. Algorithm 1 shows how DRUiD handles new samples. We describe DRUiD in detail below.

As long as the bound indicates that β_1^* and β_2^* are similar, DRUiD uses the last computed model β_1^* for prediction; however, it also maintains an incrementally updated model β_{cur}. If the bound indicates that the concept is changing, DRUiD switches to the incrementally updated model β_{cur}. Finally, once enough labeled samples from the new concept are available, DRUiD recomputes β_1^* using a full batch learning pass.

Algorithm 1 DRUiD

initialization: $\beta_{\text{cur}} \leftarrow \beta_1^*$, $n_{\mathcal{A}} \leftarrow 0$, $n_{\mathcal{R}} \leftarrow 0$, $\Delta g \leftarrow \mathbf{0}$, $numWarnings \leftarrow 0$

procedure HANDLENEWLABELEDSAMPLE($\{x_i, y_i\}$)
- Let $\{x_r, y_r\}$ be the oldest sample in the sliding window W
- Update sliding window W and count of added (removed) samples $n_{\mathcal{A}}$ ($n_{\mathcal{R}}$)
- $\Delta g \leftarrow \Delta g + \nabla \ell_i(\beta_1^*) - \nabla \ell_r(\beta_1^*)$
- $\beta_{\text{cur}} \leftarrow \text{incrementalUpdate}(\beta_{\text{cur}}, \{x_i, y_i\})$
- **if** $n_{\mathcal{A}} < N$ **then**
  - Collect $\|\Delta g\|$ for fitting
- **else**
  - **if** $n_{\mathcal{A}} = N$ **then**
    - Fit χ_d to collected $\|\Delta g\|$ and choose T_α such that $\Pr[\|\Delta g\| \leq T_\alpha] > \alpha$
  - **if** $\|\Delta g\| > T_\alpha$ **then** $numWarnings \leftarrow numWarnings + 1$
  - **else** $numWarnings \leftarrow 0$
  - **if** $numWarnings > T_N$ **then**
    - Train on window W: $\beta_1^* \leftarrow \text{batchTrain}(W)$
    - Reset window: $\beta_{\text{cur}} \leftarrow \beta_1^*$, $n_{\mathcal{A}} \leftarrow 0$, $\Delta g \leftarrow \mathbf{0}$

procedure PREDICT(x)
- **if** $numWarnings = 0$ **then return** ${\beta_1^*}^T x$
- **else return** $\beta_{\text{cur}}^T x$

DRUiD detects concept drifts by monitoring changes in $\|r\|$ from Theorem 1. When new labeled samples arrive, DRUiD updates Δg and the sliding window (since r is a linear function of Δg, monitoring changes in Δg is equivalent to monitoring changes in r); it also fits a χ_d distribution to $\|\Delta g\|$, where degrees of freedom d is the number of attributes of the data. Once enough new samples have arrived to accurately estimate the distribution parameters of $\|\Delta g\|$, DRUiD tests for potential concept drifts (we denote this constant N and set it to the window size in our evaluation). We use a simple one-tailed test[1]: a potential concept drift occurs whenever $\|\Delta g\|$ is above a user-determined α percentile of the fitted χ_d distribution, denoted as T_α: $\Pr[\|\Delta g\| \leq T_\alpha] > \alpha$.

Even when a potential concept drift is detected, DRUiD does not immediately recompute the model. Instead, it waits until $\|\Delta g\|$ is above the threshold T_α for T_N times in a row (we set T_N to the window size). This not only guarantees the batch learner a sufficiently large sample from the new concept, but also helps reduce false positives caused by outliers. To maintain high accuracy while collecting enough samples for batch learning, DRUiD switches to using β_{cur} for predictions instead of β_1^*. The model β_{cur} is initialized to β_1^* after batch recomputation and is incrementally updated for each new labeled sample, for example using an SGD update step [31]. However, it is only used after the first potential concept drift is detected, and only until enough samples are collected.

