DriftSurf: Stable-State / Reactive-State Learning 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 (i) statistically better than standalone drift detection and (ii) 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 several key 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 discarding 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

[*]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 Inc, Cupertino, California, USA. Correspondence to: Ashraf Tahmasbi <tahmasbi@iastate.edu>, Ellango Jothimurugesan <ejothimu@cs.cmu.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

drift when the model misclassifies data points drawn from a previously unobserved region of the feature space, but the 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 helps overcome these drift detection challenges. DriftSurf works by incorporating drift detection into a broader two-state process. The algorithm starts with a single model beginning in the *stable state* and transitions to the *reactive state* based on a drift detection trigger, and then starts a second 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 end of the reactive state, the algorithm transitions back to the stable state, keeping the model that was the best performer during the reactive state. DriftSurf's primary advantage over standalone drift detection is that 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—indeed, our theoretical analysis shows that DriftSurf is statistically better than standalone drift detection. Other advantages include (i) when restarting with a new model does not lead to better post-drift performance, the original model will continue to be used; and (ii) switching to the new model for predictions happens only when it begins outperforming the old model, accounting for potentially lower accuracy of the new model as it warms up. Meanwhile, the addition of this stable-state/reactive-state process does not unduly delay the time to recover from a drift, 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 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 10 synthetic and real-world 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 (Hentschel et al., 2019; Klinkenberg, 2004) 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 §1, or detecting change in the underlying data distribution which include tests given by (Kifer et al., 2004; Sebastião & Gama, 2007); the connection between the two approaches is made in (Hinder et al., 2020). 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. Tests for drift detection may also be combined, known as hierarchical change detection (Alippi et al., 2016), in which a slow but accurate second test is used to validate change detected by the first test. The two-state process of DriftSurf has a similar pattern, but differs in that DriftSurf's reactive state is based on the performance of a newly created model, which has the advantage of not prolonging the time to recover from a drift because the new model is available to use immediately.

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), DTEL (Sun et al., 2018), Diversity Pool (Chiu & Minku, 2018), and Condor (Zhao et al., 2020). 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 after drifts. 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 at most 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 $t = 1, 2, \ldots$, let $\mathbf{X}_t$ be the set of labeled 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 A with high prediction accuracy at each time step. At time t, A has access to all the data points so far, $\mathcal{S}_{1,t}$, and a constant number of processing steps (e.g., gradient computations) to output a model $\mathbf{w}_t$ from a class of functions $\mathcal{F}$ that map an unlabeled data point to a predicted label. Note this setting differs from the traditional online learning setting, as we are not limited in memory and allow for the reuse of relevant older data points in the stationary case to achieve higher accuracy than what can be achieved in a single pass.

To achieve high prediction accuracy at time t, we want to minimize the expected risk over the distribution I_t. The *expected risk* of function $\mathbf{w}$ over a distribution I is: $\mathcal{R}_I(\mathbf{w}) = \mathbb{E}_{\mathbf{x}\sim I}[f_{\mathbf{x}}(\mathbf{w})]$, where $f_{\mathbf{x}}(\mathbf{w})$ is the loss of function $\mathbf{w}$ on input $\mathbf{x}$. Thus, the objective at each time t is:

$$\min_{\mathbf{w}_t \in \mathcal{F}} \mathbb{E}_{\mathbf{x}\sim I_t}[f_{\mathbf{x}}(\mathbf{w}_t)]$$

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 $\mathcal{S}_{t_1,t_2}$. The *empirical risk* of function $\mathbf{w}$ over a sample $\mathcal{S}$ of n elements is: $\mathcal{R}_{\mathcal{S}}(\mathbf{w}) = \frac{1}{n}\sum_{\mathbf{x}\in\mathcal{S}} f_{\mathbf{x}}(\mathbf{w})$. 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 $\mathcal{R}^*_{\mathcal{S}} = \mathcal{R}_{\mathcal{S}}(\mathbf{w}^*_{\mathcal{S}})$.

Table 1: Commonly used symbols

$\mathbf{X}_t$	data points arriving at time step t
m	$= \|\mathbf{X}_t\|$, number of points arriving at each time
$\mathcal{R}_{\mathcal{S}}$	empirical risk over the set of points $\mathcal{S}$
$\mathcal{H}$	statistical error bound $\mathcal{H}(n) = hn^{-\alpha}$
h	constant factor in the statistical error bound
α	exponent in the statistical error bound
W	length of the windows $W1$ and $W2$
r	length of the reactive state
δ	threshold in condition 2 to enter the reactive state
δ'	threshold in condition 3 to switch the model
Δ	magnitude of a drift

In order to quantify the error in the expected risk from empirical risk minimization, we use a uniform convergence bound (Boucheron et al., 2005; Bousquet & Bottou, 2007). We assume the expected risk over a distribution I and the empirical risk over a sample $\mathcal{S}$ of size n drawn from I are related through the following bound:

$$\mathbb{E}[\sup_{\mathbf{w}\in\mathcal{F}} |\mathcal{R}_I(\mathbf{w}) - \mathcal{R}_{\mathcal{S}}(\mathbf{w})|] \leq \mathcal{H}(n)/2 \qquad (1)$$

where $\mathcal{H}(n) = hn^{-\alpha}$, for a constant h and $1/2 \leq \alpha \leq 1$. From this relation, $\mathcal{H}(n)$ is an upper bound on the statistical error (also known as the estimation error) over a sample of size n (Bousquet & Bottou, 2007).

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), in the stationary case, 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 $\mathcal{F}$.

However, suppose a concept drift occurs at time t_d such that $t_1 < t_d < t_2$. 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: The **recovery time** of an algorithm A is the time it takes after a drift for A to provide a solution $\mathbf{w}$ that 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 $t_{d_0} = 1$. 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, recovered periods between drifts). In §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.

Table 1 summarizes the symbols commonly used throughout the rest of the paper.

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 the corresponding 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 sub-optimality of the model trained over $\mathcal{S}$ suffers from the limited size of $\mathcal{S}$. Using larger window sizes helps with achieving a better sub-optimality, but increases the recovery time. Smaller window sizes, on the other hand, provide better recovery time, but the 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 §1, prior work (Baena-García et al., 2006; Bifet & Gavaldà, 2007; Gama et al., 2004; 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 higher confidence decision can be made.

Our algorithm, DriftSurf, is depicted in Algorithm 1, which is executed when DriftSurf is in the stable state, and Algorithm 2, which is executed when DriftSurf is in the reactive

Algorithm 1 DriftSurf-Stable-State: Processing a set of training points $\mathbf{X}_t$ arriving in time step t during a stable state

// $\mathbf{w}_{t-1}(\mathcal{S})$, $\mathbf{w}'_{t-1}(\mathcal{S}')$ are respectively the parameters
// (stream segments for training) of the predictive, and
// reactive models. Every W time steps starting with
// the creation of the current predictive model, we start
// a new "window" of size W.
// $\mathbf{w}_{b1}$, $\mathbf{w}_{b2}$ are the models with the best observed risk
// $\mathcal{R}_{b1}$, $\mathcal{R}_{b2}$ in the two most-recent windows $W1$, $W2$.
if condition 2 holds **then** {Enter reactive state}
  state $\leftarrow$ reactive
  $T \leftarrow \emptyset$ {T is a segment arriving during the last $r/2$ time steps of reactive state}
  $\mathbf{w}'_{t-1} \leftarrow \mathbf{w}_0, \mathcal{S}' \leftarrow \emptyset$ {initialize randomly a new reactive model}
  $i \leftarrow 0$ {time steps in the current reactive state}
  execute Algorithm 2 on $\mathbf{X}_t$
else
  $\mathbf{w}_t \leftarrow$ Update$(\mathbf{w}_{t-1}, \mathcal{S}, \mathbf{X}_t)$ {update $\mathbf{w}, \mathcal{S}$}
end if

Algorithm 2 DriftSurf-Reactive-State: Processing a set of training points $\mathbf{X}_t$ arriving in time step t during a reactive state

