# DriftSurf: A Risk-competitive Learning Algorithm under Concept Drift

**Ashraf Tahmasbi** [*1] **Ellango Jothimurugesan** [*2] **Srikanta Tirthapura** [1 3] **Phillip B. Gibbons** [2]

## Abstract

When learning from streaming data, a change in the data distribution, also known as concept drift, can render a previously-learned model inaccurate and require training a new model. We present an adaptive learning algorithm that extends previous drift-detection-based methods by incorporating drift detection into a broader stable-state/reactive-state process. The advantage of our approach is that we can use aggressive drift detection in the stable state to achieve a high detection rate, but mitigate the false positive rate of standalone drift detection via a reactive state that reacts quickly to true drifts while eliminating most false positives. The algorithm is generic in its base learner and can be applied across a variety of supervised learning problems. Our theoretical analysis shows that the risk of the algorithm is competitive to an algorithm with oracle knowledge of when (abrupt) drifts occur. Experiments on synthetic and real datasets with concept drifts confirm our theoretical analysis.

## 1. Introduction

Learning from streaming data is an ongoing process in which a model is continuously updated as new training data arrive. We focus on the problem of concept drift, which refers to an unexpected change in the distribution of data over time. The objective is high prediction accuracy at each time step on test data from the current distribution. To achieve this goal, a learning algorithm should adapt quickly whenever drift occurs by focusing on the *most recent data points* that represent the new concept, while also, in the absence of drift, optimizing over *all the past data points* from the current distribution (for statistical accuracy). The latter has greater importance in the setting we consider where data points may be stored and revisited to achieve accuracy greater than what can be obtained in a single pass. Moreover, computational efficiency of the learning algorithm is critical to keep pace with the continuous arrival of new data.

In a survey from Gama et al. (Gama et al., 2014), concept drift between time steps $t_0$ and $t_1$ is defined as a change in the joint distribution of examples: $p_{t_0}(X, y) \neq p_{t_1}(X, y)$. Gama et al. categorize drifts in several ways, distinguishing between *real drift* that is a change in $p(y|X)$ and *virtual drift* (also known as *covariate drift*) that is a change only in $p(X)$ but not $p(y|X)$. Drift is also categorized as either *abrupt* when the change happens across one time step, or *gradual* if there is a transition period between the two concepts.

A learning algorithm that reacts (well) to concept drift is referred to as an *adaptive algorithm*. In contrast, an *oblivious algorithm*, which optimizes the empirical risk over all data points observed so far under the assumption that the data are i.i.d., performs poorly in the presence of drift. One major class of adaptive algorithms is drift detection, which includes DDM (Gama et al., 2004), EDDM (Baena-García et al., 2006), ADWIN (Bifet & Gavaldà, 2007), PERM (Harel et al., 2014), FHDDM (Pesaranghader & Viktor, 2016), and MDDM (Pesaranghader et al., 2018). Drift detection tests commonly work by tracking the prediction accuracy of a model over time, and signal that a drift has occurred whenever the accuracy degrades by more than a significant threshold. After a drift is signaled, the previously-learned model can be discarded and replaced with a model trained solely on the data going forward.

There are a couple of challenges with using drift detection. Different tests are preferred depending on whether a drift is abrupt or gradual, and most drift detection tests have a user-defined parameter that governs a trade-off between the detection accuracy and speed (Gama et al., 2014); choosing the right test and the right parameters is hard when the types of drift that will occur are not known in advance. There is also a significant cost in prediction accuracy when a false positive results in the loss of a long-trained model and data that are still relevant. Furthermore, even when drift is accurately detected, not all drifts require restarting with a new model. Drift detection can trigger following a virtual drift when the model misclassifies data points drawn from a previously unobserved region of the feature space, but the

[*]Equal contribution [1]Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, USA [2]Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA [3]Apple, Cupertino, California, USA. Correspondence to: Ashraf Tahmasbi <tahmasbi@iastate.edu>, Ellango Jothimurugesan <ejothimu@cs.cmu.edu>.

older data still have valid labels and should be retained. We have also encountered real drifts in our experimental study where a model with high parameter dimension can adapt to simultaneously fit data from both the old and new concepts, and it is more efficient to continue updating the original model rather than starting from scratch.

Our contribution is DriftSurf, an adaptive algorithm that overcomes these challenges with drift detection. DriftSurf works by maintaining two models at each time step and incorporating drift detection into a broader two-state process. The algorithm begins in the *stable state* and transitions to the *reactive state* based on a drift detection trigger, and then starts a new model. During the reactive state, the model used for prediction is greedily chosen as the best performer over data from the immediate previous time step (each time step corresponds to a batch of arriving data points). At the conclusion of the reactive state, the algorithm transitions back to the stable state, keeping the model that was the best performer throughout the entire reactive state. Our approach has several advantages over standalone drift detection: (i) most false positives will be caught by the reactive state and lead to continued use of the original long-trained model and all the relevant past data; (ii) when restarting with a new model does not lead to better post-drift performance, the original model will continue to be used; and (iii) switching to the new model for predictions happens only when it begins outperforming the old model, accounting for the potentially lower accuracy of the new model as it warms up. Meanwhile, the addition of this stable-state/reactive-state process does not come at a significant cost in the recovery time, because the switch to a new model happens greedily within one time step of it outperforming the old model (as opposed to switching only at the end of the reactive state).

We present a theoretical analysis of DriftSurf, showing that it is "risk-competitive" with Aware, an adaptive algorithm defined in Section 5 that has oracle access to when a drift occurs and at each time step maintains a model trained over the set of all data since the previous drift. We also provide experimental comparisons of DriftSurf to Aware and two adaptive learning algorithms: a state-of-the-art drift-detection-based method MDDM and a state-of-the-art ensemble method AUE (Brzezinski & Stefanowski, 2013). Our results on eight datasets with concept drifts show that DriftSurf generally outperforms both MDDM and AUE.

## 2. Related Work

Most adaptive learning algorithms can be classified into three major categories: Window-based, drift detection, and ensembles. Window-based methods, which include the family of FLORA algorithms (Widmer & Kubat, 1996) train models over a sliding window of the recent data in the stream. Alternatively, older data can be forgotten gradually by weighting the data points according to their age with either linear (Koychev, 2000) or exponential (Klinkenberg, 2004; Hentschel et al., 2019) decay. Window-based methods are guaranteed to adapt to drifts, but at a cost in accuracy in the absence of drift.

The aforementioned drift detection methods can be further classified as either detecting degradation in prediction accuracy with respect to a given model, which include all of the tests mentioned in Section 1, or detecting change in the underlying data distribution which include tests given by (Kifer et al., 2004; Sebastião & Gama, 2007). In this paper, we focus on the subset of concept drifts that are performance-degrading, and that can be detected by the first class of these drift detection methods. As observed in (Harel et al., 2014), under this narrower focus, the problem of drift detection has lower sample and computational complexity when the feature space is high-dimensional. Furthermore, this approach ignores drifts that do not require adaptation, such as changes only in features that are weakly correlated with the label.

Finally, there are ensemble methods, such as DWM (Kolter & Maloof, 2007), Learn++.NSE (Elwell & Polikar, 2011), AUE (Brzezinski & Stefanowski, 2013), DWMIL (Lu et al., 2017), and DTEL (Sun et al., 2018). An ensemble is a collection of individual models, often referred to as experts, that differ in the subset of the stream they are trained over. Ensembles adapt to drift by including both older experts that perform best in the absence of drift and newer experts that perform best in the presence of drift. The predictions of each individual expert are typically combined using a weighted vote, where the weights depend on each expert's recent prediction accuracy. Strictly speaking, DriftSurf is an ensemble method, but differs from traditional ensembles by maintaining only two models and where only one model is used to make a prediction at any time step. The advantage of DriftSurf is its efficiency, as the maintenance of each additional model in an ensemble comes at either a cost in additional training time, or at a cost in the accuracy of each individual model if the available training time is divided among them. The ensemble algorithm most similar to ours is from (Bach & Maloof, 2008), which also maintains just two models: a long-lived model that is best-suited in the stationary case, and a newer model trained over a sliding window that is best-suited in the case of drift. Their algorithm differs from DriftSurf in that instead of using a drift detection test to switch, they are essentially always in what we call the reactive state of our algorithm, where they choose to switch to a new model whenever its performance is better over a window of recent data points. Their algorithm has no theoretical guarantee, and without the stable-state/reactive-state process of our algorithm, there is no control over false switching to the newer model in the stationary case.

## 3. Model and Preliminaries

We consider a data stream setting in which the training data points arrive over time. For $t = 0, 1, 2, \ldots$, let $\mathbf{X}_t$ be the set of data points arriving at time step $t$. We consider a constant arrival rate $m = |\mathbf{X}_t|$ for all $t$. (Our discussion and results can be readily extended to Poisson and other arrival distributions.) Let $\mathcal{S}_{t_1,t_2} = \cup_{t=t_1}^{t_2-1} \mathbf{X}_t$ be a segment of the stream of points arriving in time steps $t_1$ through $t_2 - 1$. Let $n_{t_1,t_2} = m(t_2 - t_1)$ be the number of data points in $\mathcal{S}_{t_1,t_2}$.

Each $\mathbf{X}_t$ consists of data points drawn from a distribution $I_t$ not known to the learning algorithm. In the *stationary* case, $I_t = I_{t-1}$; otherwise, a *concept drift* has occurred at time $t$. We seek an adaptive learning algorithm with high prediction accuracy at each time step.

The model being trained is drawn from a class of functions $\mathcal{F}$. A function in this class is parameterized by a vector of weights $\mathbf{w} \in \mathbb{R}^d$. In a data stream setting, given a stream segment $\mathcal{S}_{t_1,t_2}$ of training data points, the best we can do when the data are all drawn from the same distribution is to minimize the empirical risk over this stream segment. The *empirical risk* of function $\mathbf{w}$ over a sample $\mathcal{S}$ of $n$ elements is: $\mathcal{R}_{\mathcal{S}}(\mathbf{w}) = \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{S}} f_{\mathbf{x}}(\mathbf{w})$, where $f_{\mathbf{x}}(\mathbf{w})$ is the loss of function $\mathbf{w}$ on input $\mathbf{x}$. The optimizer of the empirical risk is denoted as $\mathbf{w}^*_{\mathcal{S}}$, defined as $\mathbf{w}^*_{\mathcal{S}} = \arg\min_{\mathbf{w} \in \mathcal{F}} \mathcal{R}_{\mathcal{S}}(\mathbf{w})$. The optimal empirical risk is $\mathcal{R}^*_{\mathcal{S}} = \mathcal{R}_{\mathcal{S}}(\mathbf{w}^*_{\mathcal{S}})$.

Let $\mathbf{w}$ be the solution learned by an algorithm $A$ over stream segment $\mathcal{S} = \mathcal{S}_{t_1,t_2}$. Following prior work (Bousquet & Bottou, 2007; Jothimurugesan et al., 2018), we define the difference between $A$'s empirical risk and the optimal empirical risk over this stream segment as its sub-optimality: $\texttt{SUBOPT}_{\mathcal{S}}(A) := \mathcal{R}_{\mathcal{S}}(\mathbf{w}) - \mathcal{R}_{\mathcal{S}}(\mathbf{w}^*_{\mathcal{S}})$. Based on (Bousquet & Bottou, 2007), achieving a sub-optimality on the order of $\mathcal{H}(n_{t_1,t_2})$ over stream segment $\mathcal{S}_{t_1,t_2}$ asymptotically minimizes the total (statistical + optimization) error for $\mathcal{F}$, where $\mathcal{H}(n) = hn^{-\alpha}$, for a constant $h$ and $1/2 \leq \alpha \leq 1$. Note that this is true when all data points are drawn from the same distribution.

However, suppose a concept drift occurs at time $t_d$ such that $t_1 < t_d < t_2$ and data points in $\mathcal{S}_{t_1,t_d}$ and $\mathcal{S}_{t_d,t_2}$ are drawn from distributions $I_1$ and $I_2$, respectively. We could still define empirical risk and sub-optimality of an algorithm $A$ over stream segment $\mathcal{S}_{t_1,t_2}$. But, balancing sub-optimality with $\mathcal{H}(n_{t_1,t_2})$ does not necessarily minimize the total error. Algorithm $A$ needs to first recover from the drift such that the predictive model is trained only over data points drawn from the new distribution. We define recovery time as follows:

**Definition 1.** *The **recovery time** of an algorithm $A$ is the time it takes after a drift for this algorithm to provide a solution $\mathbf{w}$ which is maintained solely over data points drawn from the new distribution.*

Let $t_{d_1}, t_{d_2}, \ldots$ be the sequence of time steps at which a drift occurs, and define $t_{d_0} = 0$. The goals for an adaptive learning algorithm $A$ are **(G1)** to have a small recovery time $r_i$ at each $t_{d_i}$ and **(G2)** to achieve sub-optimality on the order of $\mathcal{H}(n_{t_{d_i},t})$ over every stream segment $\mathcal{S}_{t_{d_i},t}$ for $t_{d_i} + r_i < t < t_{d_{i+1}}$ (i.e., during the stationary periods between drifts). In Section 5, we formalize the latter as $A$ being "risk-competitive" with an oracle algorithm Aware. It implies that $A$ is asymptotically optimal in terms of its total error, despite concept drifts.

## 4. DriftSurf: Adaptive Learning over Streaming Data in Presence of Drift

We present our algorithm DriftSurf for adaptively learning from streaming data that may experience drift. Incremental learning algorithms work by repeatedly sampling a data point from a training set $\mathcal{S}$ and using its gradient to determine an update direction. This set $\mathcal{S}$ expands as new data points arrive. In the presence of a drift from distribution $I_1$ to $I_2$, without a strategy to remove from $\mathcal{S}$ data points from $I_1$, the model trains over a mixture of data points from $I_1$ and $I_2$, often resulting in poor prediction accuracy on $I_2$. One systematic approach to mitigating this problem would be to use a sliding window-based set $\mathcal{S}$ from which further sampling is conducted. Old data points are removed when they fall out of the sliding window (regardless of whether they are from the current or an old distribution). However, the problem with this approach is that the corresponding sub-optimality of the model trained over $\mathcal{S}$ is bounded by the (limited) size of $\mathcal{S}$. Larger window sizes help with achieving a better sub-optimality, but will increase the recovery time. On the other hand, smaller window sizes provide better recovery time, but the corresponding sub-optimality of the algorithm over $\mathcal{S}$ increases. An ideal algorithm manages the set $\mathcal{S}$ such that it contains as many as possible data points from the current distribution and resets it whenever a (significant) drift happens, so that it contains only data points from the new distribution.