[1] We caution against ascribing such tests too much meaning. If values of Δg are i.i.d. Gaussians, then $\|\Delta g\| \sim \chi_d$. However, as with similar tests in the literature, in practice the elements are seldom i.i.d. Gaussians and even successive Δg are often not independent. Our evaluation in Section 5 explores a range of thresholds.

5 Evaluation

We evaluate DRUiD on real-world and synthetic datasets.

We consider batch model computation as a heavy operation which requires waking up the stronger processor (in heterogeneous architectures) or communication with a remote server (in connected devices). An effective edge-computing algorithm is able to tradeoff a small number of model computations for additional accuracy. To provide a point of comparison to other concept drift detection algorithms, we also consider drift detection events as model computation.

We use *tradeoff curves* to evaluate performance and compare algorithms. For every configuration of algorithm parameters, we plot a point with the resulting accuracy as the Y coordinate and the number of computations as the X coordinate. This builds a curve that shows how the algorithm behaves as we change its parameters. Practitioners can then choose a suitable operating point based on how many batch model computations they are willing to accept.

5.1 Experimental Setup

We compare DRUiD to several baseline algorithms. Since we are interested in linear models, the baselines were chosen accordingly.

- **Sliding Window** is a non-adaptive, periodic sliding window algorithm, as in the original FLORA [30]. A period parameter determines how often batch model recomputation is performed: every labeled example, every two labeled examples, and so on. The implementation of the algorithm uses Liblinear [9] logistic regression with L_2 regularization.
- **Incremental SGD** uses an SGD-based [31] first-order method update to the model. We use the SGDClassifier implementation in sklearn [25], with logistic regression loss. A full batch model is computed only once to obtain the initial model.
- **DDM** by Gama et al. [13] and **EDDM** by Baena-García et al. [1] are two popular concept drift detectors that can use batch mode or incremental base learners. They monitor the base learner accuracy and decide when to update models. We implemented batch and incremental modes for both algorithms using the Tornado framework [26].
- **PredSign**: to compare to existing bounds on model predictions [23], we describe an algorithm that uses the signs of the classification bounds from Section 4.2 to decide when a model should be recomputed. As with DRUiD, we update Δg and the sliding window when labeled samples arrive. Unlabeled sample are first evaluated using the bounds U and L. When the bounds U and L have different sign, this could indicate a concept drift. Once the number of samples for which the U and L bounds disagree on the sign exceeds a user-defined threshold T_D, PredSign recomputes the model β_1^*.
- **ADWIN** [3] is a concept drift detector for batch or incremental base learners. We used the implementation in scikit-multiflow [21]: every time a concept drift is detected, we compute a new model from the samples in the adap-

tive window. ADWIN's performance across all experiments was equivalent or inferior to DDM's and EDDM's, and is therefore not included in the figures.

We mainly focus on L_2 regularized logistic regression (Table 1). The fraction of labeled examples is set to 10%: every 10^{th} sample of the stream is considered labeled while the other samples are treated as unlabeled (their labels are only used for evaluation, not training). For the labeled examples, we use prequential evaluation: we first use the samples to test the model and only then to train it [14]. We set a window size of 2000 samples (i.e., 200 labels per window), and use the first 2000 samples from the stream to tune the learning rates and regularization parameters. Results using different window sizes were fairly similar, but this size resulted in best performance for the EDDM and DDM baselines.

The tradeoff curve for each algorithm is created by running it on the data with different parameters. For DDM, we set drift levels $\alpha \in [0.01, 30]$. The warning level β was set according to the drift level – if $\alpha > 1$, then $\beta = \alpha - 1$; otherwise $\beta = \alpha$. For EDDM, we set warning levels $\alpha \in [0.1, 0.99999]$. The drift level β was set according to the warning level – if the $\alpha > 0.05$, then $\beta = \alpha - 0.05$; otherwise $\beta = \alpha$. We ran PredSign with threshold $T_D \in [60, 50000]$. For DRUiD, we set α values $\in [0.01, 0.9999999]$. For ADWIN, we set δ values $\in [0.0001, 0.9999999]$. Finally, the period parameter for Sliding Window was set between 60 to 30000.