// $\mathbf{w}_{t-1}$, $\mathcal{S}$, $\mathbf{w}'_{t-1}$, $\mathcal{S}'$, $\mathbf{w}_{b1}$, $\mathbf{w}_{b2}$, $\mathcal{R}_{b1}$, $\mathcal{R}_{b2}$ are as defined
// in Algorithm 1, except that $W1$, $W2$ are the two most-
// recent windows started *before* the current reactive state.
if condition 2 does NOT hold **then** {Early exit}
  state $\leftarrow$ stable
  execute Algorithm 1 on $\mathbf{X}_t$
else
  $i \leftarrow i + 1$
  $\mathbf{w}_t \leftarrow$ Update$(\mathbf{w}_{t-1}, \mathcal{S}, \mathbf{X}_t)$ {update $\mathbf{w}, \mathcal{S}$}
  $\mathbf{w}'_t \leftarrow$ Update$(\mathbf{w}'_{t-1}, \mathcal{S}', \mathbf{X}_t)$ {update $\mathbf{w}', \mathcal{S}'$}
  if $i = \frac{r}{2}$ **then**
    $\mathbf{w}'_f \leftarrow \mathbf{w}'_{t-1}$ {take a snapshot of reactive model}
  else if $\frac{r}{2} < i \leq r$ **then**
    add $\mathbf{X}_t$ to T
  end if
  if $i = r$ **then** {Exit reactive state}
    state $\leftarrow$ stable
    if condition 3 holds **then**
      $\mathbf{w}_t \leftarrow \mathbf{w}'_t, \mathcal{S} \leftarrow \mathcal{S}'$ {change the predictive model}
    end if
  else if $\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}'_t) < \mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_t)$ **then**
    use $\mathbf{w}'_t$ instead of $\mathbf{w}_t$ for predictions at the next time step {greedy policy}
  end if
end if

state. The algorithm starts in the stable state, and the steps are shown for processing the batch of points arriving at time step t. When in the stable state, there is a single model, $\mathbf{w}_{t-1}$, called the *predictive* model. Our test for entering the reactive state is based on dividing the time steps since the creation of that model into windows of size W. DriftSurf enters the reactive state at the sign of a drift, given by the following condition:

$$\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_b) > \mathcal{R}_b + \delta, \text{where } b = \arg\min_{b \in b1,b2} \mathcal{R}_b \quad (2)$$

and δ is a predetermined threshold that represents the tolerance in performance degradation (the selection of δ is discussed in §6), and $\mathbf{w}_{b1}$ ($\mathbf{w}_{b2}$) are the parameters of the predictive model that provided the best-observed risk $\mathcal{R}_{b1}$ ($\mathcal{R}_{b2}$) over the most-recent window $W1$ (second most-recent window $W2$). E.g., $\mathcal{R}_{b1} = \mathcal{R}_{\mathbf{X}_{b1+1}}(\mathbf{w}_{b1}) = \min_{j \in W1} \mathcal{R}_{\mathbf{X}_j}(\mathbf{w}_{j-1})$. Although most drift detection techniques rely on their predictive model to detect a drift, we keep a snapshot of the predictive model that provided the best-observed risk over two jumping windows of up to W time steps because: (i) having a frozen model that does not train over the most recent data increases the chance of detecting slow, gradual drifts; (ii) each frozen model is at most $2W$ time steps old which makes it reflective of the current predictive model; and (iii) the older of the models reflects the best over W steps, while the younger of the models is guaranteed to have at least W steps that it can be used for drift detection tests, which are both key factors in obtaining our theoretical analysis.

If condition 2 does not hold, DriftSurf assumes there was no drift in the underlying distribution and remains in the stable state. It calls Update, an *update process* that expands $\mathcal{S}$ to include the newly arrived set of data points $\mathbf{X}_t$ and then updates the (predictive) model parameters using $\mathcal{S}$ for incremental training (examples in Appendix A). Otherwise, DriftSurf enters the reactive state, adds a new model $\mathbf{w}'_{t-1}$, called the *reactive model*, with randomly initialized parameters, and initializes its sample set $\mathcal{S}'$ to be empty. To save space, the growing sample set $\mathcal{S}'$ 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 entered the reactive state) (Algorithm 2), DriftSurf checks that condition 2 still holds (to handle a corner case discussed below), adds $\mathbf{X}_t$ to $\mathcal{S}$ and $\mathcal{S}'$, the sample sets of the predictive and reactive models, and updates $\mathbf{w}_{t-1}$ and $\mathbf{w}'_{t-1}$. During the reactive state, DriftSurf uses for prediction at t whichever model $\mathbf{w}$ or $\mathbf{w}'$ performed the best in the previous time step $t-1$. This greedy heuristic yields better performance during the reactive state by switching to the newly added model sooner in the presence of drift.

Upon exiting the reactive state (when i=r), DriftSurf chooses the predictive model to use for the subsequent stable state. It switches to the reactive model $\mathbf{w}'$ if condition 3 holds:

$$\mathcal{R}_T(\mathbf{w}'_f) < \mathcal{R}_T(\mathbf{w}_b) - \delta', \text{where } b = \arg\min_{b \in b1,b2} \mathcal{R}_b \quad (3)$$

and $\mathbf{w}'_f$ is the snapshot of reactive model (at $i = r/2$), $\mathbf{w}_b$ is snapshot of the predictive model with the best-observed performance over the last two windows and δ' is set to be much smaller than δ (our experiments use $\delta' = \delta/2$). This condition checks their performance over the test set of data points T that arrived during the last $r/2$ time steps of the reactive state (note that neither $\mathbf{w}'_f$ nor $\mathbf{w}_b$ have been trained over this test set). This provides an unbiased test to decide on switching the model. Otherwise, DriftSurf continues with the prior predictive model.

Handling a corner case. Consider the case that a drift happens when DriftSurf is in the reactive state (due to an earlier false positive on entering the reactive state). In this case, no matter what predictive model DriftSurf chooses 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 keeps checking condition 2 and drops out of the reactive state if it fails to hold (because the failure indicates a false positive). Then the next time the condition holds, a fresh reactive state is started. This way the new reactive model will be trained solely on the new distribution.

Algorithm 1 and 2 are generic in the individual base learner. For the experimental evaluation in §6, we focus on base learners where the update process is STRSAGA (Jothimurugesan et al., 2018), a variance-reduced SGD for streaming data. Compared to SGD, STRSAGA has a faster convergence rate and better performance under different arrival distributions. The time and space complexity of DriftSurf is within a constant factor of the individual base learner.

5. Analysis of DriftSurf

In this section, we show that DriftSurf achieves goals **G1** and **G2** from §3. 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 n data points all 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.

We assume that Aware and DriftSurf use base learners that

efficiently learn to within statistical accuracy:

Assumption 1. *Let t_0 be the time the base learner B was initialized. At each time step t,*

$$\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_0,t}}(B)] \leq \mathcal{H}(n_{t_0,t}).$$

As an example, a base learner that uses STRSAGA as the update process satisfies Assumption 1 by Lemma 3 in (Jothimurugesan et al., 2018). We use STRSAGA in the bulk of our experimental evaluation.

As a means of achieving goal **G2** (sub-optimality 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 1.

Definition 1. *For $c \geq 1$, an adaptive learning algorithm A is said to be c-risk-competitive to* Aware *at time step $t > t_d$ if $\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(A)] \leq c\mathcal{H}(n_{t_d,t})$, where t_d is the time step of the most recent drift and $n_{t_d,t} = |\mathcal{S}_{t_d,t}|$.*

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

Let $t_{d_1}, t_{d_2}, \ldots$ be the sequence of time steps at which a drift occurs. We assume that each drift at t_{d_i} is abrupt and that it satisfies the following assumption of sustained performance-degradation.

Assumption 2. *For the drift at time t_{d_i}, and for both frozen models $\mathbf{w}_b \in \{\mathbf{w}_{b1}, \mathbf{w}_{b2}\}$ stored at t_{d_i}, we have $\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) > \mathcal{R}_b$ for each time $t_{d_i} < t < t_{d_{i+1}}$ as long as* DriftSurf *has not recovered. Furthermore, we denote Δ to be the magnitude of the drift where $\Delta = \min_{\mathbf{w}_b}(\mathcal{R}_J(\mathbf{w}_b) - \mathcal{R}_I(\mathbf{w}_b))$ where I denotes the distribution at the time $t_{d_i} - 1$ before the drift, and J denotes the distribution at t_{d_i}.*

Typically in drift detection, the magnitude of a drift is defined as the difference in the expected risks over the old and new distributions with respect to the current predictive model. But that definition results in a moving target after the drift but before replacement of the model, as the model gets updated with new data, and possibly slowly converges on the new distribution, making the drift harder to detect. Instead in our approach in DriftSurf, detection is done on frozen models snapshotted prior to the drift, and we accordingly define the drift magnitude with respect to the frozen models. The implication of Assumption 2 is that after a drift, the current predictive model being continually updated with the new data does not automatically adapt to the drift for at least W time steps and actually needs to be replaced.

Finally, we assume that all loss functions $f_{\mathbf{x}}$ are bounded $[0, 1]$, that the optimal expected risk $R^*_{I_t} = \inf_{\mathbf{w}\in\mathcal{F}} \mathcal{R}_{I_t}(\mathbf{w}) = 0$ for each distribution I_t, that the batch size $m > 16/\delta'$, that each drift magnitude $\Delta > \delta$, that $2W$ is upper bounded by both $\exp(\frac{1}{2}m\delta^2)$ and $\exp(\frac{1}{2}m(\Delta - \delta)^2)$ for each drift magnitude Δ, and that for each frozen model $\mathbf{w}_b$ that yielded a minimal observed risk $\mathcal{R}_b$, that its expected risk is at least as good as its expectation.

5.1. Stationary Environment

We will show that DriftSurf is competitive to Aware in the stationary environment during the time $1 < t < t_{d_1}$ before any drift happens. By Assumption 1 the expected sub-optimality of Aware and DriftSurf are (respectively) bounded by $\mathcal{H}(n_{1,t})$ and $\mathcal{H}(n_{t_e,t})$, 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 $n_{t_e,t}$, the size of the predictive model's sample set, is close to $n_{1,t}$. To achieve this, we first give a constant upper bound p_s on the probability of entering the reactive state:

Lemma 1. *In the stationary environment for $1 < t < t_{d_1}$, the probability of entering the reactive state is upper bounded by $p_s = 2\exp(-\frac{1}{8}m\delta^2)$.*

In the proof (Appendix B.1), we use sub-Gaussian concentration in the empirical risk under a bounded loss function.