As noted in Section 1, prior work (Gama et al., 2004; Baena-García et al., 2006; Bifet & Gavaldà, 2007; Harel et al., 2014; Pesaranghader & Viktor, 2016; Pesaranghader et al., 2018) has sought to achieve this ideal algorithm by developing better and better drift detection tests, but with limited success due to the challenges of balancing detection accuracy and speed, and the high cost of false positives. Instead, we couple aggressive drift detection with a stable-state/reactive-state process that mitigates the shortcomings of prior approaches. Unlike prior drift detection approaches, DriftSurf views performance degrading as only a sign of a potential drift: the final decision about resetting $\mathcal{S}$ and the predictive model will not be made until the end of the reactive state, when more evidence has been gathered and a

**Algorithm 1** DriftSurf: Processing a set of training points $\mathbf{X}_t$ that arrives in time step $t, t > 0$

// $\mathbf{w}_{t-1}$, $\mathbf{w}'_{t-1}$, and $\mathbf{w}''_{t-1}$ are (respectively) the parameters of the current predictive model, the reactive model (if in the reactive state), and the stable model (if in the stable state)
// $\mathcal{S}$, $\mathcal{S}'$, and $\mathcal{S}''$ are (respectively) the sample sets (stream segments) used for training the predictive model, the reactive model (if in the reactive state), and the stable model (if in the stable state)
// $r$ is the length of the reactive state
**if** state.stable **then**
  **if** Enter Reactive State **then** {condition 1 or 3 holds}
    Set state to reactive
    $T \leftarrow \emptyset$ {segment arrived during reactive state}
    $\mathbf{w}'_{t-1} \leftarrow \mathbf{w}_0$, $\mathcal{S}' \leftarrow \emptyset$ {initialize randomly}
    $\mathbf{w}^f \leftarrow \mathbf{w}_{t-1}$ {frozen parameters of the predictive model}
    $c \leftarrow 0$ {time steps in the current reactive state}
  **else**
    $\mathbf{w}_t \leftarrow$ Update$(\mathbf{w}_{t-1}, \mathcal{S}, \mathbf{X}_t)$
    $\mathbf{w}''_t \leftarrow$ Update$(\mathbf{w}''_{t-1}, \mathcal{S}'', \mathbf{X}_t)$
  **end if**
**end if**
**if** state.reactive **then**
  Add $\mathbf{X}_t$ to $T$
  $\mathbf{w}_t \leftarrow$ Update$(\mathbf{w}_{t-1}, \mathcal{S}, \mathbf{X}_t)$
  $\mathbf{w}'_t \leftarrow$ Update$(\mathbf{w}'_{t-1}, \mathcal{S}', \mathbf{X}_t)$
  $c \leftarrow c + 1$
  **if** Exit Reactive State **then** $\{c == r\}$
    Set state to stable
    $\mathbf{w}''_{t-1} \leftarrow \mathbf{w}_0$, $\mathcal{S}'' \leftarrow \emptyset$ {initialize randomly}
    **if** $\mathcal{R}_T(\mathbf{w}^f) > \mathcal{R}_T(\mathbf{w}'_t)$ **then** {condition 2 holds}
      $\mathbf{w}_t \leftarrow \mathbf{w}'_t$, $\mathcal{S} \leftarrow \mathcal{S}'$ {change the predictive model}
    **end if**
  **else if** $\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}'_{t-1}) < \mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1})$ **then**
    Use $\mathbf{w}'_t$ instead of $\mathbf{w}_t$ for predictions only at the next time step {greedy policy during the reactive state}
  **end if**
**end if**

higher confidence decision can be made.

Our algorithm, DriftSurf, is depicted in Algorithm 1. The algorithm starts in the stable state, and the steps are shown for processing the batch of points arriving at time step $t$. If the algorithm is in the stable state, DriftSurf checks for a sign of drift to decide if it needs to enter the reactive state. This decision is made by checking the following condition:

$$\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) > \mathcal{R}_b + \delta \tag{1}$$

where $\mathbf{w}_{t-1}$ is the parameters of the current predictive model, $\mathcal{R}_b$ is the best observed risk of this model and $\delta$ is a predetermined threshold that represents the tolerance in performance degradation.

**Algorithm 2** Update$(\mathbf{w}, \mathcal{S}, \mathbf{X}_t)$: Process of updating parameters $\mathbf{w}$ using SGD, given sample set $\mathcal{S}$ and newly arrived data points $\mathbf{X}_t$

// $\rho$ is the computational power and determines the number of update steps that can be performed
// $\eta$ is the learning rate
Add $\mathbf{X}_t$ to $\mathcal{S}$
**for** $j = 1$ to $\rho$ **do**
  Sample a point $p$ uniformly from $\mathcal{S}$
  $g \leftarrow \nabla f_p(\mathbf{w})$ {$f_p$ is the loss function at $p$}
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot g$
**end for**
**return** $\mathbf{w}$

If condition 1 does not hold, DriftSurf assumes there was no drift in the underlying distribution. It continues in the stable state by expanding $\mathcal{S}$ by adding the newly arrived set of data points $\mathbf{X}_t$ and updating the parameters of its predictive model. However, if DriftSurf decides to enter the reactive state, it changes its state to the reactive state, and freezes the current predictive model by copying its parameters to $\mathbf{w}^f$. In addition, DriftSurf adds a new model with randomly initialized parameters $\mathbf{w}'_{t-1}$. The sample set $\mathcal{S}'$ of this model is initialized to be empty. Note that this sample set does not need to be stored independently and consume additional memory space to store data points, but instead, it can be represented by pointers into $\mathcal{S}$.

If, at time step $t$, DriftSurf is in the reactive state (including the time step that it has just decided to enter the reactive state), DriftSurf adds $\mathbf{X}_t$ to $\mathcal{S}$ and $\mathcal{S}'$, sample sets of the predictive and reactive models. Then, it updates $\mathbf{w}_{t-1}$ and $\mathbf{w}'_{t-1}$. During the reactive state the choice of decision model follows a greedy approach meaning that among $\mathbf{w}$ and $\mathbf{w}'$ the one with the better performance in the previous time step will be chosen to be the current decision model. This helps to obtain good performance during the reactive state. The idea is that if there was no drift and entering the reactive state was the result of false drift detection, we expect the previous predictive model be the winner of this greedy approach. However, if entering the reactive state was the result of an actual drift, the greedy approach helps to switch to the newly added model sooner.

Upon exiting the reactive state, DriftSurf needs to choose the predictive model. It will switch to $\mathbf{w}'$ for its predictive model if the reactive model outperforms the previous predictive model over the set of data points, $T$, that arrived during the reactive state:

$$\mathcal{R}_T(\mathbf{w}') < \mathcal{R}_T(\mathbf{w}^f). \tag{2}$$

If condition 2 holds, DriftSurf resets its predictive model to the parameters of $\mathbf{w}'$. Note that sample set $\mathcal{S}$ will be reset accordingly, too. Otherwise, it continues with the updated version of the previous predictive model.

Algorithm 1 is generic in the individual base learner. Algorithm 2 shows an example where the parameters of each base learner are incrementally updated using SGD. For the purpose of theoretical analysis in Section 5 and experimental evaluation in Section 6, the update process we focus on is STRSAGA (Jothimurugesan et al., 2018), which is a variance-reduced SGD for streaming data, because of its fast convergence rate and ability to deal with different arrival distributions. For any update process, we let $\rho$ denote the computational power available at each time step. When the update process is an SGD-style algorithm, $\rho$ is the number of gradients that can be computed.

**Handling a corner case**. Consider the case that a drift happens when we are in the reactive state. In this case, no matter what predictive model we choose at the end of the reactive state, both the current predictive and reactive models are trained over a mixture of data points from both the old and new distributions. This will decrease the chance of recovering from the actual drift. To avoid this problem, DriftSurf adds a new model with parameters $\mathbf{w}''$ upon returning to the stable state. This model, which we refer to as the stable model, trains over the stream segment $\mathcal{S}''$ arrived after entering the stable state. DriftSurf keeps comparing the performance of the predictive and stable models. In addition to condition 1, DriftSurf enters the reactive state upon the following condition:

$$\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) > \mathcal{R}_{\mathbf{X}_t}(\mathbf{w}''_{t-1}) + \delta' \tag{3}$$

where $\delta'$ is set to be much smaller than $\delta$.

## 5. Analysis of DriftSurf

In this section, we show that DriftSurf achieves the goals **G1** and **G2** from Section 3 for an adaptive learning algorithm in the presence of (abrupt) concept drifts. As in prior work (Bousquet & Bottou, 2007; Jothimurugesan et al., 2018), we assume that $\mathcal{H}(n) = hn^{-\alpha}$, for a constant $h$ and $\frac{1}{2} \leq \alpha \leq 1$, is an upper bound on the statistical error over a set of data points of size $n$ drawn from the same distribution.

Aware is an adaptive learning algorithm with oracle knowledge of when drifts occur. At each drift, the algorithm restarts the predictive model to a random initial point and trains it over data points that arrive after the drift. The main obstacle for other adaptive learning algorithms to compete with Aware is that they are not told exactly when drifts occur.

The Aware implementation we are comparing to uses STRSAGA for incremental training (i.e., as its update process).

At any time step $t$, sub-optimality of this algorithm over its training sample set $\mathcal{S}$ of size $n$ is bounded as follows:

**Lemma 1.** (LEMMA 3 IN (JOTHIMURUGESAN ET AL., 2018)) *Suppose all $f_{\mathbf{x}}$ are convex and their gradients are $L$-Lipschitz continuous, and that $\mathcal{R}_{\mathcal{S}}$ is $\mu$-strongly convex. Also, assume that the condition number $L/\mu$ is bounded by a constant at each time. At the end of each time step $t$, the expected sub-optimality of* STRSAGA *over its sample set $\mathcal{S}$ of size $n$ is:*

$$\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}}(\text{STRSAGA})] \leq (1 + o(1))\mathcal{H}(n)$$

*where $\mathcal{H}(n) = hn^{-\alpha}$, for a constant $h$ and $\frac{1}{2} \leq \alpha \leq 1$.*

As a means of achieving goal **G2** (suboptimality on the order of $\mathcal{H}(n_{t_d,t})$ after a drift at time $t_d$), we will show that the empirical risk of DriftSurf after a drift is "close" to the risk of Aware, where *close* is defined formally in terms of our notion of risk-competitiveness in Definition 2.

**Definition 2.** *For $c \geq 1$, a recovered adaptive learning algorithm $A$ is said to be $c$-risk-competitive to* Aware *at time step $t > t_d$ if:*

$$\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(A)] \leq c \cdot (1 + o(1))\mathcal{H}(n_{t_d,t})$$

*where $t_d$ is the time step of the most recent drift. Also, $n_{t_d,t} = |\mathcal{S}_{t_d,t}|$, and $\mathcal{H}(n) = hn^{-\alpha}$, for a constant $h$ and $\frac{1}{2} \leq \alpha \leq 1$.*

We will analyze the risk-competitiveness of DriftSurf in two cases: (i) stationary environment and (ii) after a drift. Additionally, we will provide high probability analysis of the recovery time after a drift (goal **G1**).

For simplicity, in the rest of this section, we assume only a single abrupt drift happens at time step $t_d$. In addition, we assume all loss functions $f_{\mathbf{x}}$ are convex and their gradients are $L$-Lipschitz continuous, and that the empirical risk $\mathcal{R}_{\mathcal{S}}$ is $\mu$-strongly convex, where $\mu$ is the regularization hyperparameter. Lastly, we assume the condition number $L/\mu$ is bounded by a constant at each time step.

### 5.1. Stationary Environment

Our goal is to achieve an empirical risk that is competitive to that of Aware in a stationary environment. Note that at any time $0 < t < t_d$, before any drift happens, we are in a stationary environment. Based on Lemma 1, at any time $0 < t < t_d$ the expected sub-optimality of Aware and DriftSurf are (respectively) bounded by $(1 + o(1))\mathcal{H}(n_{0,t})$ and $(1 + o(1))\mathcal{H}(n_{t_e,t})$, where $t_e$ is the time that the current predictive model of DriftSurf was initialized. To prove DriftSurf is risk-competitive to Aware, we need to show that the expected size of its sample set $\mathcal{S}_{t_e,t}$ is close to $n_{0,t}$. To achieve this, we first in Lemma 2 (and similarly in Lemma 8) show that the probability of entering the reactive state in a stationary environment is very small.

**Lemma 2.** *In a stationary environment, at any time $t$ the probability of entering the reactive state because of condition 1 is bounded by $(2+o(1))n_{t_e,t}^{-\frac{1}{4}}$, where $|\mathcal{S}_{t_e,t}| = n_{t_e,t}$ and $\mathcal{S}_{t_e,t}$ is the stream segment that the predictive model of* DriftSurf *is trained over.*

At a high level, the proof (Appendix B.1) relies on showing that $\mathbb{E}[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b]$ is smaller than the sub-optimality of $\mathbf{w}_{t-1}$ over $\mathcal{S}_{t_e,t}$. The provided upper bound for the probability that condition 1 holds follows Lemma 1 and using Markov's inequality. A similar upper bound can be obtained for the probability that condition 3 holds.

Besides, if DriftSurf enters the reactive state, Lemma 3 shows that the probability of switching to the reactive model is very small.

**Lemma 3.** *In a stationary environment, if* DriftSurf *enters the reactive state, the probability of switching to the reactive model at the end of the reactive state is bounded by $O(e^{-(\frac{\beta-r}{3r})})$, where $r$ is the length of the reactive state and $\beta$ is the number of time steps that the predictive model was around before entering the reactive state, i.e. $|\mathcal{S}| = \beta \times m$.*

The proof relies on the fact that STRSAGA is a stochastic optimization method that provides the following property at each iteration $i$:

$$\mathbb{E}[\mathcal{R}_{\mathcal{S}}(\mathbf{w}_i) - \mathcal{R}^*_{\mathcal{S}}] \leq \min \begin{cases} \rho_n[\mathcal{R}_{\mathcal{S}}(\mathbf{w}_{i-1}) - \mathcal{R}^*_{\mathcal{S}}] \\ \min\limits_{k<n}[(\mathcal{R}_{T'}(\mathbf{w}_i) - \mathcal{R}^*_{T'}) + \dfrac{n-k}{n}\mathcal{H}(k)] \end{cases} \quad (4)$$

where $\rho_n = 1 - \min(\frac{1}{n}, \frac{\mu}{L})$, $|\mathcal{S}| = n$, and $T' \subset \mathcal{S}$ is of size $k$.