5.2 Electricity Pricing Dataset

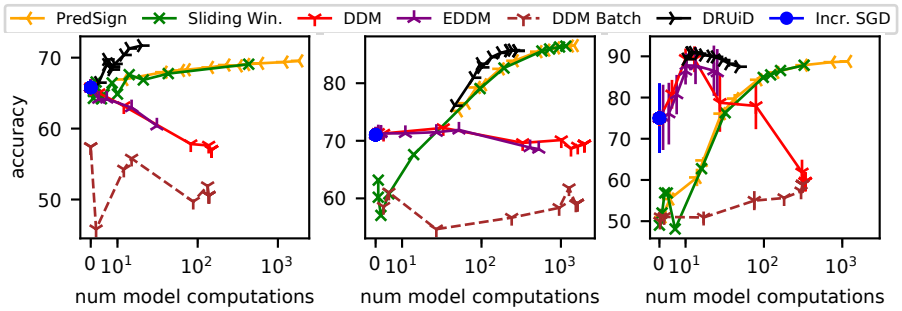
The **ELEC2** dataset described by Harris et al. [16] contains 45,312 instances from Australian New South Wales Electricity, using 8 input attributes recorded every half an hour for two years . The classification task is to predict a rise ($y_i = +1$) or a fall ($y_i = -1$) in the price of electricity. We used the commonly available MOA [4] version of the dataset without the date, day, and time attributes.

Figure 1a compares the performance of the different algorithms on the ELEC2 dataset. Every point in the graph represents one run of an algorithm on the entire stream with a specific value of the algorithm's meta-parameter – the closer to the top-left corner, the better. Connecting the points creates a curve that describes the tradeoff between computation and accuracy.

Overall, the accuracy of sliding window algorithms that use batch learning (Sliding Window, PredSign and DRUiD) is superior to that of the incremental learning algorithms (Incremental SGD, DDM and EDDM).

DRUiD gives the best tradeoff of model computations to accuracy: at every point, it offers the highest accuracy with the fewest model computations. For example, DRUiD achieves 70% accuracy using 10 batch model computations throughout the entire stream (0.02% of stream size), while Sliding Window and PredSign need two orders of magnitude more computation to reach similar accuracy. DDM, EDDM and Incremental SGD are unable to achieve such accuracy on this dataset, despite careful tuning efforts.

Though PredSign is able to match the performance of the sliding window algorithm, DRUiD offers a superior computation-accuracy tradeoff. Bounding the model difference (Section 4.1) results in fewer false concept drift detections than the approach in [23], which bounds the model prediction (Section 4.2).



(a) ELEC2. (b) Cover type. (c) Sine1.

Fig. 1. The tradeoff between accuracy and number of model computations in ELEC2, Forest Covertype, and Sine1 datasets, for different parameter configurations of each algorithm (EDDM Batch performance is similar to DDM Batch). Vertical lines in Sine1 show standard deviation over 5 experiments. The optimal number of model computations is 10, since this dataset has 10 concepts. In all cases DRUiD achieves a better tradeoff, showing equal or superior accuracy at lower computational cost than all other algorithms across a large range of configurations.

Surprisingly, the accuracy of DDM and EDDM drops even when we use more model computations. Digging deeper, we saw that DDM and EDDM switch too soon to a new model without sufficient training. A thorough parameter sweep using a fine grid, including the parameter values recommended by the authors, shows that for most configurations DDM and EDDM do not detect any concept drifts in this data and simply use the initial model from the first window – it is the optimal point for these algorithms on this dataset. The few configurations that cause DDM and EDDM to detect drifts end up performing poorly as they switch to a new model too soon, without sufficient training samples.

DDM and EDDM perform poorly in batch mode on all tested datasets, since across all test configurations they yield few samples between the warning and drift threshold, resulting in low accuracy models built from few samples[2].