Besides, if DriftSurf enters the reactive state in the stationary case, Lemma 2 asymptotically bounds the probability of switching to the reactive model by $q_s(\beta)$ to approach 0, where β is the age of the frozen model $\mathbf{w}_b$ used in condition 3.

Lemma 2. *In the stationary environment for $1 < t < t_{d_1}$, if* DriftSurf *enters the reactive state, the probability of switching to the reactive model at the end of the reactive state is bounded by $q_s = c_1/\beta^2$ for $\beta > c_2$, where β is the number of time steps between the initialization of the model $\mathbf{w}_b$ and the time it was frozen, and the constants $c_1 = (2h/m^\alpha)^{mr\delta'/4}$ and $c_2 = \frac{1}{m}(2h/\delta')^{1/\alpha}$.*

In the proof (Appendix B.1), we use the convergence of the base learner and Bennett's inequality.

As the probability of falsely switching to the reactive model goes to 0, DriftSurf is increasingly likely to hold onto the predictive model. Using the above results, we bound the size of the predictive model's sample set to at least half of the size of Aware's sample set, with high probability.

Corollary 1. *With probability $1 - \epsilon$, the size of the sample set $\mathcal{S}$ for the predictive model in the stable state is larger than $\frac{1}{2}n_{1,t}$ at any time step $2W + c_4/(\epsilon - c_3) \leq t < t_{d_1}$, where $n_{1,t}$ is the total number of data points that arrived until time t, and constants $c_3 = c_1((c_2 + W) - 1/c_2)p_s$ and $c_4 = (2c_3 - 8)c_1^2p_s^2 + 6c_1p_s$ (where c_1 and c_2 are the constants in Lemma 2).*

Based on the result of Corollary 1, we show that the predictive model of DriftSurf in the stable state is $\frac{7}{4^{1-\alpha}}$-risk-competitive with Aware with probability $1-\epsilon$, at any time step $2W + c_4/(\epsilon - c_3) \leq t < t_{d_1}$. This is a special case of the forthcoming Theorem 1 in §5.2.

In addition, it follows from Lemma 1 and Corollary 1 that DriftSurf maintains an asymptotically larger expected number of samples compared to the standalone drift detection algorithm that resets the model whenever condition 2 holds (this algorithm is DriftSurf without the reactive state).

Lemma 3. *In the stationary environment for $1 < t < t_{d_1}$, let β be the age of the predictive model in* DriftSurf *and let γ be the age of the model of standalone drift detection. For $(2W + \frac{2c_4}{1-2c_3}) < t < t_{d_1}, \mathbb{E}[\beta] > t/4$ (where c_3 and c_4 are the constants in Corollary 1). Meanwhile, even as $t \to \infty$ (in the absence of drifts), $\mathbb{E}[\gamma] > 1/p_s - o(1)$.*

When each model is trained to statistical accuracy (Assumption 1), the total (statistical+optimization) error bound is asymptotically limited by the statistical error for the number of samples maintained. Hence, DriftSurf is statistically better than standalone drift detection in a stationary environment.

5.2. In Presence of Abrupt Drifts

Consider an abrupt drift that occurs at time t_{d_i}, and let Δ be its magnitude. Suppose the drift occurs while DriftSurf is in the stable state. The case of drift occurring when DriftSurf is in the reactive state is handled in Appendix B.2. We show that DriftSurf has a bounded recovery time (goal **G1**). In order to do so, we first give a lower bound p_d on the probability of entering the reactive state:

Lemma 4. *For $t_{d_i} < t < W$, the probability of entering the reactive state while* DriftSurf *has not yet recovered is lower bounded by $p_d = 1 - 2\exp(-(\frac{1}{8}m(\Delta - \delta)^2)$.*

Next, we give a lower bound q_d on the probability of switching to the reactive model at the end of the reactive state:

Lemma 5. *For $t_{d_i} < t < W$, the probability of switching to the reactive model at the end of the reactive state while* DriftSurf *has not yet recovered is lower bounded by $q_d = 1 - 2\exp(-C^2)$ where $C = (\Delta - \delta')\sqrt{mr}/2 - 2^{\alpha+1}h/(mr)^{\alpha-1/2}$ subject to $C > 0$.*

The proofs of the preceding two lemmas are similar to their stationary counterparts due to the use of frozen models: for the W time steps after the drift, by Assumption 2, the previous frozen models will not be displaced by a newer model that has been partially trained over data after the drift.

Following from Lemmas 4 and 5, the recovery time of DriftSurf is bounded by W with a probability $1 - \epsilon_r$ where ϵ_r is parameterized by p_d, q_d, which is shown in Lemma 11 in Appendix B.2.

We next show the risk-competitiveness of DriftSurf after recovery (goal **G2**). The time period after recovery until the next drift is a stationary environment for DriftSurf, in which each model is trained solely over points drawn from a single distribution, allowing for an analysis similar to the stationary environment before any drifts occurred.

Theorem 1. *With probability $1 - \epsilon$, the predictive model of* DriftSurf *in the stable state is $\frac{7}{4^{1-\alpha}}$-risk-competitive with* Aware *at any time step $t_{d_i} + 3W + c_4/(\epsilon_s - c_3) \leq t < t_{d_{i+1}}$, where t_{d_i} is the time step of the most recent drift and $\epsilon = \epsilon_s + \epsilon_r$ (where c_3, c_4 are the constants in Corollary 1).*

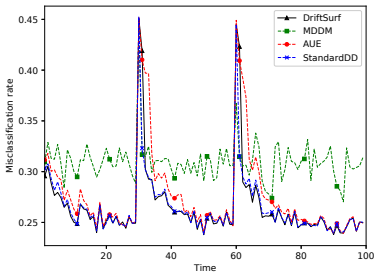
At a high level, ϵ_r and ϵ_s, respectively, capture the error rates in false negatives in drift detection and false positives in the stationary period afterwards. The full proof is in Appendix B.2.

6. Experimental Results

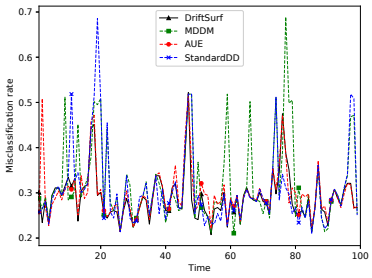
In this section, we present experimental results on datasets with drifts that (i) empirically confirm the advantage of DriftSurf's stable-state / reactive-state approach over Standard Drift Detection (StandardDD), (ii) empirically confirm the risk-competitiveness of DriftSurf with Aware, and (iii) show the effectiveness of DriftSurf via comparison to two state-of-the-art adaptive learning algorithms, the drift-detection-based method MDDM and the ensemble method AUE. Both StandardDD and MDDM are standalone drift detection algorithms, with the key difference being that StandardDD's drift detector matches the test used by DriftSurf to enter the reactive state, enabling us to quantify the gains of having a reactive state. More details on these algorithms, and additional algorithm comparisons, are provided in Appendix C.1.

We use five synthetic, two semi-synthetic and three real datasets for binary classification, chosen to include all such datasets that the authors of MDDM and AUE use in their evaluations. These datasets include both abrupt and gradual drifts. 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 that arrive over the course of the stream. More detail on the datasets is provided in Appendix C.2.

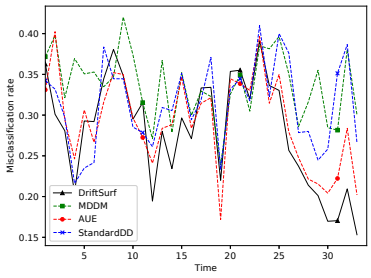
In our experiments, a batch of data points arrives at each time step. We first evaluate the performance of each algorithm by measuring the misclassification rate over this batch, and then each algorithm gains access to the labeled data to update their model(s); i.e., test-then-train. The base learner in each algorithm is a logistic regression model with STRSAGA as the update process. More details on this base learner, hyperparameter settings, and additional base learners, are provided in Appendix C.3. All reported results of



(a) CoverType



(b) PowerSupply



(c) Electricity

Figure 1: Misclassification rate over time for CoverType, PowerSupply, and Electricity

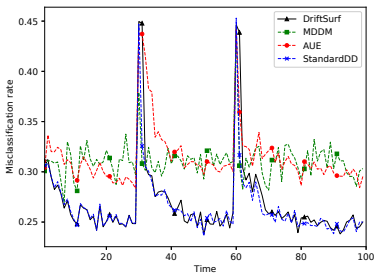


Figure 2: CoverType (update steps divided among each model)

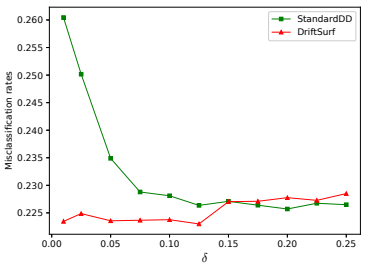


Figure 3: All datasets, DriftSurf and StandardDD under varying threshold δ

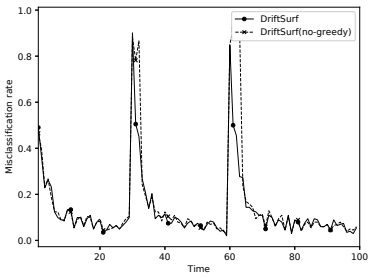


Figure 4: RCV1, DriftSurf and DriftSurf (no-greedy)

the misclassification rates represent the median over five trials.