In the proof (Appendix B.1), we let $T'$ be the first $r \times m$ elements of $\mathcal{S}$. Note that the expected sub-optimality of $\mathbf{w}^f$ and $\mathbf{w}'$ over $T$ and $T'$ are the same. The provided upper bound for the probability that condition 2 holds follows the mentioned property of STRSAGA on $T'$ and using Markov's inequality.

Using the above results, we can compute the expected size of $\mathcal{S}$ for the predictive model of DriftSurf in a stable state.

**Corollary 1.** *In a stationary environment, at any time step $t > 0$ such that $\sqrt[4]{n_{0,t}} > 6\sqrt[4]{2}re^{1/3}$, the expected size of sample set $\mathcal{S}$ for the predictive model in a stable state is larger than $\frac{n_{0,t}}{4}$, where $n_{0,t}$ is the total number of data points arrived until time step $t$ and $r$ is the length of the reactive state.*

Based on the results of Corollary 1, we can provide the expected risk-competitiveness of DriftSurf's predictive model at any time $t$ in a stable state:

**Lemma 4.** *In a stationary environment,* DriftSurf *is expected $\frac{7}{4^{1-\alpha}}$-risk-competitive to* Aware*, at any time step $t > 0$ in a stable state such that $\sqrt[4]{n_{0,t}} > 6\sqrt[4]{2}re^{1/3}$, where $r$ is the length of the reactive state.*

*Proof.* Lemma 1 bounds the expected sub-optimality of DriftSurf over $\mathcal{S}_{t_e,t}$ as $(1+o(1))\mathcal{H}(n_{t_e,t})$. We apply property 4 to relate the expected sub-optimality over $\mathcal{S}_{t_e,t}$ to the expected sub-optimality over $\mathcal{S}_{0,t}$, where the expected size of $\mathcal{S}_{t_e,t}$ is bounded by Corollary 1 as $n_{t_e,t} \geq \frac{n_{0,t}}{4}$:

$$\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{0,t}}(\text{DriftSurf})] \leq (1+o(1))\mathcal{H}\left(\frac{n_{0,t}}{4}\right) + \frac{n_{0,t} - n_{0,t}/4}{n_{0,t}}\mathcal{H}\left(\frac{n_{0,t}}{4}\right) \leq \frac{7}{4^{1-\alpha}}(1+o(1))\mathcal{H}(n_{0,t}). \quad \square$$

In addition to expected risk-competitiveness, DriftSurf provides a minimum risk-competitiveness:

**Corollary 2.** *At any time step $t > 0$, the size of sample set $\mathcal{S}$ for the predictive model in a stable state is larger than $r \times m$, where $r$ is the length of a reactive state. Therefore,* DriftSurf *is at worst $2(\frac{t}{r})^{\alpha}$-risk-competitive with* Aware.

### 5.2. In Presence of an Abrupt Drift

Suppose an (abrupt) drift happens at any time $t_d$. Let $p$ ($p^*$) be the probability that DriftSurf enters a reactive state (switches to the reactive model at the end of a reactive state, respectively). In this section, we first show that w.h.p. DriftSurf has a small recovery time (goal **G1**).

**Lemma 5.** *Suppose a drift happens at time $t_d$. With probability $1-\epsilon$, the recovery time of* DriftSurf *is bounded by $kr + \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$, where $k < \sqrt{\frac{1-\epsilon_2}{\epsilon_2}(\frac{1-p^*}{p^{*2}})}$ is the number of times* DriftSurf *enters the reactive state before recovering from drift, and $\epsilon = \epsilon_1 + \epsilon_2$.*

The high-level proof sketch for this Lemma (full proof in Appendix B.2) is to divide the recovery time of DriftSurf into two parts: (i) time steps spent in reactive state, and (ii) time steps spent in the stable state before recovery. To bound the first part, we need to bound the number of times DriftSurf enters the reactive state and multiply that by $r$, the length of each reactive state. This can be obtained using Cantelli's inequality. On the other hand, the second part can be bounded by bounding the sum of $k$ independent geometric random variables, each with distribution $\sim Ge(p)$.

After recovering from the drift, the analysis is similar to the stationary case with the difference being that Aware, unlike the recovered model, uses data points arrived during the recovery time. Lemma 6 provides the risk-competitive analysis of DriftSurf at any time step $t > t_d$.

**Lemma 6.** *The predictive model of* DriftSurf *in the stable state is expected* $\frac{15}{8^{1-\alpha}}$*-risk-competitive with* Aware *with probability at least* $1-\epsilon$*, at any time step* $i > t_d + l$ *such that* $\sqrt[4]{m(t-t_d-l)} \geq \max(6\sqrt[4]{2}re^{1/3}, \sqrt[4]{ml})$*, where* $t_d$ *is the time of the drift,* $l = kr + \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$ *where* $k < \sqrt{\frac{1-\epsilon_2}{\epsilon_2}(\frac{1-p^*}{p^{*2}})}$*, and* $\epsilon = \epsilon_1 + \epsilon_2$.

*Proof.* Based on Lemma 5, with probability $1-\epsilon$, DriftSurf recovers from drift after $l = kT' + \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$ time steps, where $k < \sqrt{\frac{1-\epsilon_2}{\epsilon_2}(\frac{1-p^*}{p^{*2}})}$, and $\epsilon = \epsilon_1 + \epsilon_2$. After recovering from the drift, the situation is similar to the stationary case. Let $t_r$ be the time step that DriftSurf recovers from the drift at time $t_d$. Also, let $t_e$ be the time step that the current predictive model was initialized.

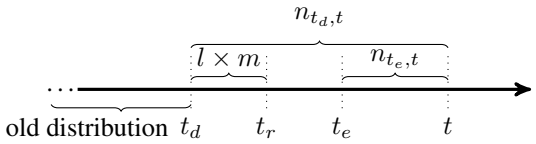



*Figure 1.* A drift happens at time $t_d$. DriftSurf recovers by time $t_r$. The current predictive model is initialized at time $t_e$.

To show DriftSurf is $\frac{15}{8^{1-\alpha}}$-risk-competitive to Aware, we want to show $n_{t_e,t} \geq \frac{n_{t_d,t}}{8}$. Using Corollary 1, we expect to have $n_{t_e,t} \geq \frac{n_{t_r,t}}{4}$ at any time step $t \geq t_r$ such that $\sqrt[4]{n_{t_r,t}} \geq 6\sqrt[4]{2}re^{1/3}$, where $r$ is the length of reactive state. Therefore, $3n_{t_e,t} \geq n_{t_r,t_e}$. On the other hand, we have

$$\begin{aligned} n_{t_e,t} &= n_{t_d,t} - n_{t_d,t_r} - n_{t_r,t_e} \\ &= n_{t_d,t} - l \times m - n_{t_r,t_e} \geq n_{t_d,t} - l \times m - 3n_{t_e,t}. \end{aligned}$$

Also, at any time step $t$ such that $\sqrt[4]{m(t-t_d-l)} \geq \max(6\sqrt[4]{2}re^{1/3}, \sqrt[4]{ml})$, we have $t - t_d \geq 2l$. Therefore,

$$4n_{t_e,t} \geq n_{t_d,t} - l \times m \geq \frac{n_{t_d,t}}{2}.$$

It remains to bound the expected sub-optimality over $\mathcal{S}_{t_d,t}$, which is similar to the proof of Lemma 4. Lemma 1 bounds the expected sub-optimality over $\mathcal{S}_{t_e,t}$ as $(1+o(1))\mathcal{H}(n_{t_e,t})$, and property 4 relates the expected sub-optimality over $\mathcal{S}_{t_e,t}$ to the expected sub-optimality over $\mathcal{S}_{t_d,t}$:

$$\begin{aligned} &\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(\text{DriftSurf})] \leq \\ &(1+o(1))\mathcal{H}\left(\frac{n_{t_d,t}}{8}\right) + \frac{n_{t_d,t} - n_{t_d,t}/8}{n_{t_d,t}}\mathcal{H}\left(\frac{n_{t_d,t}}{8}\right) \\ &\leq \frac{15}{8^{1-\alpha}}(1+o(1))\mathcal{H}(n_{t_d,t}). \quad \square \end{aligned}$$

## 6. Experimental Results

In this section, we present the experimental results. We empirically confirm the risk-competitiveness of DriftSurf with Aware throughout a set of experiments on streamed in synthetic, semi-synthetic datasets, real-world datasets in presence of drift. We also show the effectiveness of DriftSurf through comparison to two state-of-the-art adaptive learning algorithms, the ensemble method AUE and the drift-detection-based method MDDM, which are used with the same parameters provided by the authors. More detail on these algorithms is provided in Appendix C.1.

*Table 1.* Basic statistics of datasets

| | DATASET | # INSTANCE | # DIM |
|---|---|---|---|
| SYNTHETIC | SEA | 100000 | 3 |
| | HYPERPLANE | 100000 | 10 |
| | SINE1 | 10000 | 2 |
| SEMI-SYNTHETIC | RCV1 | 20242 | 47235 |
| | COVERTYPE | 581012 | 54 |
| REAL | AIRLINE | 5810462 | 13 |
| | ELECTRICITY | 45312 | 13 |
| | POWERSUPPLY | 29928 | 2 |

Table 1 presents an overview of the datasets used in our experiments. The prediction task for each dataset is binary classification. Drifts in semi-synthetic datasets are generated by rotating data points or changing the labels of the real-world datasets that originally do not contain any drift. We divide each dataset into equally-sized batches to arrive over the course of the stream. More detail on the datasets is provided in Appendix C.2.

In our experiments, a batch of data points arrives at each time step. We first evaluate the performance of each algorithm by measuring the misclassification rate over this batch, and then each algorithm gains access to the labeled data to update their model(s). The base learner in each algorithm is a logistic regression model trained using STRSAGA. Hyperparameter settings are discussed in Appendix C.3. All reported results of the misclassification rates represent the median over five trials.

We present the misclassification rates at each time step in Figures 2 and 3 on the PowerSupply and CoverType datasets (see Appendix D.1 for other datasets). A drift occurs at time 76 in PowerSupply, and at times 30 and 60 in CoverType. We observe DriftSurf outperforms MDDM because false positives in drift detection lead to unnecessary resetting of the predictive model in MDDM, while DriftSurf avoids the performance loss by catching most false positives via the reactive state and returning to the older model. In particular, the CoverType dataset was especially problematic for MDDM, which continually signaled a drift. We also observe DriftSurf adapts faster than AUE on CoverType. This is because after an abrupt drift, the predictions of DriftSurf are solely from the new model, while for AUE, the predictions are a weighted average of each expert in the ensemble. Immediately after a drift, the older, inaccurate experts of AUE

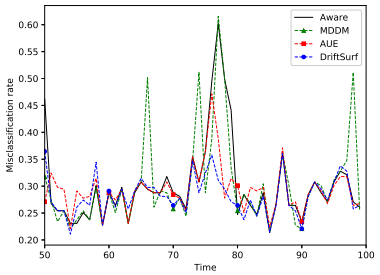



*Figure 2.* Misclassification rate over time for PowerSupply

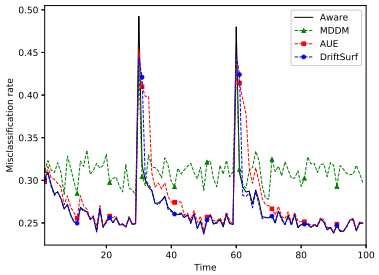



*Figure 3.* Misclassification rate over time for CoverType

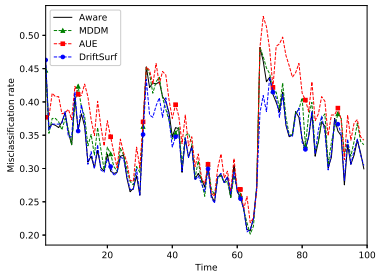



*Figure 4.* Misclassification rate over time for Airline ($\rho = 4m$ divided among models)

have reduced, but non-zero weights that negatively impact the accuracy. On both datasets, we observe the recovery time of DriftSurf is within one reactive state, and confirm that DriftSurf is competitive with Aware.

Table 2 summarizes the results for all the datasets in terms of the total average of the misclassification rate over time. To better demonstrate the impact of the reactive state strategy, we considered the sensitivity towards different levels of noise in the SEA dataset, shown in the first four rows of Table 2. We observe the stability of DriftSurf in the presence of noise, while the drift detection of MDDM suffers with false positives and overreacts to higher levels of noise.

We observe that on a majority of the datasets in Table 2, DriftSurf is the best performer. In some instances (Airline dataset), DriftSurf and AUE have a tight performance, and in some cases (Electricity dataset) AUE outperforms DriftSurf. A factor is the different computational power (number of gradient computations per time step) used by each algorithm. AUE maintains an ensemble of ten experts, while DriftSurf maintains just two, and so AUE uses five times the computation of DriftSurf. To account for the varying computational efficiency of each algorithm, we did another experiment where the available computational power for each algorithm is divided equally among all of its models, shown in Figure 4 for the Airline dataset, with more results in Appendix D.2. After normalizing for equal computational power, we observe DriftSurf has better accuracy and recovers faster after drift compared to AUE.

Appendices D.3–D.5 contain additional experimental results. In Appendix D.3, we report the results for single-pass SGD and an oblivious algorithm (STRSAGA with no adaptation to drift), which are generally worse across each dataset. One exception is that the oblivious algorithm has the best accuracy on the Electricity dataset because the drift does not warrant training a new model from scratch. Appendix D.4 studies the impact of DriftSurf's design choice of using greedy prediction during the reactive state, showing that it performs similarly or better than waiting until the end of the reactive state before deciding whether to transition to a new model. Finally, Appendix D.5 includes results for each algorithm when SGD is used as the update process instead of STRSAGA. We observe that using SGD results in lower accuracy for each algorithm, and also that, relatively, AUE gains an edge because its ensemble of ten experts mitigates the higher variance updates of SGD.