5.3 Forest Covertype

The **Forest Covertype** dataset contains the forest cover type for 30x30 meter cells obtained from US Forest Service data [5]. It includes 581,012 samples with 54 attributes each that are divided into 7 classes. To convert the problem to a binary classification problem, we set the label y_i to $+1$ for class number 2, the Lodgepole Pine cover type, and -1 for the rest of the classes (this results in near equal number of positive and negative examples).

Figure 1b shows the computation-accuracy tradeoff for this dataset. Sliding window algorithms give more accurate results than the incremental algorithms,

[2] As far as we can tell, this is consistent with practice: the implementations of DDM and EDDM that we found use incremental base learners [26, 22, 4].

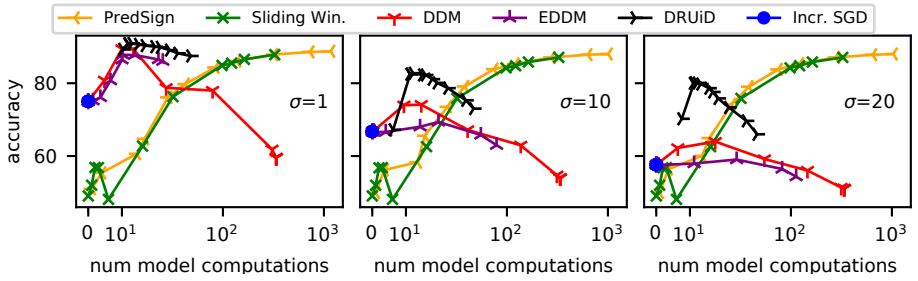


Fig. 2. Tradeoff curves for the Sine1+ dataset with different scale values (σ). The accuracy on the incremental based algorithms drops where the scale is larger.

except for the extreme case where the number of model computations is close to zero. DRUiD shows the best accuracy with the least model computations.

5.4 Sine1+

The **Sine1+** artificial dataset is based on the Sine1 artificial dataset presented in [13, 1], but extended to more than 2 attributes and to allow non-uniform scales. It contains 9 abrupt concept drifts, each with 10,000 samples (hence the optimal number of model computations is 10). The dataset has $d \geq 2$ attributes: x_1 is uniformly distributed in $[0, \sigma]$, where $\sigma \geq 1$ sets its scale compared to other attributes $x_2, ..., x_d$ which are uniformly distributed in $[0, 1]$. In the first concept, points that lie below the curve $x_d = sin\left(\frac{(x_1/\sigma)+\sum_{i=2}^{d-1} x_i}{d-1}\right)$ are classified as $+1$ and the rest are classified as -1. After the concept drift, the classification is reversed. Note that for $d = 2, \sigma = 1$ we get the original Sine1 dataset, with the separating line $x_2 = sin(x_1)$.

Figure 1c shows the computation-accuracy tradeoff for the original Sine1 dataset ($d = 2$ and $\sigma = 1$). Every point in the graph is the average of 5 runs with different random seeds (for every seed, all algorithms see the same data), and the vertical lines are the standard deviation error bars.

DRUiD detects all concept drifts even when set to very low sensitivity levels (high α), so the number of model computations does not go below 10. It maintains high accuracy and a small number of model computations for all α values. For almost every number of model computations, PredSign accuracy is higher than Sliding Window. This is because PredSign can change the timing of its model computations, which is not possible in Sliding Window.

Figure 2 shows the computation-accuracy tradeoff for $d = 2$ and different σ values. As we explain below, DRUiD's batch mode computation maintains high accuracy even as the problem becomes increasingly ill-conditioned.