We present the misclassification rates at each time step on the CoverType, PowerSupply, and Electricity datasets (see Appendix D.1 for other datasets) in Figure 1. A drift occurs at times 30 and 60 in CoverType, at times 17, 47, and 76 in PowerSupply, and at time 20 in Electricity. 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. CoverType and Electricity were especially problematic for MDDM, which continually signaled a drift. We also observe DriftSurf adapts faster than AUE on CoverType and Electricity. 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 have reduced, but non-zero weights that negatively impact the accuracy. In particular, on CoverType, we observe the recovery time of DriftSurf is within one reactive state.

StandardDD also suffers from false-positive drift detection, especially on PowerSupply and Electricity. However, it outperforms all the other algorithms on CoverType. It detects the drifts at the right moment and resets its predictive model. Following the greedy approach during the reactive state allows DriftSurf to converge to its newly created model with only a one time step lag.

Table 2: Average of misclassification rate (equal number of update steps for each model)

ALGORITHM / DATASET	AUE	MDDM	StandardDD	DriftSurf	Aware
SEA0	0.093	**0.086**	0.097	**0.086**	0.137
SEA20	0.245	0.289	0.249	**0.243**	0.264
SEA-GRADUAL	0.162	0.165	0.160	**0.159**	0.177
HYPER-SLOW	**0.112**	0.116	0.116	0.118	0.110
HYPER-FAST	0.179	**0.163**	0.168	0.173	0.191
SINE1	0.212	**0.176**	0.184	0.187	0.171
MIXED	0.209	**0.204**	**0.204**	**0.204**	0.192
CIRCLES	0.379	0.372	0.377	**0.371**	0.368
RCV1	0.167	**0.125**	0.126	**0.125**	0.121
COVERTYPE	0.279	0.311	**0.267**	0.268	0.267
AIRLINE	**0.333**	0.345	0.338	0.334	0.338
ELECTRICITY	0.296	0.344	0.320	**0.290**	0.315
POWERSUPPLY	0.301	0.322	0.308	**0.292**	0.309

Table 2 summarizes the results for all the datasets in terms of the total average of the misclassification rate over time. In the first two rows, we observe the stability of DriftSurf in the presence of 20% additive noise in the synthetic SEA dataset, again demonstrating the benefit of the reactive state while MDDM's performance suffers due to the increased false positives. We also observe that DriftSurf performs well on datasets with gradual drifts, such as SEA-gradual and Circles, where the stable-state / reactive-state approach is more

accurate at identifying when to switch the model, compared to MDDM and StandardDD, respectively. Overall, DriftSurf is the best performer on a majority of the datasets in Table 2. For some datasets (Airline, Hyper-Slow) AUE outperforms DriftSurf. A factor is the different computational power (e.g., number of gradient computations per time step) used by each algorithm. AUE maintains an ensemble of ten experts, while DriftSurf maintains just one (except during the reactive state when it maintains two), and so AUE uses at least five (up to ten) times the computation of DriftSurf. To account for the varying computational efficiency of each algorithm, we conducted another experiment where the available computational power for each algorithm is divided equally among all of its models. (A different variation on AUE that is instead limited by only maintaining two experts is also studied in Appendix D.2.) The misclassification rates for each dataset are presented in Table 3, where we observe DriftSurf dominates AUE across all datasets. The CoverType dataset is visualized in Figure 2 (compare to Figure 1a for equal computational power given to each model), where we observe a significant penalty to the accuracy of AUE because of the constrained training time per model.

Table 3: Average of misclassification rate (update steps divided among each model)

ALGORITHM / DATASET	AUE	MDDM	StandardDD	DriftSurf	Aware
SEA0	0.201	**0.089**	0.097	0.094	0.133
SEA20	0.291	0.283	0.253	**0.249**	0.266
SEA-GRADUAL	0.240	0.172	0.161	**0.160**	0.174
HYPER-SLOW	0.191	**0.116**	0.117	0.130	0.117
HYPER-FAST	0.278	**0.164**	0.168	0.188	0.191
SINE1	0.309	**0.178**	0.180	0.209	0.168
MIXED	0.259	**0.204**	**0.204**	**0.204**	0.191
CIRCLES	0.401	0.372	0.380	**0.369**	0.368
RCV1	0.403	0.131	**0.128**	0.143	0.120
COVERTYPE	0.317	0.313	**0.267**	0.271	0.267
AIRLINE	0.369	0.351	**0.338**	0.348	0.338
ELECTRICITY	0.364	0.339	0.319	**0.308**	0.311
POWERSUPPLY	0.313	0.309	0.311	**0.307**	0.311

Another advantage of the stable-state / reactive-state approach of DriftSurf is its robustness in the setting of the threshold δ. In general, drift detection tests have a threshold that poses a trade-off in false positive and false negative rates (for StandardDD, Lemmas 1 and 4 in §5), which can be difficult to tune without knowing the frequency and magnitude of drifts in advance. Across a range of δ, Figure 3 shows the misclassification rates for DriftSurf compared to StandardDD, averaged across the datasets in Table 2 (see Appendix D.3 for results per dataset). We observe that the performance of DriftSurf is resilient in the choice of δ. We also confirm that lower values of δ, corresponding to aggressive drift detection in the stable state, allow DriftSurf to detect subtle drifts while not sacrificing performance because the reactive state eliminates most false positives.

We also study the impact of the design choice in DriftSurf of using greedy prediction during the reactive state. While in the reactive state, the predictive model used at one time step 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. The natural alternative choice is that switching to the new reactive model can happen only at the end of the reactive state; we call this DriftSurf (no-greedy). The comparison of these two choices is visualized on the RCV1 dataset in Figure 4, where we observe the delayed switch of DriftSurf (no-greedy) to the new model following the drifts at times 30 and 60. The full results for each dataset are presented in Appendix D.4, where we observe that DriftSurf performs equal or better than DriftSurf (no-greedy) on 11 of the 13 datasets in Table 2, and, averaging over all the datasets, has a misclassification rate of 0.221 compared to 0.229.

Appendices D.5–D.8 contain additional experimental results. In Appendix D.5, 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. Appendix D.6 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. Appendix D.7 studies base learners beyond logistic regression, showing the advantage of DriftSurf's stable-state/reactive-state approach on both Hoeffding Trees and Naive Bayes classifiers. Finally, Appendix D.8 reports additional numerical results on the recovery time of each algorithm.

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. Further analysis showed that DriftSurf's reactive-state approach provides statistically better learning than standalone drift detection. Our experimental results confirmed our theoretical analysis and also showed high accuracy in the presence of abrupt and gradual drifts, generally outperforming state-of-the-art algorithms MDDM and AUE. Furthermore, DriftSurf maintains at most two models while achieving high accuracy, and therefore its computational efficiency is significantly better than an ensemble method like AUE.

Acknowledgments. This research was supported in part by NSF grants CCF-1725663 and CCF-1725702 and by the Parallel Data Lab (PDL) Consortium.

References

Alippi, C., Boracchi, G., and Roveri, M. Hierarchical change-detection tests. *IEEE Trans. Neural Netw. Learn. Syst*, 28(2):246–258, 2016.

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. and Gavaldà, R. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pp. 249–260, 2009.

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

Boucheron, S., Bousquet, O., and Lugosi, G. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 9:323–375, 2005.

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.

Chiu, C. W. and Minku, L. L. Diversity-based pool of models for dealing with recurring concepts. In *IJCNN*, pp. 1–8, 2018.

Daneshmand, H., Lucchi, A., and Hofmann, T. Starting small-learning with adaptive sample sizes. In *ICML*, pp. 1463–1471, 2016.

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.

Hinder, F., Artelt, A., and Hammer, B. Towards non-parametric drift detection via dynamic adapting window independence drift detection (dawidd). In *ICML*, pp. 4249–4259, 2020.

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.

Montiel, J., Read, J., Bifet, A., and Abdessalem, T. Scikit-multiflow: A multi-output streaming framework. *JMLR*, 19(72):1–5, 2018.

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.

Shaker, A. and Hüllermeier, E. Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. *Neurocomputing*, 150: 250–264, 2015.

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.

Zhao, P., Cai, L.-W., and Zhou, Z.-H. Handling concept drift via model reuse. *Machine Learning*, 109:533–568, 2020.