*Table 2.* Total average of misclassification rate

| DATASET | Aware | DriftSurf | MDDM | AUE |
|---|---|---|---|---|
| SEA0 | 0.139 | **0.088** | **0.088** | 0.094 |
| SEA10 | 0.199 | **0.158** | 0.176 | 0.163 |
| SEA20 | 0.267 | **0.245** | 0.288 | 0.248 |
| SEA30 | 0.350 | **0.335** | 0.363 | 0.339 |
| HYPER-SLOW | 0.118 | 0.117 | 0.117 | **0.113** |
| HYPER-FAST | 0.192 | 0.173 | **0.162** | 0.178 |
| SINE1 | 0.169 | 0.189 | **0.177** | 0.211 |
| RCV | 0.118 | **0.126** | 0.129 | 0.167 |
| COVERTYPE | 0.267 | **0.267** | 0.313 | 0.278 |
| AIRLINE | 0.338 | **0.334** | 0.347 | **0.334** |
| ELECTRICITY | 0.315 | 0.315 | 0.340 | **0.3** |
| POWERSUPPLY | 0.309 | **0.294** | 0.320 | 0.3 |

## 7. Conclusion

We presented DriftSurf, an adaptive algorithm for learning from streaming data that contains concept drifts. Our risk-competitive theoretical analysis showed that DriftSurf has high accuracy competitive with Aware both in a stationary environment and in the presence of abrupt drifts. Our experimental results confirmed our theoretical analysis and also showed high accuracy in the presence of either abrupt or gradual drift that generally outperforms state-of-the-art algorithms MDDM and AUE. Furthermore, DriftSurf maintains just two models while achieving high accuracy, and therefore its computational efficiency is significantly better than an ensemble method like AUE.

## References


Bach, S. H. and Maloof, M. A. Paired learners for concept drift. In *ICDM*, pp. 23–32, 2008.

Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., and Morales-Bueno, R. Early drift detection method. In *StreamKDD*, pp. 77–86, 2006.

Bifet, A. and Gavaldà, R. Learning from time-changing data with adaptive windowing. In *ICDM*, pp. 443–448, 2007.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. MOA: Massive online analysis. *JMLR*, 11:1601–1604, 2010.

Bousquet, O. and Bottou, L. The tradeoffs of large scale learning. In *NIPS*, pp. 161–168, 2007.

Brzezinski, D. and Stefanowski, J. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Trans. Neural Netw. Learn. Syst*, 25(1): 81–94, 2013.

Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., and Keogh, E. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Elwell, R. and Polikar, R. Incremental learning of concept drift in nonstationary environments. *IEEE Trans. Neural Netw.*, 22(10):1517–1531, 2011.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. Learning with drift detection. In *Advances in Artificial Intelligence-SBIA*, pp. 286–295, 2004.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44, 2014.

Harel, M., Crammer, K., El-Yaniv, R., and Mannor, S. Concept drift detection through resampling. In *ICML*, pp. 1009–1017, 2014.

Harries, M. Splice-2 comparative evaluation: Electricity pricing. Technical report, University of New South Wales, 1999.

Hentschel, B., Haas, P. J., and Tian, Y. Online model management via temporally biased sampling. *ACM SIGMOD Record*, 48(1):69–76, 2019.

Ikonomovska, E. Airline dataset. URL http://kt.ijs.si/elena_ikonomovska/data.html. (Accessed on 02/06/2020).

Janson, S. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018.

Jothimurugesan, E., Tahmasbi, A., Gibbons, P. B., and Tirthapura, S. Variance-reduced stochastic gradient descent on streaming data. In *NeurIPS*, pp. 9906–9915, 2018.

Kifer, D., Ben-David, S., and Gehrke, J. Detecting change in data streams. In *VLDB*, pp. 180–191, 2004.

Klinkenberg, R. Learning drifting concepts: Example selection vs. example weighting. *IDA*, 8(3):281–300, 2004.

Kolter, J. Z. and Maloof, M. A. Dynamic weighted majority: An ensemble method for drifting concepts. *JMLR*, 8: 2755–2790, 2007.

Koychev, I. Gradual forgetting for adaptation to concept drift. In *ECAI Workshop on Current Issues in Spatio-Temporal Reasoning*, 2000.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. RCV1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

Lu, Y., Cheung, Y.-m., and Tang, Y. Y. Dynamic weighted majority for incremental learning of imbalanced data streams with concept drift. In *IJCAI*, pp. 2393–2399, 2017.

Pesaranghader, A. and Viktor, H. L. Fast hoeffding drift detection method for evolving data streams. In *ECML PKDD*, pp. 96–111, 2016.

Pesaranghader, A., Viktor, H. L., and Paquet, E. A framework for classification in data streams using multi-strategy learning. In *ICDS*, pp. 341–355, 2016.

Pesaranghader, A., Viktor, H. L., and Paquet, E. McDiarmid drift detection methods for evolving data streams. In *IJCNN*, pp. 1–9, 2018.

Sebastião, R. and Gama, J. Change detection in learning histograms from data streams. In *PAI*, pp. 112–123, 2007.

Sun, Y., Tang, K., Zhu, Z., and Yao, X. Concept drift adaptation by exploiting historical knowledge. *IEEE Trans. Neural Netw. Learn. Syst.*, 29(10):4822–4832, 2018.

Widmer, G. and Kubat, M. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23 (1):69–101, 1996.

## A. Pseudocode of STRSAGA

The process for updating a model considered in this paper is STRSAGA (Jothimurugesan et al., 2018), shown in Algorithm 3. STRSAGA differs from Algorithm 2 (SGD) in that (i) it uses variance-reduced update steps that result in faster convergence, and (ii) it handles streaming data that do not arrive at a steady rate by controlling the rate at which its sample set grows. (In this paper, we only consider data that arrive at a fixed rate at each time step, but by using STRSAGA, the results can be readily extended to Poisson and other arrival distributions.) In STRSAGA, data points are not sampled from the entire available stream segment, but instead from a separately maintained sample set. Newly arriving data are first added to a buffer (called `WaitingRoom`), and then points are moved from `WaitingRoom` to the sample set at a controlled rate "to ensure that the optimization error on the subset that has been trained is balanced with the statistical error of the effective sample size" (Jothimurugesan et al., 2018). The implementation of STRSAGA we use in this paper uses the "alternating schedule" in its sampling.

**Algorithm 3** Update($\mathbf{w}, \mathcal{S}, \mathbf{X}_t$): Process of updating parameters $\mathbf{w}$ using STRSAGA, given sample set $\mathcal{S}$ and newly arrived data points $\mathbf{X}_t$

// $\rho$ is the computational power and determines the number of update steps that can be performed
// $\eta$ is the learning rate
Add $\mathbf{X}_t$ to `WaitingRoom` {`WaitingRoom` is the set of training points not added to $\mathcal{S}$ yet}
**for** $j = 1$ to $\rho$ **do**
  **if** `WaitingRoom` is non-empty & $j$ is even **then**
    Move a single point, $p$, from `WaitingRoom` to $\mathcal{S}$
    $\alpha(p) \leftarrow 0$ {$\alpha(p)$ is the prior gradient of $p$, initialized to 0}
  **else**
    Sample a point $p$ uniformly from $\mathcal{S}$
  **end if**
  $A \leftarrow \sum_{x \in \mathcal{S}} \alpha(x)/|\mathcal{S}|$ {$A$ is the average of all gradients and can be maintained incrementally}
  $g \leftarrow \nabla f_p(\mathbf{w})$ {$f_p$ is the loss function at $p$}
  $\mathbf{w} \leftarrow \mathbf{w} - \eta(g - \alpha(p) + A)$
  $\alpha(p) \leftarrow g$
**end for**
**return** $\mathbf{w}$

## B. Proofs from the Analysis of DriftSurf

This section contains proof details from the analysis of DriftSurf. Throughout, we will assume that $\mathcal{H}(n) = hn^{-\alpha}$, for a constant $h$ and $1/2 \leq \alpha \leq 1$, is an upper bound on the statistical error over a set of data points of size $n$. Also, assume all $f_x$ are convex and their gradients are $L$-Lipschitz continuous, and that $\mathcal{R}_\mathcal{S}$ is $\mu$-strongly convex for the set of training samples $\mathcal{S}$. In addition, we assume that the condition number $L/\mu$ is bounded by a constant at each time, and $\rho = 2m$, where $\rho$ denotes the number of gradients that can be computed at each time step $t$ and $m = |\mathbf{X}_t|$ is the number of points arriving at time $t$. Table 3 summarizes the notation used in this section.

*Table 3.* Summary of notation used in the analysis

| | |
|---|---|
| $\mathbf{X}_t$ | Data points arriving at time step $t$ |
| $m$ | = $\lvert\mathbf{X}_t\rvert$, the number of points arriving at each $t$ |
| $r$ | length of the reactive state (in time steps) |
| $\rho$ | the number of gradients computed at each time step |
| $\alpha$ | the exponent in the statistical error bound $\mathcal{H}(n) = hn^{-\alpha}$ |
| $p$ | probability DriftSurf enters a reactive state after a given drift |
| $p^*$ | probability DriftSurf switches to the reactive model at end of a given reactive state |

In Section 5, we defined Aware to be an adaptive learning algorithm with oracle knowledge of when drifts occur. Through Lemmas 4 and 6, we showed that under certain conditions DriftSurf is expected risk-competitive to Aware. Here, we state the

consequence with regards to the total error. As stated in Section 3, the total error of an algorithm $A$ over the stream segment $\mathcal{S}$ is the sum of the statistical and optimization errors; under uniform convergence bounds, the total error is bounded by $\mathcal{H}(|\mathcal{S}|) + \mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}}(A)]$ (Bousquet & Bottou, 2007). Empirical risk minimization (ERM), which is a process with no limit on the computational power, over a stream segment $\mathcal{S}$ yields a model with total error equal to the statistical error. When DriftSurf is risk-competitive with Aware, then the total error of DriftSurf can be bounded relative to the error of ERM by the following lemma.

**Lemma 7.** *Suppose the last drift occurred at time $t_d$. If* DriftSurf *is $c$-risk-competitive to* Aware *at time $t > t_d$, then the total error of* DriftSurf *is at most a $(c + 1 + o(1))$ factor of the error bound of* ERM*, $\mathcal{H}(\mathcal{S}_{t_d,t})$.*

*Proof.* By the definition of risk-competitiveness to Aware, $\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(\text{DriftSurf})] \leq c(1 + o(1))\mathcal{H}(n_{t_d,t})$. Adding the statistical error, the total error is at most $(c(1 + o(1)) + 1)\mathcal{H}(n_{t_d,t})$. □

Note that although the ERM error bound, $\mathcal{H}()$, is only an upper bound, it is usually considered to be a tight bound (Bousquet & Bottou, 2007).

In the remainder of this section we complete the proofs for the results in Section 5 that establish the conditions under which DriftSurf is risk-competitive with Aware both in a stationary environment and in the presence of an abrupt drift.

### B.1. In a Stationary Environment

In Lemma 2 and Lemma 8, we show the probability of entering the reactive state in a stationary environment is small.

**Lemma 2.** *In a stationary environment, at any time $t$ the probability of entering the reactive state because of condition 1 is bounded by $(2 + o(1))n_{t_e,t}^{-\frac{1}{4}}$, where $|\mathcal{S}_{t_e,t}| = n_{t_e,t}$ and $\mathcal{S}_{t_e,t}$ is the stream segment that the predictive model of* DriftSurf *is trained over.*

*Proof.* In a stationary environment, DriftSurf enters the reactive state at time $t$ because of condition 1 if $\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) > \mathcal{R}_b + \delta$. Therefore,

$$\begin{aligned}\Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) > \mathcal{R}_b + \delta] &= \Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta] \\ &= \Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta | A] \times \Pr[A] + \Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta | \bar{A}] \times \Pr[\bar{A}] \\ &\leq \Pr[A] + \Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta | \bar{A}]\end{aligned}$$

where $A$ is defined such that $\Pr[A] = \Pr\left[\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_{t-1}) - \mathcal{R}^*_{\mathcal{S}_{t_e,t}} \geq \mathbb{E}[\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_{t-1}) - \mathcal{R}^*_{\mathcal{S}_{t_e,t}}]/(n_{t_e,t})^{-\frac{\alpha}{2}}\right]$. Using Markov's inequality, we have $\Pr[A] \leq (n_{t_e,t})^{-\frac{\alpha}{2}}$. On the other hand, again using Markov's inequality we have,

$$\begin{aligned}\Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta] &\leq \mathbb{E}[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b]/\delta \\ &= \left(\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_{t-1}) - \mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_b)\right)/\delta \\ &= \left((\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_{t-1}) - \mathcal{R}^*_{\mathcal{S}_{t_e,t}}) - (\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_b) - \mathcal{R}^*_{\mathcal{S}_{t_e,t}})\right)/\delta \\ &\leq \left(\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_{t-1}) - \mathcal{R}^*_{\mathcal{S}_{t_e,t}}\right)/\delta\end{aligned}$$

where $\mathbf{w}_b$ is the parameters of the predictive model at the time we observed $\mathcal{R}_b$. Thus, $\Pr\left[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta | \bar{A}\right] \leq \mathbb{E}[\mathcal{R}_{\mathcal{S}_{t_e,t}}(\mathbf{w}_{t-1}) - \mathcal{R}^*_{\mathcal{S}_{t_e,t}}]/(\delta(n_{t_e,t})^{\frac{\alpha}{2}})$. Therefore, for $1/2 \leq \alpha \leq 1$ we have,

$$\Pr[\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) - \mathcal{R}_b > \delta] \leq \left(o(1) + \frac{\delta + 1}{\delta}\right) n_{t_e,t}^{-\frac{1}{4}}$$

□

The following lemma bounds the probability of entering the reactive state due to condition 3.