Conversely, incremental algorithms are sensitive to non-uniform attribute scales. Figure 3 shows the effect of scale on convergence in the Sine1+ dataset. The top figures show the accuracy over time of Incremental SGD, DDM, and DRUiD between two consecutive concept drifts (EDDM behaves similarly to

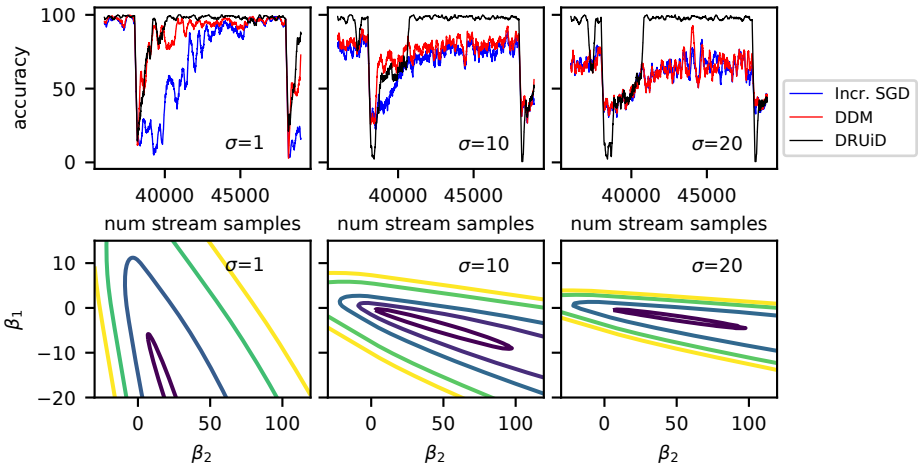


Fig. 3. Effect of different σ values on Sine1+. Top: accuracy over time between two concept drifts. Bottom: the contour lines of the L_2-regularized logistic regression objective functions. The optimization problem becomes increasingly ill-conditioned when the scale σ increases, so Incremental SGD recover more slowly after a concept drift.

DDM). As σ increases, the convergence time of the incremental based algorithms also grows, as expected [6, p. 467]. The bottom figures show the contour lines of the L_2-regularized logistic regression objective functions for different σ values (the loss surface). As σ increases, the shape of the loss surface becomes more elliptic, with a higher condition number and slower convergence for gradient descent methods. For higher condition number, the batch based algorithms require more iterations of gradient descent to converge on recomputation of a new model. However, this recomputation is performed on the strong processor or cloud server, with no effect on the accuracy. The effect of σ does not depend on the number of attributes. Using $d = 50$ attributes yields similar results to only 2: larger σ values reduce the accuracy of incremental algorithms, even though σ only affects x_1 (figures omitted due to lack of space).

Feature normalization or other preconditioning is not always possible in streams that have concept drifts, since the distribution of the attributes is not known and can change unexpectedly. Algorithms that use batch learning are better suited for ill-conditioned streams than relying solely on incremental learning.

5.5 Ridge Regression

We evaluate DRUiD performance on an artificial regression task, and compare it to Incremental SGD and Sliding Window (PredSign, DDM, and EDDM only support classification). We generate 10 concepts, each with a different true model β_{true} with 2 coefficients drawn from a standard normal distribution. For each epoch concept we generate 10,000 samples, where each sample x has 2 attributes drawn from a standard normal distribution. As with Sine1+, one of the attributes

is then expanded by a factor of $\sigma \geq 1$. Each label is $y = x^T \beta_{\text{true}} + \epsilon$, where ϵ is random Gaussian noise: $\epsilon \sim N(0, 1)$.

As in the Sine1+ dataset, non-uniform scaling increases the condition number of the problem. The computation and accuracy tradeoff follow similar trends as in classification (figures omitted due to lack of space). For a well-conditioned problem, the incremental algorithm and DRUiD achieve the same RMSE as the best periodic algorithm (albeit with far fewer model computations). However, when the condition number increases, DRUiD achieves a better tradeoff than the incremental algorithm, since the batch learning convergence rate does not affect the accuracy along the stream.

6 Conclusions

DRUiD is an online algorithm for data streams with concept drifts, designed for edge computing systems. It improves accuracy with minimal cost by running batch computations of new data only when the model changes. DRUiD relies on an improved bound for the difference between two linear models with convex differentiable loss. Evaluation on real and synthetic data shows that DRUiD provides a better tradeoff between model computation and accuracy than traditional concept drift detectors, and that its batch-based computation is better suited for ill-conditioned problems than methods based on incremental learning.