A. Pseudocode of STRSAGA

As shown in Figure 1 and Figure 2, DriftSurf calls a function Update($\mathbf{w}, \mathcal{S}, \mathbf{X}_t$) that takes a model $\mathbf{w}$, a sample set $\mathcal{S}$, and a set of training points $\mathbf{X}_t$. We let ρ be the computational power available at each time step; for an SGD-based algorithm, ρ is the number of gradients that can be computed in a time step. The Update function performs ρ updates to $\mathbf{w}$ and returns the resulting model; as a side effect, it also updates $\mathcal{S}$.

In this paper, we primarily use STRSAGA (Jothimurugesan et al., 2018), shown in Algorithm 4, for our Update function. STRSAGA differs from SGD (Algorithm 3) 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 SGD, given sample set $\mathcal{S}$ and newly arrived data points $\mathbf{X}_t$

// ρ is the computational power and determines the number of update steps that can be performed
// η is the learning rate
Add $\mathbf{X}_t$ to $\mathcal{S}$
for $j \leftarrow 1$ to ρ **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 $\mathbf{w}$

Algorithm 4 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 $\mathbf{X}_t$

// ρ is the computational power and determines the number of update steps that can be performed
// η is the learning rate
Add $\mathbf{X}_t$ to `WaitingRoom` {`WaitingRoom` is the set of training points not added to $\mathcal{S}$ yet}
for $j \leftarrow 1$ to ρ **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 $\mathbf{w}$

The time complexity of Algorithms 3 and 4 is on the order of ρ times the cost of a gradient computation with respect to a single data point. Each gradient computation is typically $O(d)$ for model parameter dimension d. The space complexity of Algorithm 3 is $O(D(|\mathcal{S}| + |\mathbf{X}_t|) + d)$ to store the samples and model parameters, where D is the dimension of the data. The space complexity of Algorithm 4 incurs an additive $O(d(|\mathcal{S}| + |\mathbf{X}_t|))$ to store the prior gradients $\alpha(p)$ for each data point (for linear models, this cost is reduced to $O(|\mathcal{S}| + |\mathbf{X}_t|)$ since each gradient is a scalar multiple of the corresponding

Table 4: 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)
W	length of the windows $W1$ and $W2$ (in time steps)
α	the exponent in the statistical error bound $\mathcal{H}(n) = hn^{-\alpha}$
h	the constant factor in the statistical error bound $\mathcal{H}(n) = hn^{-\alpha}$
δ	the threshold in condition 2 for entering the reactive state
δ'	the threshold in condition 3 for switching the model at the end of the reactive state
Δ	the magnitude of a given sustained performance-degrading drift
p_s	upper bound on the probability DriftSurf enters the reactive state in a stationary environment
p_d	lower bound on the probability DriftSurf enters the reactive state in the presence of drift
q_s	upper bound on the probability DriftSurf switches the model at the end of the reactive state in a stationary environment
q_d	lower bound on the probability DriftSurf switches the model at the end of the reactive state in the presence of drift

data point).

B. Proofs from the Analysis of DriftSurf

This section contains proof details from the analysis of DriftSurf (§5). As noted in §5, we make the following assumptions throughout our analysis. We 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. Next, the base learner reduces the sub-optimality over its sample set to within the statistical error bound (Assumption 1, which is repeated below).

Assumption 1. *Let t_0 be the time the base learner B was initialized. At each time step t,*

$$\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_0,t}}(B)] \leq \mathcal{H}(n_{t_0,t}).$$

Let $t_{d_1}, t_{d_2}, \ldots$ denote the sequence of time steps at which a drift occurs. We assume that each drift at t_{d_i} is abrupt (i.e., the distribution changes across a single time step)—this is solely for the analysis, as our algorithm more generally applies to gradual drifts, as our experimental results show. Further, we assume for the analysis that each drift induces sustained performance-degradation in DriftSurf (Assumption 2, which is repeated below). Note if there is only one stored frozen model $\mathbf{w}_{b1}$ associated with the predictive model at the time of the drift (because the second window has not started yet), then Assumption 2 applies only to $\mathbf{w}_{b1}$. Also note each drift magnitude Δ for the drift at t_{d_i} may be distinct, and the dependence on i is suppressed notationally.

Assumption 2. *For the drift at time t_{d_i}, and for both frozen models $\mathbf{w}_b \in \{\mathbf{w}_{b1}, \mathbf{w}_{b2}\}$ stored at t_{d_i}, we have $\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_{t-1}) > \mathcal{R}_b$ for each time $t_{d_i} < t < t_{d_{i+1}}$ as long as* DriftSurf *has not recovered. Furthermore, we denote Δ to be the magnitude of the drift where $\Delta = \min_{\mathbf{w}_b}(\mathcal{R}_J(\mathbf{w}_b) - \mathcal{R}_I(\mathbf{w}_b))$ where I denotes the distribution at the time $t_{d_i} - 1$ before the drift, and J denotes the distribution at t_{d_i}.*

We assume that all loss functions $f_{\mathbf{x}}$ are bounded $[0, 1]$, that $R^*_{I_t} = \inf_{\mathbf{w}\in\mathcal{F}} \mathcal{R}_{I_t}(\mathbf{w}) = 0$ for each distribution I_t, that the batch size $m > 16/\delta'$, that each drift magnitude $\Delta > \delta$, and that the window size W for storing a frozen model is upper bounded by both $\frac{1}{2}\exp(\frac{1}{2}m\delta^2)$ and $\frac{1}{2}\exp(\frac{1}{2}m(\Delta - \delta)^2)$ for each drift magnitude Δ. Finally, we assume that for each frozen model $\mathbf{w}_b$ that yielded a minimal observed risk $\mathcal{R}_b$, that its expected risk is at least as good as its expectation; i.e., when $\mathbf{w}_b$ is trained over n points, then $\mathcal{R}(\mathbf{w}_b) < \mathbb{E}[\mathcal{R}(\mathbf{w}')]$ for a model $\mathbf{w}'$ trained over n points from the same distribution.

Table 4 summarizes the notation used in this section.

We first establish a couple of preliminary facts we will use.

Observation 1. *Suppose the base learner B trains a model $\mathbf{w}$ over a stream segment $\mathcal{S} \sim I^n$. Then the expected risk is bounded $\mathbb{E}[\mathcal{R}_I(\mathbf{w}) - \mathcal{R}^*_I] \leq 2\mathcal{H}(n)$, where $R^*_I = \inf_{\mathbf{w}'\in\mathcal{F}} \mathcal{R}_I(\mathbf{w}')$.*

Observation 1 follows from Assumption 1 and applying Equation 1 twice.

Each loss function ℓ_x is bounded in $[0, 1]$, and hence, is sub-Gaussian with parameter $\sigma = 1/2$. Therefore, the sum of independent losses has the following concentration.

Theorem 2. *(Hoeffding Bound) Suppose a model* $\mathbf{w}$ *is trained over* $\mathcal{S} \sim I^{n_1}$*. The empirical risk on the test set* $\mathcal{T} \sim J^{n_2}$ *is bounded relative to the expected risk on the distribution* J *as*

$$\Pr[\mathcal{R}_{\mathcal{T}}(\mathbf{w}) > \mathcal{R}_J(\mathbf{w}) + \epsilon] \leq \exp(-2n_2\epsilon^2)$$

and

$$\Pr[\mathcal{R}_{\mathcal{T}}(\mathbf{w}) < \mathcal{R}_J(\mathbf{w}) - \epsilon] \leq \exp(-2n_2\epsilon^2).$$

We will also use the following fact about sub-Gaussian random variables:

Theorem 3. *For a sequence (not necessarily independent) of zero-mean random variables* $Z_1, Z_2, \ldots, Z_k$*, each sub-Gaussian with parameter* σ*, the maximum is bounded*

$$\Pr[\max Z_i \geq \sqrt{2\sigma^2(\log k + \epsilon)}] \leq \exp(-\epsilon).$$

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

B.1. In a Stationary Environment

In §5.1, we considered only the stationary environment during the time $1 < t < t_{d_1}$ before any drifts. In this section, we generalize the results to the stationary environment for any time $t_{d_i} + r_i \leq t < t_{d_{i+1}}$, where r_i is the recovery time for the drift at t_{d_i}. We refer to such a time period as a *recovered state*, in which each model of DriftSurf is trained solely over points from the newest distribution.

Lemma 6. *(Generalized statement of Lemma 1.) In a recovered state, the probability of entering the reactive state is upper bounded by bounded by* $p_s = 2\exp(-\frac{1}{8}m\delta^2)$.