**Lemma 8.** *In a stationary environment, at any time $t$ the probability of entering the reactive state because of condition 3 is bounded by $(2 + o(1))n_{t_e,t}^{-\frac{1}{4}}$, where $|\mathcal{S}_{t_e,t}| = n_{t_e,t}$ and $\mathcal{S}_{t_e,t}$ is the stream segment that the predictive model of* DriftSurf *is trained over.*

*Proof.* Similar to the proof of Lemma 2. □

**Lemma 9.** *In a stationary environment, the probability of entering the reactive state at any time step $t$ is bounded by $(4 + o(1))n_{t_e,t}^{-\frac{1}{4}}$, where $|\mathcal{S}_{t_e,t}| = n_{t_e,t}$ and $\mathcal{S}_{t_e,t}$ is the stream segment that the predictive model of* DriftSurf *is trained over.*

*Proof.* Using Lemma 2 and Lemma 8. □

**Lemma 3.** *In a stationary environment, if* DriftSurf *enters the reactive state, the probability of switching to the reactive model at the end of the reactive state is bounded by $O(e^{-(\frac{\beta-r}{3r})})$, where $r$ is the length of the reactive state and $\beta$ is the number of time steps that the predictive model was around before entering the reactive state i.e. $|\mathcal{S}| = \beta \times m$.*

*Proof.* DriftSurf switches to the reactive model $\mathbf{w}'$ at the end of a reactive state if condition 2 holds, i.e. if $\mathcal{R}_T(\mathbf{w}^f) > \mathcal{R}_T(\mathbf{w}')$, where $\mathbf{w}^f$ is the state of the predictive model before entering the reactive state. The probability of this can be written as follows:

$$\begin{aligned}
\Pr[\mathcal{R}_T(\mathbf{w}^f) > \mathcal{R}_T(\mathbf{w}')] &= \Pr[\mathcal{R}_T(\mathbf{w}^f) - \mathcal{R}_T^* > \mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*] \\
&= \Pr[\mathcal{R}_T(\mathbf{w}^f) - \mathcal{R}_T^* > \mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^* | A] \Pr[A] \\
&+ \Pr[\mathcal{R}_T(\mathbf{w}^f) - \mathcal{R}_T^* > \mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^* | \bar{A}] \Pr[\bar{A}] \\
&\leq \Pr[A] + \Pr[\mathcal{R}_T(\mathbf{w}^f) - \mathcal{R}_T^* > \mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^* | \bar{A}]
\end{aligned}$$

where $A$ is defined such that $\Pr[A] = \Pr[\frac{\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^*}{\mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*} > \rho_{rm}^{-\frac{2}{3}(\beta-r)m}]$, where $T' \subseteq \mathcal{S}$ such that $|T'| = |T| = rm$, $\widetilde{\mathbf{w}'}$ is the predictive model after training over $T'$, and $\rho_{rm} = 1 - \min(\frac{1}{mr}, \frac{\mu}{L})$. $\Pr[A]$ can be bounded as follows:

$$\begin{aligned}
\Pr[\frac{\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^*}{\mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*} > \rho_{rm}^{-\frac{2}{3}(\beta-r)m}] &= \Pr[\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^* > \rho_{rm}^{-\frac{2}{3}(\beta-r)m}(\mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*)] \\
&= \Pr[\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^* > \rho_{rm}^{-\frac{2}{3}(\beta-r)m}(\mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*) | A'] \Pr[A'] \\
&+ \Pr[\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^* > \rho_{rm}^{-\frac{2}{3}(\beta-r)m}(\mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*) | \bar{A}'] \Pr[\bar{A}'] \\
&\leq \Pr[A'] + \Pr[\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^* > \rho_{rm}^{-\frac{2}{3}(\beta-r)m}(\mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*) | \bar{A}']
\end{aligned}$$

where $A'$ is defined such that $\Pr[A'] = \Pr[\rho_{mr}^{-(\frac{\beta-r}{3})m}\mathbb{E}[\mathcal{R}_{T'}(\widetilde{\mathbf{w}'}) - \mathcal{R}_{T'}^*] < \mathcal{R}_T(\mathbf{w}') - \mathcal{R}_T^*]$. Using Markov's inequality we have $\Pr[A'] \leq \rho_{mr}^{(\frac{\beta-r}{3})m}$ and as a result $\Pr[A] \leq 2\rho_{mr}^{(\frac{\beta-T'}{3})m}$. Therefore, using property 4 of STRSAGA we have,

$$\Pr[\mathcal{R}_T(\mathbf{w}^f) > \mathcal{R}_T(\mathbf{w}')] \leq 3\rho_{mr}^{(\frac{\beta-r}{3})m} \leq 3\left(1 - \frac{1}{mr}\right)^{(\frac{\beta-r}{3})m} \leq 3\left(\frac{1}{e}\right)^{(\frac{\beta-r}{3r})}$$

□

We can now prove Corollary 1.

**Corollary 1.** *In a stationary environment, at any time step $t > 0$ such that $\sqrt[4]{n_{0,t}} > 6\sqrt[4]{2}re^{1/3}$, the expected size of sample set $\mathcal{S}$ for the predictive model in a stable state is larger than $\frac{n_{0,t}}{4}$, where $n_{0,t}$ is the total number of data points arrived until time step $t$ and $r$ is the length of the reactive state.*

*Proof.* Lemma 9 and Lemma 3 (respectively) provide upper bounds on the probabilities of entering the reactive state and the probability of switching to the reactive model at the end of a reactive state. We can find a lower bound on the probability of $|\mathcal{S}| > \frac{mt}{k}$ for some value of $k > 1$ and therefore:

$$
\begin{aligned}
\mathbb{E}[|\mathcal{S}|] &\geq \frac{mt}{k} \prod_{j=(\frac{k-1}{k})t}^{t} \left(1 - \frac{e^{-\frac{j-r}{3r}}}{\sqrt[4]{mj}}\right) \\
&\geq \frac{mt}{k} \left(1 - \sum_{j=(\frac{k-1}{k})t}^{t} \frac{e^{-\frac{j-r}{3r}}}{\sqrt[4]{mj}}\right) \\
&= \frac{mt}{k} \left(1 - \frac{e^{1/3}}{\sqrt[4]{m}} \sum_{j=(\frac{k-1}{k})t}^{t} \frac{1}{e^{j/3r} j^{1/4}}\right) \\
&\geq \frac{mt}{k} \left(1 - \frac{e^{1/3}}{\sqrt[4]{m}} \sum_{j=(\frac{k-1}{k})t}^{t} \frac{1}{(1+\frac{j}{3r}) j^{1/4}}\right) \\
&\geq \frac{mt}{k} \left(1 - \frac{3re^{1/3}}{\sqrt[4]{m}} \sum_{j=(\frac{k-1}{k})t}^{t} \frac{1}{j^{5/4}}\right) \\
&\geq \frac{mt}{k} \left(1 - \frac{3re^{1/3}}{\sqrt[4]{m}} \left[\frac{(\frac{1}{\sqrt[4]{2}})^{\lfloor \log_2 (\frac{k-1}{k})t \rfloor} - (\frac{1}{\sqrt[4]{2}})^{\lfloor \log_2 t \rfloor}}{1 - \frac{1}{\sqrt[4]{2}}}\right]\right) \\
&= \frac{mt}{k} \left(1 - \frac{3re^{1/3}}{\sqrt[4]{m}} \left[\frac{(\frac{1}{\sqrt[4]{2}})^{\lfloor \log_2 t \rfloor} ((\frac{k-1}{k})^{-1/4} - 1)}{(1 - 2^{-1/4})}\right]\right) \\
&= \frac{mt}{k} \left(1 - \frac{3re^{1/3}}{(tm)^{1/4}} \left[\frac{(\frac{k-1}{k})^{-1/4} - 1}{1 - 2^{-1/4}}\right]\right)
\end{aligned}
$$

where the second line uses Weierstrass' inequality. Let $k = 2$ and $\sqrt[4]{tm} > 6\sqrt[4]{2}re^{1/3}$, therefore $\mathbb{E}[|\mathcal{S}|] \geq \frac{tm}{4}$.

□

With the preceding lemmas, we can now establish the risk-competitiveness of DriftSurf in the stationary case. The full proof is given in Section 5.

**Lemma 4.** *In a stationary environment,* DriftSurf *is expected* $\frac{7}{4^{1-\alpha}}$*-risk-competitive to* Aware*, at any time step* $t > 0$ *in a stable state such that* $\sqrt[4]{n_{0,t}} > 6\sqrt[4]{2}re^{1/3}$*, where* $r$ *is the length of the reactive state.*

Corollary 2 guarantees a minimum risk-competitiveness.

**Corollary 2.** *At any time step* $t > 0$*, the size of sample set* $\mathcal{S}$ *for the predictive model in a stable state is larger than* $r \times m$*, where* $r$ *is the length of a reactive state. Therefore,* DriftSurf *is at worst* $2(\frac{t}{r})^{\alpha}$*-risk-competitive with* Aware*.*

*Proof.* The proof is similar to the proof of Lemma 4 and is a consequence of the algorithm's design as DriftSurf may change its predictive model only at the end of a reactive state, which lasts $r$ time steps. □

### B.2. In Presence of an Abrupt Drift

For the case of abrupt drift, we first bound the recovery time for DriftSurf through Lemmas 10 and 5, and then establish risk-competitiveness after recovery in Lemma 6. To simplify the analysis, we use two parameters, $p$ and $p^*$, where $p$ represents the probability that DriftSurf enters a reactive state at a time step after the drift and $p^*$ represents the probability that DriftSurf switches to the reactive model at the end of a reactive state. These two parameters are hard to analytically estimate because they depend on the magnitudes and frequencies at which concept drifts occur (which in turn impact the age of the stable model and consequently its risk), for which there is no agreed upon model. Additionally, the reactive and stable models are trained over points drawn from different distributions, and so $p^*$, which is determined by the difference in risks of the two models, may depend on the different *approximation errors* for each distribution (the approximation error is the

optimal expected risk within the function class $\mathcal{F}$). By parameterizing $p, p^*$, we are able to show the general results in this section without making too simplistic assumptions about concept drifts. (Note that such concerns did not arise in the simpler setting of a stationary environment in Section B.1.)

**Lemma 10.** *Suppose a drift happens at time $t_d$. With probability at least $1-\epsilon$, the number of times* DriftSurf *enters the reactive state before recovering from this drift is less than $\sqrt{\frac{1-\epsilon}{\epsilon}(\frac{1-p^*}{p^{*2}})}$.*

*Proof.* Let $X$ be a random variable denoting the number of times DriftSurf enters the reactive state after the drift at time $t_d$ and before recovering from it. Using Cantelli's inequality for any real number $\lambda > 0$, we have:

$$\Pr[X - \mu \geq \lambda] \leq \frac{\sigma^2}{\sigma^2 + \lambda^2}$$

where $\mu = \mathbb{E}[X] = \frac{1}{p^*}$ and $\sigma^2 = \mathbb{V}ar\,[X] = \frac{1-p^*}{p^{*2}}$. Let $\lambda = \frac{k}{p^*}$, therefore,

$$\Pr[X \geq \frac{(k+1)}{p^*}] \leq \frac{1}{1 + \frac{k^2}{1-p^*}} \leq \epsilon$$

□

Using Lemma 10, w.h.p. we can estimate the recovery time of DriftSurf as follows:

**Lemma 11.** *Let $X = \sum_{i=1}^{k} X_i$, where $k \geq 1$ and $X_i$ for $i = 1, ..., k$, are independent geometric random variables distributed $X_i \sim Ge(p)$ and $\mathbb{E}[X] = \frac{k}{p}$. For any $\lambda \geq 1$, we have:*

$$\Pr\left[X \geq \frac{\lambda k}{p}\right] \leq e^{-k(\frac{\lambda}{2} - \ln 2)}$$

*Proof.* Similar to the proof of Theorem 2.1 in (Janson, 2018) and by setting parameter $t$ (defined in their proof) to $\frac{p}{2}$. □

We can now prove Lemma 5 from Section 5.2.

**Lemma 5.** *Suppose a drift happens at time $t_d$. With probability $1-\epsilon$, the recovery time of* DriftSurf *is bounded by $kr + \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$, where $k < \sqrt{\frac{1-\epsilon_2}{\epsilon_2}(\frac{1-p^*}{p^{*2}})}$ is the number of times* DriftSurf *enters the reactive state before recovering from drift, and $\epsilon = \epsilon_1 + \epsilon_2$.*

*Proof.* Let $X = \sum_{i=1}^{k} X_i$, where $k \geq 1$ and $X_i$ for $i = 1, ..., k$, are independent geometric random variables with distributions: $X_i \sim Ge(p)$. Using Lemma 11 for $\lambda = 1$ we have:

$$\Pr\left[X \geq \frac{k}{p}\right] \leq e^{-k(\frac{1}{2} - \ln 2)}$$

Therefore, with probability at least $1-\epsilon_1$, we have $X < \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$. Consequently, w.h.p. the total number of time steps before recovering from the drift will be less than $kr + \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$. Besides, using Lemma 10 we have with probability $1-\epsilon_2$, $k < \sqrt{\frac{1-\epsilon_2}{\epsilon_2}(\frac{1-p^*}{p^{*2}})}$. □

With the preceding lemmas, we can now establish the risk-competitiveness of DriftSurf following an abrupt drift. The full proof is given in Section 5.

**Lemma 6.** *The predictive model of* DriftSurf *in the stable state is expected $\frac{15}{8^{1-\alpha}}$-risk-competitive with* Aware *with probability at least $1-\epsilon$, at any time step $i > t_d + l$ such that $\sqrt[4]{m(t - t_d - l)} \geq \max(6\sqrt[4]{2}re^{1/3}, \sqrt[4]{ml})$, where $t_d$ is the time of the drift, $l = kr + \frac{2}{p}(\ln\frac{1}{\epsilon_1} + k\ln 2)$ where $k < \sqrt{\frac{1-\epsilon_2}{\epsilon_2}(\frac{1-p^*}{p^{*2}})}$, and $\epsilon = \epsilon_1 + \epsilon_2$.*

## C. Additional Details on the Experimental Setup

This section contains additional details on the algorithms, datasets, and training for the experimental evaluation.

### C.1. Algorithms Evaluated

In our experimental evaluation, we compare our algorithm DriftSurf to MDDM (Pesaranghader et al., 2018) and AUE (Brzezinski & Stefanowski, 2013), as representatives of state-of-the-art drift-detection-based and ensemble-based algorithms, respectively. The MDDM algorithm maintains a sliding window over the prediction results, which is a binary series indicating for each data point whether the model's predicted label matches the true label. MDDM signals a drift whenever a weighted mean over the sliding window is worse than the best observed weighted mean so far by a specified threshold. Upon signaling a drift, the current model is discarded and a new model is initialized starting at the current time step. Pesaranghader et al. offer three variants of their algorithm, MDDM-A, MDDM-G, and MDDM-E, differing in the weighting scheme applied over the sliding window. Pesaranghader et al. remark that "all three variants had comparable levels of accuracy" across each dataset they tested and that "the optimal shape for the weighting function is data, context and application dependent" (Pesaranghader et al., 2018). Generally, we do not know the type of drifts that will occur in advance, and so in our experiments, we used the intermediate choice MDDM-G, corresponding to a geometric weighting. (We also perform a sensitivity study among all three variants.) We reused the source code for MDDM-G available in the Tornado framework from Pesaranghader et al., and we used their default parameters for their algorithm: the window size $n = 100$, the confidence level $\delta_w = 10^{-6}$, and the geometric weighting factor $r = 1.01$.