References

1. Baena-García, M., Campo-Ávila, J., Fidalgo-Merino, R., Bifet, A., Gavald, R., Morales-Bueno, R.: Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams (2006)
2. B.B, P.R., Saluja, P., Sharma, N., Mittal, A., Sharma, S.: Cloud computing for internet of things & sensing based applications. ICST (2012)
3. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. SIAM '07 (2007)
4. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. JMLR **11**, 1601–1604 (2010)
5. Blackard, J.A., Dean, D.J.: Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Computers and Electronics in Agriculture **24**, 131–151 (1999)
6. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
7. Crammer, K., Kulesza, A., Dredze, M.: Adaptive regularization of weight vectors. Mach. Learn. **91**(2), 155–187 (05 2013)
8. Dunn, J.: Convexity, monotonicity, and gradient processes in hilbert space. J. Math. Anal. Appl. **53**, 145–158 (01 1976)
9. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. J. Mach. Learn. Res. **9**, 1871–1874 (06 2008)
10. Gabel, M., Keren, D., Schuster, A.: Monitoring least squares models of distributed streams. KDD '15 (2015)

11. Gabel, M., Keren, D., Schuster, A.: Anarchists, unite: Practical entropy approximation for distributed streams. KDD '17 (2017)
12. Gama, J.: A survey on learning from data streams: current and future trends. Progress in Artificial Intelligence **1**(1), 45–55 (04 2012)
13. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Brazilian Symposium on Artificial Intelligence (2004)
14. Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Mach. Learn. **90**(3), 317–346 (Mar 2013)
15. Harel, M., Crammer, K., El-Yaniv, R., Mannor, S.: Concept drift detection through resampling. ICML '14 (2014)
16. Harries, M., South Wales, N.: Splice-2 comparative evaluation: Electricity pricing (08 1999)
17. Klinkenberg, R., Joachims, T.: Detecting concept drift with support vector machines. ICML '00 (2000)
18. Klinkenberg, R., Renz, I.: Adaptive information filtering: Learning drifting concepts. FGML-98 (1998)
19. Liaqat, D., Jingoi, S., de Lara, E., Goel, A., To, W., Lee, K., De Moraes Garcia, I., Saldana, M.: Sidewinder: An energy efficient and developer friendly heterogeneous architecture for continuous mobile sensing. ASPLOS '16 (2016)
20. Mahdavinejad, M.S., Rezvan, M., Barekatain, M., Adibi, P., Barnaghi, P., Sheth, A.P.: Machine learning for internet of things data analysis: a survey. Digital Communications and Networks **4**(3), 161 – 175 (2018)
21. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: A multi-output streaming framework. JMLR **19**(72), 1–5 (2018)
22. Nishida, K.: Learning and Detecting Concept Drift. Ph.D. thesis, Hokkaido University (2008)
23. Okumura, S., Suzuki, Y., Takeuchi, I.: Quick sensitivity analysis for incremental data modification and its application to leave-one-out CV in linear classification problems. KDD '15 (2015)
24. Orabona, F., Crammer, K.: New adaptive algorithms for online classification. NIPS '10 (2010)
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. JMLR **12**, 2825–2830 (2011)
26. Pesaranghader, A., Viktor, H.L., Paquet, E.: A framework for classification in data streams using multi-strategy learning. In: Calders, T., Ceci, M., Malerba, D. (eds.) Discovery Science (2016)
27. Read, J.: Concept-drifting data streams are time series; the case for continuous adaptation. CoRR **abs/1810.02266** (2018)
28. Sharfman, I., Schuster, A., Keren, D.: Shape sensitive geometric monitoring. PODS '08 (2008)
29. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal **3**(5), 637–646 (10 2016)
30. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**(1), 69–101 (4 1996)
31. Xu, W.: Towards optimal one pass large scale learning with averaged stochastic gradient descent. CoRR **abs/1107.2490** (2011)
32. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. ICML '03 (2003)