Proof. Let I denote the distribution that each batch is sampled from at each time since the beginning of the first stable state corresponding to the recovered state that DriftSurf is in. The best observed risk $\mathcal{R}_b$ (and the corresponding frozen model $\mathbf{w}_b$) is one of the observed empirical risks $\mathcal{R}_{\mathbf{X}_i}(\mathbf{w}_{i-1})$ from the latest time step $i = t$ to at most $i = t - 2W$ time steps ago. Each empirical risk $\mathcal{R}_{\mathbf{X}_i}(\mathbf{w}_{i-1})$ is the sum of independent sub-Gaussians and is also sub-Gaussian with $\sigma = \frac{1}{2\sqrt{m}}$. Applying Theorem 3 on the sequence $Z_i = \mathcal{R}_I(\mathbf{w}_{i-1}) - \mathcal{R}_{\mathbf{X}_i}(\mathbf{w}_{i-1})$, we have with probability $1 - \exp(-\epsilon_1)$,

$$Z_b = \mathcal{R}_I(\mathbf{w}_b) - \mathcal{R}_b \leq \sqrt{\frac{1}{2m}(\log 2W + \epsilon_1)}.$$

Furthermore, by Theorem 2 (Hoeffding Bound), with probability $1 - \exp(-2m\epsilon_2^2)$,

$$\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_b) \leq \mathcal{R}_I(\mathbf{w}_b) + \epsilon_2.$$

Choosing $\epsilon_1 = 2m\epsilon_2^2 = m\delta^2/8$, we have with probability at least $1 - p_s = 1 - (\exp(-\epsilon_1) + \exp(-2m\epsilon_2^2))$, that $\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_b) - \mathcal{R}_b \leq \delta$, using the triangle inequality and the assumption $2W < \exp(\frac{1}{2}m\delta^2)$. □

Corollary 2. *In a recovered state, if* DriftSurf *enters the reactive state, then with probability at least* $1 - p_s^r$ *it will exit the reactive state early.*

Proof.

$$\Pr[\text{early exit}] = \Pr[\bigcup_{i=1}^{i=r} \text{early exit after } i \text{ time steps}] \geq \sum_{i=1}^{r}(1 - p_s)p_s^{i-1} = (1 - p_s)(\frac{p_s^r - 1}{p_s - 1}) = 1 - p_s^r.$$

□

The bound p_s for entering the reactive state is a constant, which by itself, leads to only a constant lower bound asymptotically on the expected age of the model for a standalone drift detection algorithm based on condition 2, which we later show in Lemma 8. The key to DriftSurf maintaining asymptotically more points is that the bound q_s for switching the model at the end of the reactive state decays as the age of the frozen model increases, which we show next in Lemma 7. In the proof we will use the following theorem (Bennett's inequality).

Theorem 4. *(Bennett's inequality) Let $X_1, X_2, \ldots, X_n$ be independent zero-mean random variables such that $|X_i| \leq M$ and let $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} Var(X_i)$. Then*

$$\Pr\left[\frac{1}{n}\sum_{i=1}^{n} X_i \geq \epsilon\right] \leq \exp\left(-\frac{n\sigma^2}{M^2} h\left(\frac{M\epsilon}{\sigma^2}\right)\right),$$

where $h(u) = (1+u)\log(1+u) - u$.

Lemma 7. *(Generalized statement of Lemma 2.) In a recovered state, if* DriftSurf *enters the reactive state, the probability of switching to the reactive model at the end of the reactive state is bounded by $q_s = c_1/\beta^2$ for $\beta > c_2$, where β is the number of time steps between the initialization of the model $\mathbf{w}_b$ and the time it was frozen, and the constants $c_1 = (2h/m^\alpha)^{mr\delta'/4}$ and $c_2 = \frac{1}{m}(2h/\delta')^{1/\alpha}$.*

Proof. Let I denote the distribution that each batch is sampled from in the current recovered state. For the model $\mathbf{w}_b$ which obtained the minimal observed risk $\mathcal{R}_b$ within its window, by the assumption that its risk is less than the expectation, we have $\mathcal{R}_I(\mathbf{w}_b) \leq 2\mathcal{H}(m\beta)$, using Observation 1 and the assumption $\mathcal{R}_I^* = 0$.

We apply Theorem 4 (Bennett's inequality) on $\mathcal{R}_T(\mathbf{w}_b) - \mathcal{R}_I(\mathbf{w}_b)$. For risk bounded in $[0,1]$, we have that $\text{Var}(\mathcal{R}_I(\mathbf{w}_b)) \leq \mathbb{E}[\mathcal{R}_I(\mathbf{w}_b)] \leq 2\mathcal{H}(m\beta)$. Therefore, $\mathcal{R}_T(\mathbf{w}_b) < 2\mathcal{H}(m\beta) + \delta'/2$ with probability $1 - \exp\left(-mr\sigma^2 h\left(\frac{\delta'}{2\sigma^2}\right)\right) \geq 1 - c_1/\beta^2$, using the assumption that $m > 16/\delta'$.

Note that for $\beta > c_2$, we have $2\mathcal{H}(m\beta) < \delta'/2$. Therefore, with probability $1 - c_1/\beta^2$,

$$\begin{aligned}\mathcal{R}_T(\mathbf{w}'_f) - \mathcal{R}_T(\mathbf{w}_b) + \delta' &\geq \mathcal{R}_T(\mathbf{w}'_f) - (2\mathcal{H}(m\beta) + \delta'/2) + \delta' \\ &> \mathcal{R}_T(\mathbf{w}'_f) - \delta' + \delta' \\ &\geq 0,\end{aligned}$$

again using the fact that the risk is bounded in $[0,1]$. □

We can now prove Corollary 3 to bound the size of the predictive model's sample set.

Corollary 3. *(Generalized statement of Corollary 1.) Let t_r be the time step* DriftSurf *enters a recovered state after a drift at time t_{d_i}. With probability $1-\epsilon$, the size of the sample set $\mathcal{S}$ for the predictive model in the stable state is larger than $n_{t_r,t}/2$ at any time step $t_r + 2W + c_4/(\epsilon - c_3) \leq t < t_{d_{i+1}}$, where $n_{t_r,t}$ is the total number of data points that arrived from time t_r until time t, and constants $c_3 = c_1((c_2+W) - 1/c_2)p_s$ and $c_4 = (2c_3 - 8)c_1^2 p_s^2 + 6c_1 p_s$ (where c_1 and c_2 are the constants in Lemma 2 (same as in Lemma 7)).*

Proof. Let $t' = t - t_r$. Let β_j denote the age of the predictive model at time $t_r + j$. For $t' < t_{d_{i+1}}$, DriftSurf is in a recovered state, and so the probability of discarding the predictive model (entering the reactive state, not exiting early, and changing to the reactive model) at each time step $t_r + j$ can be bounded in terms of p_s and q_s as $\Pr[\text{discard at } t_r + j | \beta_j] \leq p_s^{r+1} q_s(\max(0, \beta_j - 2W))$, since the frozen model is at most $2W$ time steps behind the predictive model. To ensure the probability of switching the model is well defined, let $q'_s(x) = c_1/x^2$ for $x > c_2 + 2W$, and $q'_s(x) = 1$ for $x \leq c_2 + 2W$.

For simplicity, we bound $\Pr[\text{discard at } t_r + j | \beta_j] \leq p_s q'_s(\beta_j - 2W)$. Therefore,

$$\begin{aligned}
\Pr[\beta_{t'} > t'/2] &\geq \Pr[\text{do not discard the model between } t'/2 \text{ and } t'] \\
&\geq \prod_{j \in (t'/2, t']} \Pr[\text{not discard at } j | \beta_j \geq j - t'/2] \\
&\geq 1 - \sum_{j=t'/2+1}^{t'} \Pr[\text{discard at } j | \beta_j \geq j - t'/2] \\
&\geq 1 - \sum_{j=t'/2+1}^{t'} p_s q'_s(j - t'/2 - 2W) \\
&= 1 - \sum_{i=1}^{t'/2} p_s q'_s(i - 2W),
\end{aligned}$$

where the third line is Weierstrass' inequality. The last sum is the lower Riemann sum of a decreasing function of the interval $I = (0, t'/2]$ into unit subintervals, which is upper bounded by the area over I. Continuing,

$$\begin{aligned}
\Pr[\beta_{t'} > t'/2] &\geq 1 - \int_0^{t'/2} p_s q'_s(x - 2W) dx \\
&\geq 1 - (c_2 + 2W)p_s - c_1 p_s \left[\frac{-1}{x - 2W}\right]_{c_2+2W}^{t'/2},
\end{aligned}$$

which is lower bounded by $1 - \epsilon$ whenever $t' > 2W + c_4/(\epsilon - c_3)$. □

Similarly, we can compare the expected value of the age β of the predictive model of DriftSurf to the expected value of the age γ of the predictive model in the standalone drift detection algorithm that resets the model whenever condition 2 holds.

Lemma 8. *(Generalized statement of Lemma 3.) Let t_r be the time step* DriftSurf *enters a recovered state after a drift at time t_{d_i}. At time t, let β be the age of the predictive model in* DriftSurf *and let γ be the age of the model of standalone drift detection. For $t_r + (2W + \frac{2c_4}{1-2c_3}) < t < t_{d_{i+1}}, \mathbb{E}[\beta] > (t - t_r)/4$. Meanwhile, $\mathbb{E}[\gamma] \geq \frac{1}{p_s} - o(1)$ as $t \to \infty$ (i.e., in the absence of future drifts after t_{d_i}.)*

Proof. Choosing $\epsilon = 1/2$ in the proof of 3 gives the bound on $\mathbb{E}[\beta]$. To show the bound for standalone drift detection, let $t' = t - t_r$, and recall that at every time step in a recovered state, the probability of detecting a drift is upper bounded by the constant p_s by Lemma 6. Therefore,

$$\begin{aligned}
\mathbb{E}[\gamma] &= \sum_{k=0}^{t'-1} \Pr[\gamma > k] \\
&\geq \sum_{k=0}^{t'-1} (1 - p_s)^k \\
&= \frac{1 - (1 - p_s)^{t'}}{p_s}.
\end{aligned}$$

□

B.2. In Presence of Abrupt Drifts

For the case of abrupt drift, we first bound the recovery time for DriftSurf through Lemmas 9 and 11, and then establish risk-competitiveness after recovery in Theorem 1.