The AUE algorithm (sometimes called AUE2 to distinguish from a preliminary published version of the algorithm) manages an ensemble of $k$ experts that are incrementally trained over the stream. After each batch of arrivals, AUE updates the weight of each expert based on its prediction error, and drops the lowest weighted expert to introduce a new expert. The prediction output from the ensemble is a weighted vote by its experts. We used the parameter $k = 10$ as the limit on the total number of experts, following the choice made by Brzezinski and Stefanowski in their experimental evaluation (Brzezinski & Stefanowski, 2013).

For the implementation of our algorithm DriftSurf, we used the following parameters. The length of the reactive state $r = 4$. Regarding the conditions to enter the reactive state described in Section 4, the threshold for condition 1 is $\delta = 0.1$, and the threshold for condition 3 is $\delta' = \delta/2$.

In our main experiment, on each dataset discussed below, we evaluate DriftSurf, MDDM (the MDDM-G variant), AUE, and the Aware algorithm with oracle access to when drifts occur (discussed in Section 5). We also run additional experiments for MDDM-A, MDDM-E, single-pass SGD, and an oblivious algorithm, which maintains a single model updated with STRSAGA. The version of STRSAGA in the oblivious algorithm samples uniformly from its sample set at each iteration and has no bias towards sampling more recent data arrivals.

When using STRSAGA or any other SGD-style optimization, we consider a parameter $\rho$ that dictates the number of update steps (specifically, gradient computations) that are available to train the model. The four adaptive learning algorithms maintain a different number of models—DriftSurf uses 2, Aware and MDDM use 1, and AUE uses 10. This leads us to consider two different possibilities for training at each time: (1) each algorithm can use $\rho$ steps per model; or (2) each algorithm has $\rho$ steps in total that are divided equally across its models. The second approach accounts for the varying computational efficiency of each algorithm and lets us examine the accuracy achieved when enforcing equal processing time.

### C.2. Datasets

Our experiments use the 3 synthetic, 2 semi-synthetic and 3 real-world datasets shown in Table 1 and described below. The selection of datasets included all datasets used in the experimental evaluations by Pesaranghader et al. on their MDDM algorithm (namely, SINE1 and Electricity) and Brzezinski and Stefanowski on their AUE algorithm (SEA10, Hyperplane-Slow, Hyperplane-Fast, Electricity, and Airlines).

- SEA (Bifet et al., 2010): This dataset is generated using the Massive Online Analysis (MOA) framework. There are three attributes in $[0, 10]$. The label is determined by $x_1 + x_2 \leq \theta_j$ where $j$ corresponds to 4 different concepts, $\theta_1 = 9, \theta_2 = 8, \theta_3 = 7, \theta_4 = 9.5$ (the third attribute $x_3$ is not correlated with the label). We synthetically generated 25000 points from each concept in the order 3, 2, 4, 1, following the example from the MOA manual. We experimented on four different datasets varying the amount of noise, SEA0, SEA10, SEA20, SEA30, corresponding to 0%, 10%, 20%, and 30% of the labels being swapped during the generation of the dataset.

- Hyperplane (Bifet et al., 2010): This dataset is generated using the MOA framework. For each data point, the label corresponds to its half space for an underlying hyperplane, where each coordinate of the hyperplane changes by some

magnitude for each point in the stream, representing a continually gradually drifting concept. We experimented on two variations, Hyperplane-Slow and Hyperplane-Fast, corresponding to a 0.001 and a 0.1 magnitude of change. In each case, at each point in the stream, there is a 10% probability that the direction of the change is reversed.

- SINE1 (Pesaranghader et al., 2016): This dataset contains two attributes $(x_1, x_2)$, uniformly distributed in $[0, 1]$. Label of each data is determined using a sine curve as follows: $x_2 \leq sin(x_1)$. Labels are reversed at drift points.

- RCV1 (Lewis et al., 2004): This real world data set contains manually categorized newswire stories. The original order of the data set we used was randomly permuted before inserting drift. At drift points, we introduce a sharp abrupt drift by swapping each label.

- Covertype (Dua & Graff, 2017): This real world data set contains observation of a forest area obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS). Binary class labels are involved to represent the corresponding forest cover type. The original order of the data set we used was randomly permuted before inserting drift. At drift points, we introduce an abrupt drift by rotating each data point by $180°$ along the 1st and 8th attributes. This particular rotation was chosen because it resulted in approximately 40% misclassification rate with respect to the current predictive model.

- Airline(2008) (Ikonomovska): This real world data set contains records of flight schedules. Binary class labels are involved to represent if a flight is delayed or not. Concept drift could appear as the result of changes in the flights schedules, e.g. changes in day, time, and the length of flights. In our experiments, we used the first 58100 points of the data set, and pre-processed the data by using one-hot encoding for categorical features and scaling numerical features to be in the range $[0, 1]$.

- Electricity (Harries, 1999): This real world data set contains records of the New South Wales Electricity Market in Australia. Binary class labels are involved to represent the change of the price (i.e., up and down). The concept drift may result from changes in consumption habits or unexpected events.

- Power Supply (Dau et al., 2019): This real world data set contains records of hourly power supply of an Italy electricity company which records the power from two sources: power supply from main grid and power transformed from other grids. Binary class labels are involved to represent which time of day the current power supply belongs to (i.e. am or pm). The concept drifting in this stream may results from the change in season, weather or the differences between working days and weekend.

The type of drift in each dataset is detailed in Table 4. When working with real datasets, precisely determining the time drift occurs is somewhat guesswork. Brzezinski and Stefanowski remarked they "cannot unequivocally state when drifts occur or if there is any drift" on the real datasets they considered (Brzezinski & Stefanowski, 2013). Still, we had to mark the drift times for the implementation of Aware, which resets the model whenever drifts occur. We chose these times by observing the misclassification rates of an oblivious algorithm that is not designed to adapt to drift, and noting for which time steps there was a significant increase in misclassifications on the newly arrived batch.

*Table 4.* Details of drifts in datasets

| | DATASET | DRIFT TYPE | DRIFT TIMES |
|---|---|---|---|
| SYNTHETIC | SEA | ABRUPT | [25, 50, 75] |
| | HYPERPLANE | GRADUAL | - |
| | SINE1 | ABRUPT | [20, 40, 60, 80] |
| SEMI-SYNTHETIC | RCV1 | ABRUPT | [30, 60] |
| | COVERTYPE | ABRUPT | [30, 60] |
| REAL | AIRLINE | - | [31, 67] |
| | ELECTRICITY | - | [20] |
| | POWERSUPPLY | - | [17, 47, 76] |

### C.3. Training and Hyperparameters

On each dataset, the prediction task is binary classification. Each model $\mathbf{w}$ trained is a linear model, using STRSAGA to optimize the L2-regularized logistic loss over the relevant stream segment. For a data point $(x, y)$, the corresponding loss function is $f_{(x,y)}(\mathbf{w}) = \log(1 + \exp(-y\mathbf{w}^T x)) + \frac{\mu}{2}||\mathbf{w}||_2^2$.

There are two hyperparameters used by STRSAGA, the regularization factor $\mu$ and the constant step size $\eta$. To set them, we first took each dataset in static form (opposed to streaming) and applied a random permutation, partitioning an 80% split for training and 20% for validation. (For the case of the semi-synthetic datasets where we introduced our own drift, the hyperparameter selection was done prior to modifying the data.) We used grid search to determine the values of $\mu$ and $\eta$ that optimized the validation set error after running STRSAGA over the static training set for a number of iterations equal to two times the number of data points. We searched for $\mu$ of the form $10^{-a}$ for $1 \leq a \leq 7$ and $\eta$ of the form $b \times 10^{-c}$ for $b \in \{1, 2, 5\}$ and $1 \leq c \leq 5$. The parameters we chose are given in Table 5. In experiments where we used SGD for training, we used the same constant step size $\eta$.

*Table 5.* Hyperparameters and batch sizes

| DATASET | REGULARIZATION $\mu$ | STEP SIZE $\eta$ | BATCH SIZE $m$ |
|---|---|---|---|
| SEA (ALL) | $10^{-2}$ | $1 \times 10^{-3}$ | 1000 |
| HYPER-SLOW | $10^{-3}$ | $1 \times 10^{-1}$ | 1000 |
| HYPER-FAST | $10^{-3}$ | $1 \times 10^{-2}$ | 1000 |
| SINE1 | $10^{-3}$ | $2 \times 10^{-1}$ | 100 |
| RCV1 | $10^{-5}$ | $5 \times 10^{-1}$ | 202 |
| COVERTYPE | $10^{-4}$ | $5 \times 10^{-3}$ | 5810 |
| AIRLINE | $10^{-3}$ | $2 \times 10^{-2}$ | 581 |
| ELECTRICITY | $10^{-4}$ | $1 \times 10^{-1}$ | 1333 |
| POWERSUPPLY | $10^{-3}$ | $1 \times 10^{-1}$ | 299 |

In the streaming data setting studied in this paper (Section 3), the batch size is determined by the rate of arrival of new data points, and hence not a hyperparameter to be tuned. For simplicity, we assume that data arrive over the course of $b$ time steps in equally-sized batches containing $m = (\text{dataset size})/b$ points, where $b = 100$ for all datasets other than Electricity. For the case of Electricity, we defined the number of time steps $b = 34$ so that one time step corresponds to 28 days of the collected data, and was a scale where we could visually observe drift in the results. The resulting batch sizes are shown in the last column of Table 5.

## D. Additional Experimental Results

This section contains experimental results under both training strategies of equal computational power for each model and equal computational power for each algorithm, which is divided among its models. Additionally, we report results for single-pass SGD and an oblivious algorithm using STRSAGA, results for DriftSurf without the greedy approach during the reactive state, and results for each algorithm when SGD is used as the update process instead of STRSAGA.

### D.1. Equal Computational Power for each Model

We present the misclassification rates at each time step over the new batch in Figure 5, and the average misclassification rate over all time steps is summarized in Table 6. (These results are a superset of those presented in Figures 2 and 3 and Table 2 from Section 6). Here, we used the training strategy where at every time step, each algorithm uses $\rho = 2m$ update steps for each of its models. Let us note a few general trends. The advantage of DriftSurf over MDDM is most evident on the noisy versions of SEA (also shown in Figure 6), and on CoverType and PowerSupply. The drift detection method MDDM encounters false positives that lead to unnecessary resetting of the predictive model, while DriftSurf avoids the performance loss after most of the false positives by catching them via the reactive state. In particular, the CoverType dataset was especially problematic for MDDM, which continually signaled a drift.

For true drifts when immediately switching to a new model is desirable, we observe, most evident on SINE1 and RCV1, that MDDM is the fastest to adapt, followed shortly by DriftSurf, and with AUE lagging behind. CoverType also is a clear example where DriftSurf adapts faster than AUE (but MDDM suffered as previously mentioned). For these drifts, MDDM

naturally leads because it is using a new model when it accurately detects the drift, while DriftSurf always takes at least one time step to switch because it waits until it sees a batch where the new (reactive) model outperforms the older (stable) model. Finally, AUE also takes at least one time step, because its ensemble members are weighted based on the previous performance, but it can take longer, because even if the older, inaccurate models are low-weighted, they are not weighted zero, and shortly after a drift, most of the models in the ensemble are trained on old data and can still negatively impact the predictions.

There are two major advantages of DriftSurf and AUE not immediately switching to the latest model: (i) there are drifts for which switching to a new model is not desired because the older model can still provide good accuracy, and (ii) delaying the switch to a new model can be desired if the new model has poor accuracy immediately after the drift while it warms up. Regarding the first point, observe the drift in SEA10 at $t = 25$ and the drift in Electricity. There is a notable degradation in accuracy of each algorithm at the time of the drift, but resetting the model as Aware does is a poor choice. We even observe that the oblivious algorithm (OBL) (which trains a model from the beginning of time and is not designed to adapt to drifts) outperforms Aware on these datasets. Despite the initial degradation in accuracy at the time of drift, we find that the older model is able to converge again after the drift, even while the older model is trained on data from both before and after the drift. Meanwhile, training a new model from scratch as Aware does is not worth the initial start-up cost when the older model performs well.

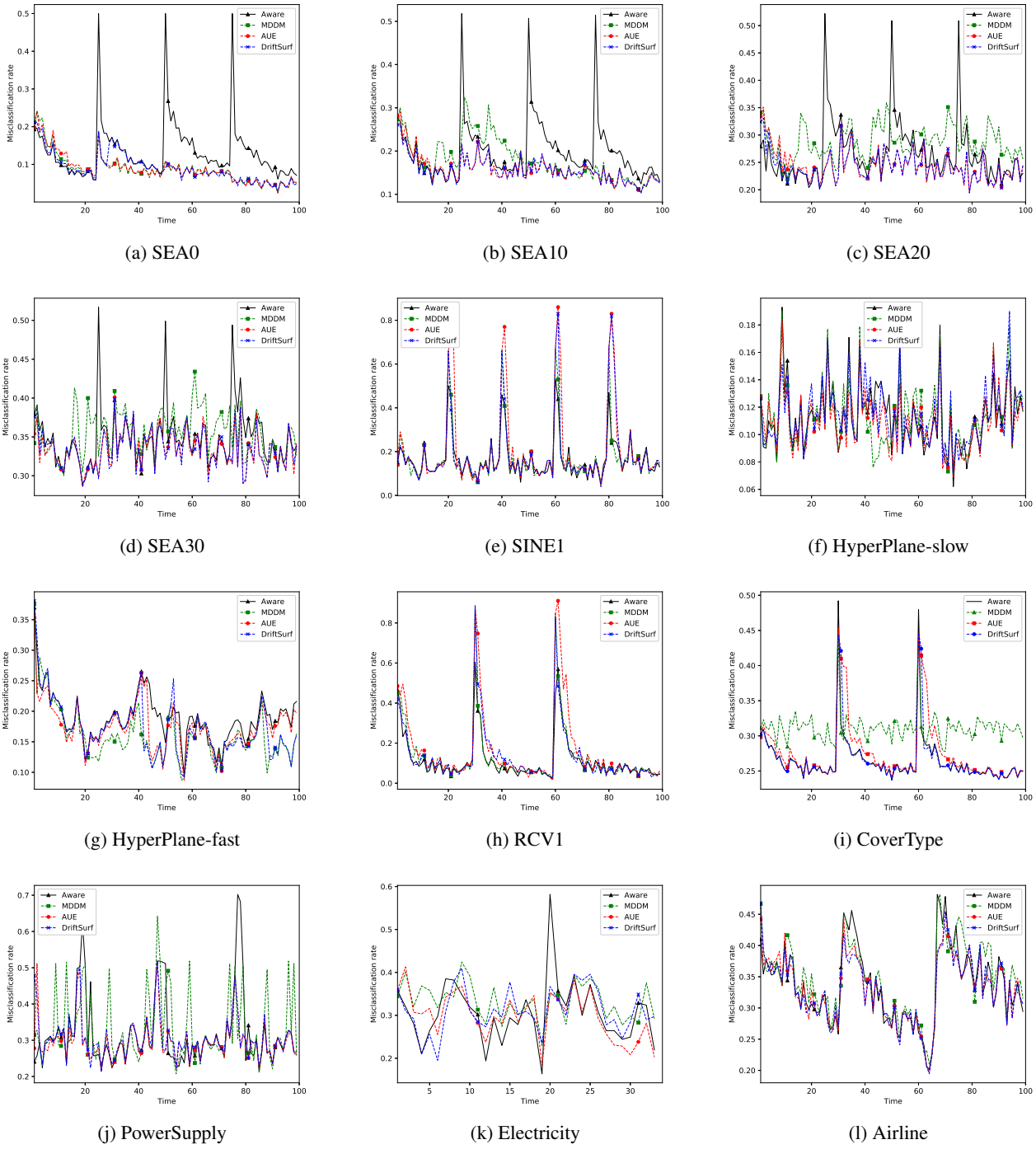
The reader may be skeptical specifically of Aware's reset to a random model for predictions at the time step drift occurs—practically, wouldn't it be preferable to use the previously-learned model for the first time step, and then switch to the new model? We considered this alternative implementation of Aware, and observed that across each dataset, the average misclassification rate of the alternative Aware was better by at most 1.1 percentage points than the version of Aware reported in Table 6, and was worse on SINE1 and RCV1. There was no case where the alternative Aware outperformed any algorithm in the table that Aware did not already outperform.