Consider the case where drift happens during a stable state. In this case, we could bound the number of times DriftSurf enters reactive state:

Lemma 9. *With probability $1-\epsilon_1$, for any value of $k>0$, the number of times* DriftSurf *enters the reactive state before recovering from a drift is less than $\frac{k+1}{q_d}$ where $\frac{1}{\epsilon_1} \leq 1+\frac{k^2}{1-q_d}$.*

Proof. Let X be a random variable denoting the number of times DriftSurf enters the reactive state after a drift 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 $\mu = \mathbb{E}[X] = \frac{1}{q_d}$ and $\sigma^2 = \mathbb{V}ar[X] = \frac{1-q_d}{{q_d}^2}$. Let $\lambda = \frac{k}{q_d}$, therefore,

$$\Pr[X \geq \frac{(k+1)}{q_d}] \leq \frac{1}{1+\frac{k^2}{1-q_d}} \leq \epsilon_1$$

□

Using Lemma 9, we can provide a high probability guarantee on the number of times DriftSurf enters a reactive state before recovering from a drift. Given that the length of a reactive state is at most r, we will have a high probability guarantee on the total time DriftSurf spends in reactive states before it recovers. In addition to that, we need to investigate the total amount of time DriftSurf will spend in stable states. Lemma 10 addresses this problem.

Lemma 10. *Let $Y = \sum_{i=1}^{k'} Y_i$, where $k' \geq 1$ and Y_i for $i = 1, ..., k'$, are independent geometric random variables distributed $Y_i \sim Ge(p_d)$ and $\mathbb{E}[Y] = \frac{k'}{p_d}$. For any $\lambda \geq 1$, we have:*

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

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

Now, by putting together the results of Lemma 9 and lemma:GeometricBound, we can now provide a probabilistic upper bound on the recovery time as follows:

Lemma 11. *With probability $1-\epsilon_r$, the recovery time of* DriftSurf *after a drift that happened during the stable state, is bounded by W, where $\epsilon_r = \epsilon_1 + \epsilon_2$, $\frac{1}{\epsilon_1} \leq 1+\frac{(\frac{Wq_d}{rg}-1)^2}{1-q_d}$, $\epsilon_2 \geq e^{-\frac{W}{rg}}$, and $g = \frac{4\ln 2}{p_d r} + 1$.*

Proof. The number of time steps in recovery time can be divided into two disjoint set of time steps: i) time steps spent in reactive state, and ii) time steps spent in stable state. Using Lemma 9, we can bound the number of times X that DriftSurf enters reactive state before recovering from a drift. Therefore, w.p. at least $1-\epsilon_1$, for any value of $k>0$, we have $X < \frac{k+1}{q_d}$, where $\frac{1}{\epsilon_1} \leq 1+\frac{k^2}{1-q_d}$. Therefore, for the proper choice of ϵ_1, total number of time steps spent in the reactive state is bounded by $r \times X < \frac{(k+1)r}{q_d}$. Note that early exiting the reactive state leads to spent less than r time steps in a reactive state, and therefore, does not change this upper bound.

On the other hand, we have Lemma 10, which bounds the number of time steps spent in the stable state. Let $Y = \sum_{i=1}^{k'} Y_i$, where $k' \geq 1$ and Y_i for $i = 1, ..., k'$, are independent geometric random variables with distributions: $Y_i \sim Ge(p)$. In fact, each Y_i denotes the number of time steps DriftSurf spent in stable state between $i-1$th and i-th times it enters reactive state before recovering from the drift. Using Lemma 10 we have:

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

Therefore, with probability $1-\epsilon_2$, for any value $k' \geq 1$ we have $Y < \frac{k'\lambda}{p_d}$, where $\epsilon_2 \geq e^{-k'(\frac{\lambda}{2}-\ln 2)}$. Note that k' is the same as X which we bounded in before by $\frac{(k+1)}{q_d}$. Consequently, for the choice of $k = \frac{Wq_d}{rg} - 1$ and $\lambda = rp_d(g-1) = 4\ln 2$, where $g = \frac{4\ln 2}{p_d r} + 1$, w.p. at least $1-\epsilon$, we have the recovery time is bounded by W, where $\epsilon = \epsilon_1 + \epsilon_2$, with the corresponding conditions on ϵ_1 and ϵ_2.

□

So far we have discussed the case where drift happens during the stable state. For the case that drift happens within the reactive state, then by the early-exit condition, the drift is likely to be detected upon returning to the stable state and then later re-entering the reactive state with the same probability bound p_d, and switching the model with the same probability bound q_d. We make this precise in the following, and bound the recovery time for either case of drift in Lemma 14.

On the other hand, we have:

Lemma 12. *If a drift happens during the stable state and* DriftSurf *enters the reactive state, then w.p. at most* $1 - p_d^r$ *it will exit the reactive state early.*

Proof.

$$\Pr[\text{early exiting}] = \Pr[\bigcup_{i=1}^{i=r} \text{early exit after } i \text{ time steps}] = \sum_{i=1}^{i=r} \Pr[\text{early exit after } i \text{ time steps}]$$
$$\geq \sum_{i=1}^{r} (1-p_d)p_d^{i-1} = (1-p_d)(\frac{p_d^r - 1}{p_d - 1}) = 1 - p_d^r$$

□

Lemma 13. *If* DriftSurf *enters reactive state due to a false positive, and then a drift happens after* j *time steps in the reactive state, then w.p. at least* $1 - p_s^j p_d^{r-j}$ DriftSurf *exits the reactive state early.*

Proof.

$$\Pr[\text{early exiting}] = \Pr[\bigcup_{i=1}^{i=r} \text{early exit after } i \text{ time steps}] = \sum_{i=1}^{i=r} \Pr[\text{early exit after } i \text{ time steps}]$$
$$\geq \sum_{i=1}^{j-1} (1-p_s)p_s^{i-1} + \sum_{i=1}^{r-j} p_s^j (1-p_d)p_d^{i-1}$$
$$= (1-p_s^j) + p_s^j(1-p_d^{r-j}) = 1 - p_s^j p_d^{r-j}$$

□

Using Lemma 13, and denoting the probability of not exiting the reactive state by ϵ_3, we can generalize Lemma 11 as follows:

Lemma 14. *With probability* $1 - \epsilon_r'$*, the recovery time of* DriftSurf*, from a drift that occurs while either in the stable state or in the reactive state, is bounded by* W*, where* $\epsilon_r' = \epsilon_1 + \epsilon_2 + \epsilon_3$*,* $\frac{1}{\epsilon_1} \leq 1 + \frac{(\frac{Wq_d}{rg} - 1)^2}{1-q_d}$*,* $\epsilon_2 \geq e^{-\frac{W}{rg}}$*,* $g = \frac{4 \ln 2}{p_d r} + 1$*, and* $\epsilon_3 \leq p_d$*.*

One more fact we will use before we prove Theorem 1 on risk-competitiveness is the the following Lemma from (Daneshmand et al., 2016), which is a consequence of the generalization bound in Equation 1.

Lemma 15. (THEOREM 3 IN (DANESHMAND ET AL., 2016)) *If* $\mathbb{E}[\texttt{SUBOPT}_{\mathcal{T}}(\mathbf{w})] \leq \epsilon$*, then* $\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}}(\mathbf{w})] \leq \epsilon + \frac{n-m}{n}\mathcal{H}(m)$ *where* $\mathcal{T} \subset \mathcal{S}, |\mathcal{T}| = m, |\mathcal{S}| = n$*, and* $\mathcal{T}$ *and* $\mathcal{S} - \mathcal{T}$ *are drawn from the same distribution.*

With the preceding lemmas, we can now establish the risk-competitiveness of DriftSurf in the stationary period between abrupt drifts at times t_{d_i} and $t_{d_{i+1}}$. Note that if two drifts occur rapidly in succession, the condition in Lemma 1 of $t_{d_i} + l < t < t_{d_{i+1}}$ may correspond to an empty domain if the recovery time bound of DriftSurf exceeds the gap between the drifts.

Theorem 1. *With probability* $1 - \epsilon$*, the predictive model of* DriftSurf *in the stable state is* $\frac{7}{4^{1-\alpha}}$*-risk-competitive with* Aware *at any time step* $t_{d_i} + 3W + c_4/(\epsilon_s - c_3) \leq t < t_{d_{i+1}}$*, where* t_{d_i} *is the time step of the most recent drift and* $\epsilon = \epsilon_s + \epsilon_r$ *where* ϵ_r *is bounded by Lemma 11 (and where* c_3 *and* c_4 *are the same constants in Corollary 1).*

Proof. By Lemma 11, with probability $1-\epsilon_r$, DriftSurf recovers from drift after W time steps. After recovering from the drift, DriftSurf is in a recovered state. Let t_r be the time step that DriftSurf recovers from the most recent drift at time $t_d = t_{d_i}$. Also, let t_e be the time step that the current predictive model was initialized.