The second advantage previously mentioned, of delaying the switch to the new model, is best exemplified on Airline. Immediately after the two drifts, DriftSurf is the best performer, followed by AUE, and then MDDM and Aware. Immediately after the drift, DriftSurf continues to use the older, stable model, which outperforms a newly created model (compare DriftSurf to Aware), because a new model needs a few time steps to train before it is a better choice, and then DriftSurf switches later. AUE is of intermediate error in the time steps immediately after the drift, because it does place greater weight on the better performing, older models, but is still worse than placing unit weight on an old model.

Finally, the Hyperplane-slow and Hyperplane-fast warrant their own discussion. These two datasets represent a continually drifting concept throughout the entire stream. For Hyperplane-slow, AUE is the best performing algorithm, while for Hyperplane-fast, MDDM is the best performing. The advantage that AUE and MDDM have over DriftSurf in these datasets is that AUE adds a new model at every time step, and MDDM has the capability of switching to a new model at any time step, and therefore, they can better fit the most recent data in the stream. On the other hand, DriftSurf is only able to create a new model upon transitioning to the reactive state, so DriftSurf does not have the capability of creating new models at time steps during its reactive state. DriftSurf is not designed for the setting where creating a new model at every time step is desirable, but nonetheless, the accuracy of DriftSurf is still comparable.

*Table 6.* Total average of misclassification rate ($\rho = 2m$ for each model)

| DATASET | Aware | DriftSurf | MDDM-G | MDDM-A | MDDM-E | AUE | 1PASS-SGD | OBL |
|---|---|---|---|---|---|---|---|---|
| SEA0 | 0.139 | 0.088 | 0.088 | 0.090 | **0.087** | 0.094 | 0.118 | 0.105 |
| SEA10 | 0.199 | **0.158** | 0.176 | 0.166 | 0.172 | 0.163 | 0.191 | 0.174 |
| SEA20 | 0.267 | **0.245** | 0.288 | 0.278 | 0.289 | 0.248 | 0.274 | 0.253 |
| SEA30 | 0.350 | **0.335** | 0.363 | 0.358 | 0.352 | 0.339 | 0.356 | 0.339 |
| HYPER-SLOW | 0.118 | 0.117 | 0.117 | 0.117 | 0.116 | **0.113** | 0.123 | 0.160 |
| HYPER-FAST | 0.192 | 0.173 | **0.162** | 0.163 | 0.164 | 0.178 | 0.173 | 0.281 |
| SINE1 | 0.169 | 0.189 | 0.177 | **0.175** | 0.178 | 0.211 | 0.223 | 0.480 |
| RCV1 | 0.118 | **0.126** | 0.129 | 0.130 | 0.130 | 0.167 | 0.276 | 0.468 |
| COVERTYPE | 0.267 | **0.267** | 0.313 | 0.311 | 0.313 | 0.278 | 0.298 | 0.321 |
| AIRLINE | 0.338 | **0.334** | 0.347 | 0.346 | 0.348 | **0.334** | 0.343 | 0.359 |
| ELECTRICITY | 0.315 | 0.315 | 0.340 | 0.339 | 0.341 | 0.300 | 0.346 | **0.291** |
| POWERSUPPLY | 0.309 | **0.294** | 0.320 | 0.315 | 0.329 | 0.300 | 0.306 | 0.310 |




(a) SEA0 (b) SEA10 (c) SEA20

(d) SEA30 (e) SINE1 (f) HyperPlane-slow

(g) HyperPlane-fast (h) RCV1 (i) CoverType

(j) PowerSupply (k) Electricity (l) Airline

*Figure 5.* Misclassification rate over time ($\rho = 2m$ for each model)

Table 6 includes results for MDDM-G (what we use generally for MDDM), as well as two other MDDM variants, MDDM-A and MDDM-E, for a more thorough comparison. The average misclassification rates were similar across each dataset, with no single MDDM variant that consistently outperformed the others. Given the poor peformance of MDDM on CoverType, we re-did the experiment on CoverType with two other drift detection methods, DDM (Gama et al., 2004) and EDDM (Baena-García et al., 2006) to investigate further. In Figure 7, we observed DDM accurately detected the two drifts, but EDDM also suffered with continual false positives.

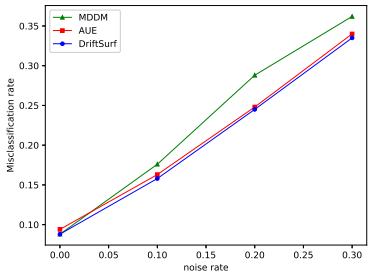
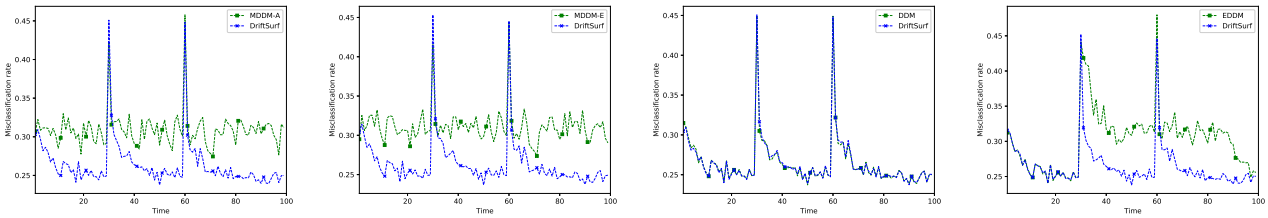



*Figure 6.* Total average of misclassification rate for SEA dataset with different levels of noise




(a) MDDM-A (b) MDDM-E (c) DDM (d) EDDM

*Figure 7.* Covertype dataset, different drift detectors ($\rho = 2m$ for each model)

### D.2. Equal Computational Power for each Algorithm

Next, we present results for the training strategy where each algorithm has access to $\rho$ update steps in total that are divided among all its models so that the computation time of each algorithm is identical. For the case $\rho = 4m$, the misclassification rate at each time step is shown in Figure 8, and the average over time is in Table 7. For the case $\rho = 2m$, the misclassification rate at each time is shown in Figure 9, and the average over time is in Table 8.

Let us discuss a few differences from the previous case where each model was trained with $\rho$ steps. Across every dataset, we observe lower accuracy for AUE, and especially so after drifts. This is because AUE is an ensemble of 10 models, and so each model is trained with only 1/5 of the steps that the models of DriftSurf get, and only 1/10 of the models for MDDM and Aware. DriftSurf now dominates AUE in average misclassification rate on each dataset except for PowerSupply; given AUE's good performance on PowerSupply in the divided $\rho$ case, we expect that DriftSurf's outperformance of AUE in the previous case in Table 6 to be due to random fluctuation.

We observe DriftSurf compares favorably to MDDM on the same datasets as it did in the undivided $\rho$ case. However, MDDM's advantages are magnified on SINE1 and RCV1, the datasets with sharp drifts that were clear to detect, and when immediate switching to the new model was desired. On PowerSupply, we observe that the false positives are not as punitive for MDDM as before, because its relative additional training per model means that its new models catch up faster. For Hyperplane, the relative additional training for MDDM was advantageous in the $\rho = 4m$ case, but in the $\rho = 2m$ case, the advantage of MDDM on Hyperplane-slow vanished and it was comparable to DriftSurf. We suspect that when fewer computational steps are available, it is no longer desirable to create new models (which take longer to warm up) so frequently as MDDM did in the $\rho = 4m$ case where it outperformed DriftSurf.

In Table 7, we also present results for a variation on AUE that is limited to only two experts, which we refer to as AUE ($k = 2$). In our comparison of each algorithm when enforcing equal computation time, dividing the $\rho$ steps equally among a total of ten experts in the original AUE is unsurprisingly detrimental to its performance. An alternative comparison is to reduce the total number of experts so that in AUE ($k = 2$), each of the two experts is updated with $\rho = 2m$ steps, identical to DriftSurf. We observe that AUE ($k = 2$) performs better than AUE on four datasets: Hyperplane-slow, Hyperplane-fast, SINE1, and Electricity. We previously mentioned that for Hyperplane, the continual drift means always using the latest

available model works well, and we mentioned that for Electricity, the drift that does not require adaptation means always using the oldest available model works well. Therefore, on these datasets, the additional eight experts of the original AUE have little utility and AUE ($k = 2$) performs better. The reason for improvement of AUE ($k = 2$) on SINE1 is less clear, but we suspect that the additional experts of the original AUE penalize the accuracy immediately after the abrupt drifts when it is desirable to assign the most weight to the newest expert.

*Table 7.* Total average of misclassification rate ($\rho = 4m$ divided among all models of each algorithm)

| DATASET | Aware | DriftSurf | MDDM-G | AUE | AUE ($k$=2) |
|---|---|---|---|---|---|
| SEA0 | 0.110 | **0.082** | 0.095 | 0.181 | 0.226 |
| SEA10 | 0.175 | 0.167 | **0.163** | 0.217 | 0.269 |
| SEA20 | 0.243 | **0.246** | 0.257 | 0.275 | 0.320 |
| SEA30 | 0.328 | **0.328** | 0.342 | 0.346 | 0.365 |
| HYPER-SLOW | 0.145 | 0.145 | 0.133 | 0.149 | **0.120** |
| HYPER-FAST | 0.222 | 0.173 | **0.154** | 0.234 | **0.154** |
| SINE1 | 0.155 | 0.188 | **0.16** | 0.269 | 0.18 |
| RCV1 | 0.109 | 0.131 | **0.112** | 0.305 | 0.404 |
| COVERTYPE | 0.26 | **0.264** | 0.3 | 0.301 | 0.314 |
| AIRLINE | 0.331 | **0.331** | 0.339 | 0.36 | 0.366 |
| ELECTRICITY | 0.305 | **0.322** | 0.325 | 0.351 | 0.326 |
| POWERSUPPLY | 0.307 | 0.307 | 0.288 | **0.286** | 0.393 |

*Table 8.* Total average of misclassification rate ($\rho = 2m$ divided among all models of each algorithm)

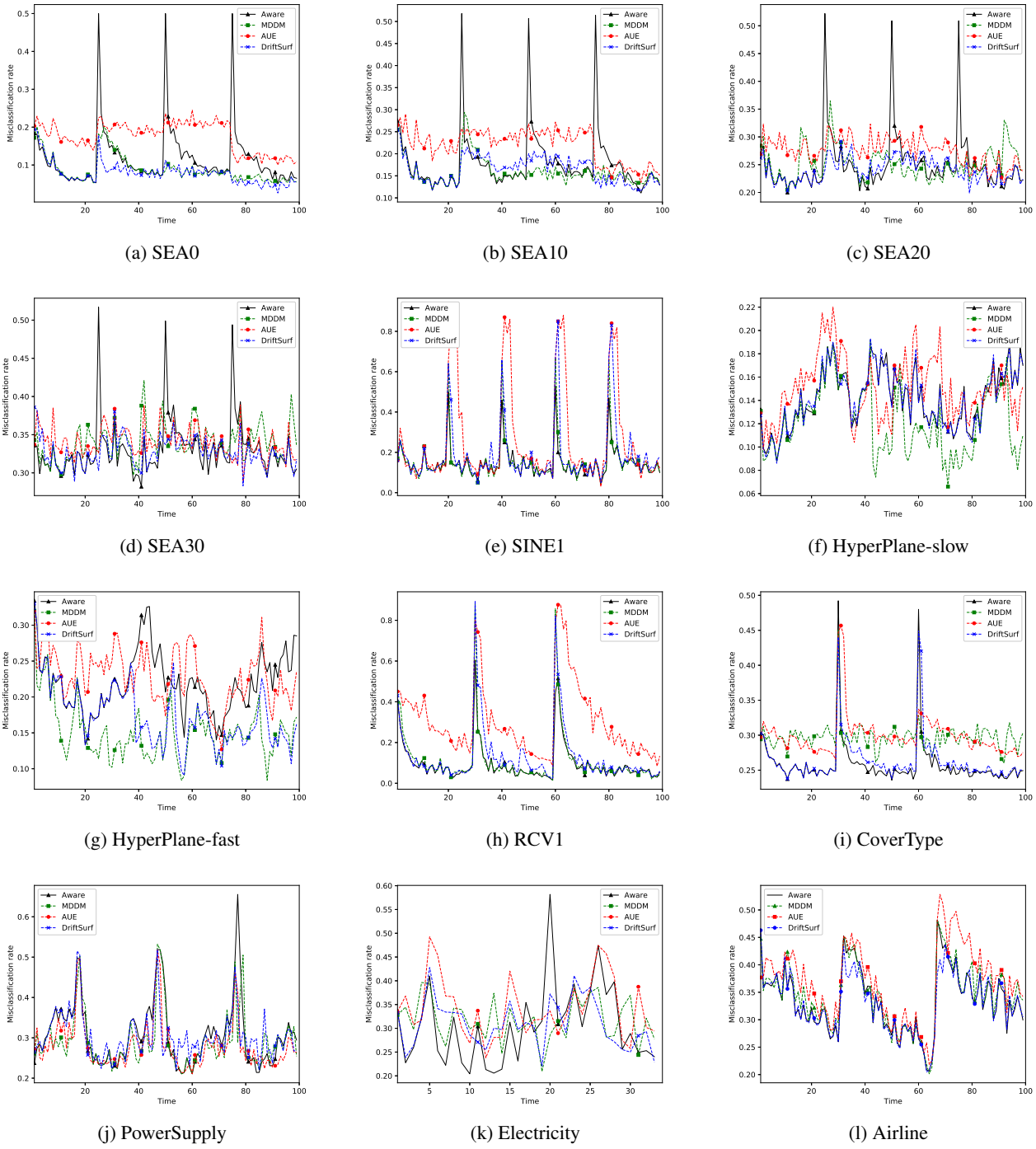
| DATASET | Aware | DriftSurf | MDDM-G | AUE |
|---|---|---|---|---|
| SEA0 | 0.13 | 0.096 | **0.089** | 0.204 |
| SEA10 | 0.183 | **0.157** | 0.182 | 0.234 |
| SEA20 | 0.257 | **0.244** | 0.275 | 0.288 |
| SEA30 | 0.343 | **0.335** | 0.366 | 0.35 |
| HYPER-SLOW | 0.117 | **0.117** | 0.118 | 0.189 |
| HYPER-FAST | 0.191 | 0.196 | **0.163** | 0.273 |
| SINE1 | 0.177 | 0.218 | **0.179** | 0.304 |
| RCV1 | 0.13 | 0.174 | **0.127** | 0.4 |
| COVERTYPE | 0.266 | **0.274** | 0.311 | 0.316 |
| AIRLINE | 0.336 | **0.353** | **0.353** | 0.372 |
| ELECTRICITY | 0.308 | **0.339** | **0.339** | 0.367 |
| POWERSUPPLY | 0.313 | 0.316 | 0.310 | **0.307** |

### D.3. Comparison to 1PASS-SGD and Oblivious

Figure 10 shows the comparison to 1PASS-SGD and the oblivious algorithm (OBL) for the RCV1 and Electricity datasets at each time. The time average misclassification rate for each dataset are in Table 6. In the case of the large, abrupt drift in RCV1, we observe that 1PASS-SGD and especially oblivious have poor performance after drift. The oblivious algorithm continues to re-sample the data from the older distributions, and leads to a model with random, or worse than random, accuracy on the current distribution. Even for 1PASS-SGD, which only trains over data from the most recent time step, we observe its convergence rate is slow after a drift, where its previous training on the old data still hinders it. On the Electricity data with a more subtle drift, we observe that oblivious is actually the best performing algorithm, as discussed earlier, because data from all over time can be trained and fit by a single model. However, 1PASS-SGD still has lower accuracy because, as a single pass method, it uses only $m$ update steps at each time even when $\rho = 2m$ are available to the other algorithms, and also because SGD has a slower convergence rate than the variance-reduced method STRSAGA.

### D.4. Evaluation of Greedy Reactive State

One design choice in the DriftSurf algorithm is that during the reactive state, the predictive model follows a greedy approach—the choice of the predictive model at the current time is the model that had the better performance in the previous time step—and then at the end of the reactive state, the decision is made whether or not to use the reactive model going forward.




(a) SEA0 (b) SEA10 (c) SEA20

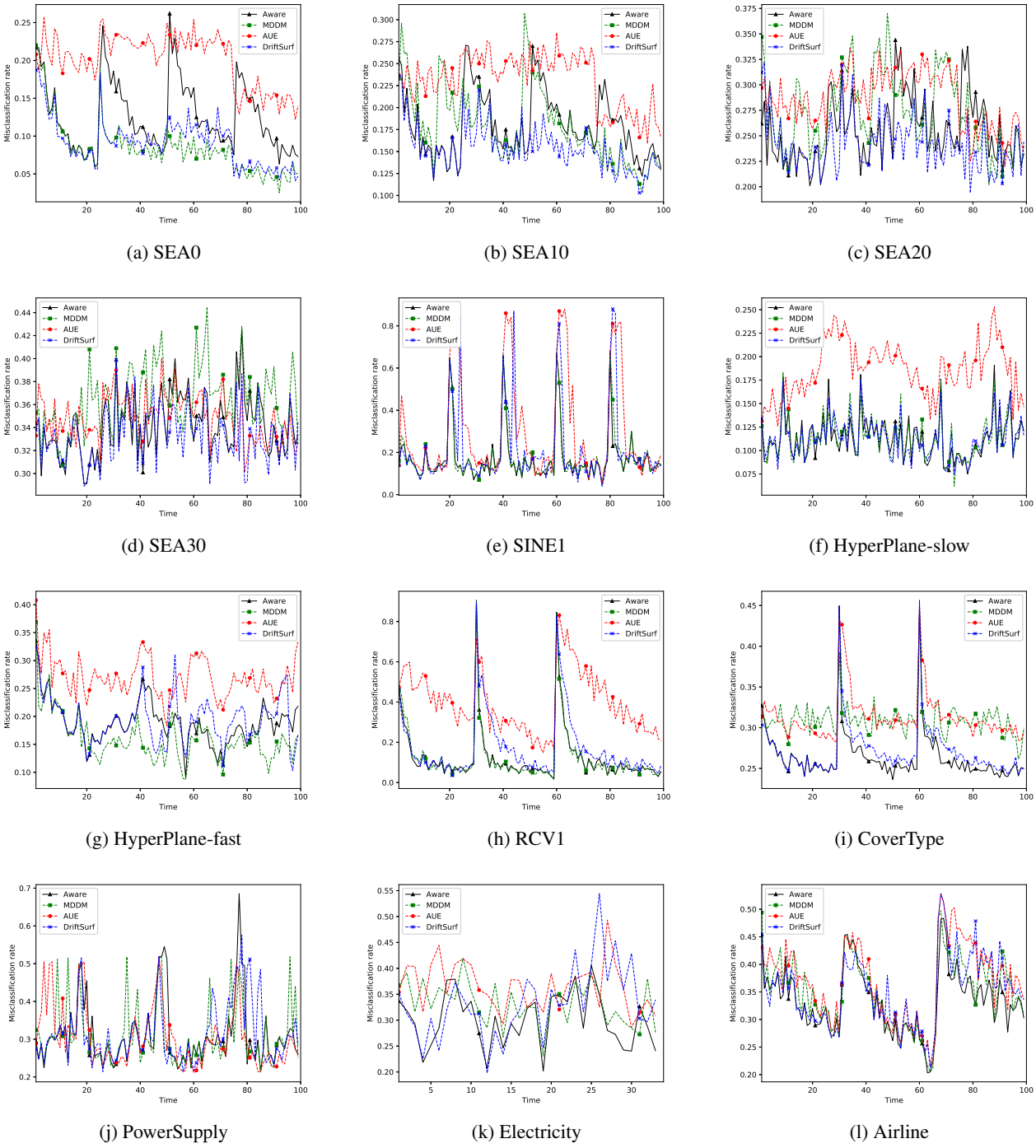
(d) SEA30 (e) SINE1 (f) HyperPlane-slow

(g) HyperPlane-fast (h) RCV1 (i) CoverType

(j) PowerSupply (k) Electricity (l) Airline

*Figure 8.* Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm)

The natural alternative choice is that switching to the new reactive model can happen only at the end of the reactive state, and the stable model is the predictive model throughout the reactive state; we call this DriftSurf (no-greedy). In Figure 11 and Table 9 we show the comparison of DriftSurf to DriftSurf (no-greedy). We observe that DriftSurf performs similar or better across each dataset, with the biggest improvements on SINE1 and RCV1, datasets that we earlier noted where MDDM and Aware perform well because there it is desirable to immediately switch to the new model after the large, abrupt drift. Figure 11 shows the delayed switch of DriftSurf (no-greedy) to the new model in the presence of drift until only the end of




(a) SEA0 (b) SEA10 (c) SEA20

(d) SEA30 (e) SINE1 (f) HyperPlane-slow

(g) HyperPlane-fast (h) RCV1 (i) CoverType

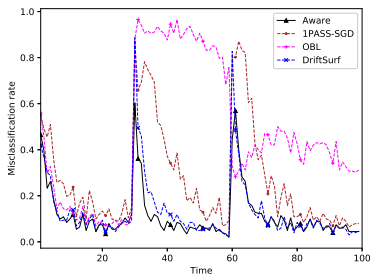
(j) PowerSupply (k) Electricity (l) Airline

*Figure 9.* Misclassification rate over time ($\rho = 2m$ divided among all models of each algorithm)

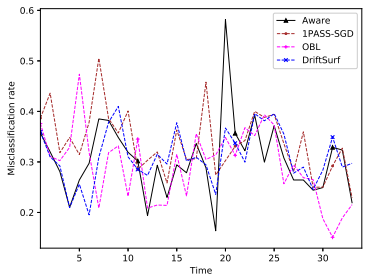
the reactive state.

### D.5. Using SGD as the Update Process

As mentioned earlier we choose STRSAGA as the update process because of two main reasons: (i) STRSAGA is designed in a way that can handle different arrival distributions, and (ii) it achieves a faster convergence rate because of using




(a) RCV1




(b) Electricity

*Figure 10.* Misclassification rate over time ($\rho = 2m$ for each model)

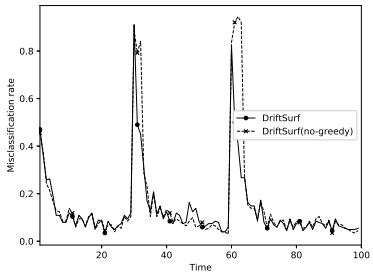



*Figure 11.* Misclassification rate over time for RCV1 - DriftSurf vs DriftSurf (no-greedy) ($\rho = 2m$ for each model)

*Table 9.* Total average of misclassification rate - DriftSurf vs DriftSurf (no-greedy) ($\rho = 2m$ for each model)

| DATASET | DriftSurf | DriftSurf (NO-GREEDY) |
|---|---|---|
| SEA0 | 0.088 | 0.092 |
| SEA10 | 0.158 | 0.159 |
| SEA20 | 0.245 | 0.243 |
| SEA30 | 0.335 | 0.338 |
| HYPER-SLOW | 0.117 | 0.118 |
| HYPER-FAST | 0.173 | 0.175 |
| SINE1 | 0.189 | 0.237 |
| RCV1 | 0.126 | 0.157 |
| COVERTYPE | 0.267 | 0.273 |
| AIRLINE | 0.334 | 0.335 |
| ELECTRICITY | 0.315 | 0.325 |
| POWERSUPPLY | 0.294 | 0.305 |

variance-reduced update step. In the next experiment, we want to study the impact of the choice of the update process in the final results. We re-run the previous experiments using SGD instead of STRSAGA. Table 10 shows the average misclassification rate for the case where $\rho = 2m$ update steps are used for each model, and Table 11 is for the case where $\rho = 2m$ steps are used by each algorithm and divided among its models.

As the results presented in Table 10 suggest, AUE, unlike the previous experiment, outperforms MDDM and DriftSurf for the majority of the studied datasets. The reason is hidden in the high variance nature of SGD. MDDM and DriftSurf both use performance-degradation for drift detection. Such drift detection is sensitive to the high variance during the training which may be mistaken for drift in the underlying distribution. However, comparing the results of DriftSurf and MDDM shows the advantage of going though a reactive state before restarting the model in reducing the false positive rate of drift detection. AUE, on the other hand, overcomes the high variance of SGD by using a bag of experts and making ensemble based decisions.

Similar to the previous experiments, to examine the accuracy achieved when enforcing equal processing time, we repeated the experiment for the case where $\rho = 2m$ steps are used by each algorithm and divided among its models. Reported results in Table 11 suggest that in such condition AUE is not able to overcome the high variance problem of SGD because it does not have enough resources to train all of its experts to the converging point.

STRSAGA because of its variance-reduced update step achieves a faster convergence rate in comparison to SGD. Difference between the reported results in Table 2 and Table 10 confirms the advantage of using STRSAGA over SGD as the update process.

*Table 10.* Total average of misclassification rate - update process: SGD ($\rho = 2m$ for each model)

| DATASET | Aware | DriftSurf | MDDM-G | AUE |
|---|---|---|---|---|
| SEA0 | 0.169 | **0.119** | 0.127 | 0.123 |
| SEA10 | 0.212 | **0.187** | 0.214 | **0.187** |
| SEA20 | 0.291 | 0.263 | 0.293 | **0.262** |
| SEA30 | 0.362 | 0.358 | 0.372 | **0.341** |
| HYPER-SLOW | 0.167 | 0.144 | 0.146 | **0.125** |
| HYPER-FAST | 0.272 | 0.196 | **0.174** | 0.196 |
| SINE1 | 0.192 | 0.235 | **0.205** | 0.242 |
| RCV1 | 0.147 | 0.171 | **0.157** | 0.206 |
| COVERTYPE | 0.274 | **0.274** | 0.326 | 0.287 |
| AIRLINE | 0.358 | 0.353 | 0.359 | **0.344** |
| ELECTRICITY | 0.335 | 0.314 | 0.348 | **0.299** |
| POWERSUPPLY | 0.355 | 0.319 | 0.375 | **0.299** |

*Table 11.* Total average of misclassification rate - update process: SGD ($\rho = 2m$ divided among all models of each algorithm)

| DATASET | Aware | DriftSurf | MDDM-G | AUE |
|---|---|---|---|---|
| SEA0 | 0.172 | 0.124 | **0.121** | 0.192 |
| SEA10 | 0.225 | **0.187** | 0.231 | 0.248 |
| SEA20 | 0.296 | **0.277** | 0.302 | 0.297 |
| SEA30 | 0.365 | **0.352** | 0.366 | 0.353 |
| HYPER-SLOW | 0.164 | 0.153 | **0.147** | 0.160 |
| HYPER-FAST | 0.270 | 0.215 | **0.177** | 0.272 |
| SINE1 | 0.190 | 0.237 | **0.204** | 0.307 |
| RCV1 | 0.147 | 0.209 | **0.160** | 0.438 |
| COVERTYPE | 0.274 | **0.283** | 0.324 | 0.315 |
| AIRLINE | 0.353 | **0.363** | 0.366 | 0.370 |
| ELECTRICITY | 0.329 | **0.353** | 0.354 | **0.353** |
| POWERSUPPLY | 0.342 | **0.318** | 0.401 | 0.323 |