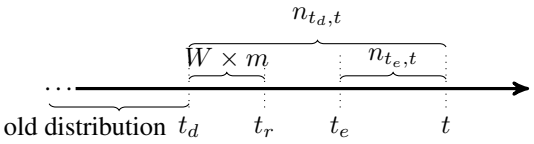


Figure 5: A drift happens at time t_d. DriftSurf recovers by time t_r. The current predictive model is initialized at time t_e.

To show DriftSurf is $\frac{7}{4^{1-\alpha}}$-risk-competitive to Aware, we want to show $n_{t_e,t} \geq \frac{n_{t_d,t}}{4}$. Using Corollary 3, w.p. $1-\epsilon_s$ we have $n_{t_e,t} \geq \frac{n_{t_r,t}}{2}$ at any time step t such that $t_r + 2W + c_4/(\epsilon_s - c_3) \leq t < t_{d_{i+1}}$. Therefore, $n_{t_e,t} \geq n_{t_r,t_e}$. On the other hand, we have

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

Also, at any time step t such that $t - t_d \geq 3W + c_4/(\epsilon_s - c_3) \leq t < t_{d_{i+1}}$, we have $t - t_d \geq 2W$. Therefore,

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

It remains to bound the expected sub-optimality over $\mathcal{S}_{t_d,t}$. Assumption 1 bounds the expected sub-optimality over $\mathcal{S}_{t_e,t}$ as $(1+o(1))\mathcal{H}(n_{t_e,t})$, and Lemma 15 relates the expected sub-optimality over $\mathcal{S}_{t_e,t}$ to the expected sub-optimality over $\mathcal{S}_{t_d,t}$:

$$\begin{aligned} \mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(\text{DriftSurf})] &\leq \mathcal{H}\left(\frac{n_{t_d,t}}{4}\right) + \frac{n_{t_d,t} - n_{t_d,t}/4}{n_{t_d,t}} \mathcal{H}\left(\frac{n_{t_d,t}}{4}\right) \\ &\leq \frac{7}{4^{1-\alpha}} \mathcal{H}(n_{t_d,t}). \end{aligned}$$

□

Similar results can be proved for the case where drift happens during a reactive state. The only difference would be that ϵ_r will be replaced by ϵ'_r, which is defined in Lemma 14.

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).

Another ensemble algorithm we compare against is the Paired Learners (PL) algorithm (Bach & Maloof, 2008). In PL, two models are maintained at a time, a long-lived model that is best-suited in the stationary case, and a new model trained over a recent sliding window of size w that is best-suited in the case of a drift. The new model is switched to based on a test as a function of the recent performance of the two models and a threshold θ. In adapting the original point-wise algorithm to the batched setting in our experiments, we set w to the batch size instead of being a tunable parameter. For the value of θ, the original paper does not suggest a default choice, so we used $\theta = 0.2$, which led to the lowest observed misclassification rate averaged across all datasets when choosing from the range [0.05, 0.5] in increments of 0.05.

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 §4, the threshold for condition 2 is $\delta = 0.1$, and the threshold for condition 3 is $\delta' = \delta/2$. The window size $W = 50$. In the experimental evaluation, we use an empirically better performing substitute for the corner case condition than the early exit process described in the pseudocode—instead of exiting the reactive state early when there is no observed performance degradation, the implementation uses lack of degradation followed by degradation as a sign of potential drift to skip the stable state and immediately re-enter the reactive state.

In our main experiment, on each dataset discussed below, we evaluate DriftSurf, MDDM (the MDDM-G variant), StandardDD, AUE, and the Aware algorithm with oracle access to when drifts occur (discussed in §5). We also run additional experiments for MDDM-A, MDDM-E, PL, 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 ρ that dictates the number of update steps (specifically, gradient computations) that are available to train the model. The different adaptive learning algorithms maintain a different number of models—DriftSurf uses between 1 and 2; Aware, MDDM, and StandardDD use 1; AUE uses 10; and PL uses 2. This leads us to consider two different possibilities for training at each time: (1) each algorithm can use ρ steps per model; or (2) each algorithm has ρ 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.

For our evaluation under equal processing time, we also evaluate another ensemble method, Condor (Zhao et al., 2020). Condor is a more computationally efficient ensemble method than AUE because it only trains one newly added expert at a time. Condor manages a total of K experts, for which weights are updated based on observed losses with exponential decay factor η, and the prediction output is a weighted vote. After each epoch of Condor, a new model is added (deleting the oldest if the total exceeds K) to minimize the loss over the previous epoch plus an added biased regularization term $\frac{\mu}{2}||\mathbf{w} - \mathbf{w}_p||^2$, where $\mathbf{w}_p$ is the weighted linear combination of the ensemble's experts. In adapting the original point-wise Condor algorithm to our batch setting, we redefine an epoch to be the batch size of the stream for consistent comparison. We set $K = 25, \eta = 0.75$ following the choice made by the authors in their experimental evaluation. Finally, we set μ to be the same regularization constant per dataset we use for L2-regularization in training the models of the other evaluated algorithms.

C.2. Datasets

Our experiments use the 5 synthetic, 2 semi-synthetic and 3 real-world datasets shown in Table 5 and described below. The selection of datasets included all datasets for binary classification 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,

Table 5: Basic statistics of datasets

	DATASET	# INSTANCE	# DIM
SYNTHETIC	SEA	100000	3
	HYPERPLANE	100000	10
	SINE1	10000	2
	MIXED	100000	4
	CIRCLES	10000	2
SEMI-SYNTHETIC	RCV1	20242	47235
	COVERTYPE	581012	54
REAL	AIRLINE	5810462	13
	ELECTRICITY	45312	13
	POWERSUPPLY	29928	2

$\theta_1 = 9, \theta_2 = 8, \theta_3 = 7, \theta_4 = 9.5$ (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. SEA-gradual is generated by generating samples from two concepts (the first two concepts discussed above) during the drift period.

- 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.

- Mixed (Pesaranghader et al., 2016): This dataset contains four attributes (x_1, x_2, x_3, x_4), where x_1 and x_2 are boolean and x_3, x_4 are uniformly distributed in $[0, 1]$. Label of each data is determined to be positive if two of x_1, x_2, and $x_4 < 0.5 + 0.3 \times \sin(3\pi x_3)$ hold. Labels are reversed at drift points.

- Circles (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 circle as the decision boundary as follows: $(x_1 - c_1)^2 + (x_2 - c_2)^2 <= r$, where (c_1, c_2) and r are (respectively) center and radius of the circle. Drift happens in a gradual manner where the center and radius of decision boundary changes over a period of time. We experimented on a generated dataset with 3 gradual drift introduced at time $25, 50$, and 75, where the transition period for each drift is 5 time steps.

- 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. For the experiments on the high-dimensional RCV1 when using the Hoeffding Tree and Naive Bayes base learners, we add a pre-processing step to the dataset to select only 100 features, as determined by the coordinates with the highest magnitude when fitting a logistic regression model to the dataset before drift was added.

- 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 $[0, 1]$. The original dataset contains 13 features. But, after using one-hot encoding the dimension increases to 679.

- 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 6. 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 6: Details of drifts in datasets

	DATASET	DRIFT TYPE	DRIFT TIMES
SYNTHETIC	SEA	ABRUPT	[25, 50, 75]
		GRADUAL	[40-60]
	HYPERPLANE	GRADUAL	-
	SINE1	ABRUPT	[20, 40, 60, 80]
	MIXED	ABRUPT	[20, 40, 60, 80]
	CIRCLES	GRADUAL	[25-30, 50-55, 75-80]
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 μ and the constant step size η. 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 μ and η 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 μ of the form 10^{-a} for $1 \leq a \leq 7$ and η 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 7. In experiments where we used SGD for training, we used the same constant step size η.

We also run experiments using Hoeffding Trees (HT) and Naive Bayes (NB) as base learners. We use the implementation of HT and NB available in the scikit-multiflow package (Montiel et al., 2018), using their default hyperparameters (namely for HT, the grace period is 200, the split criterion uses information gain, the split confidence is 10^{-7}, and the tie-threshold is 0.05). In these experiments, we use the following hyperparameters for DriftSurf: $r = 6, \delta = 0.05, \delta' = \delta/10, W = 50$.

In the streaming data setting studied in this paper (§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

Table 7: Hyperparameters and batch sizes

DATASET	REGULARIZATION μ	STEP SIZE η	BATCH SIZE m
SEA (ALL)	10^{-2}	1×10^{-3}	1000
HYPER-SLOW	10^{-3}	1×10^{-1}	1000
HYPER-FAST	10^{-3}	1×10^{-2}	1000
SINE1	10^{-3}	2×10^{-1}	100
MIXED	10^{-3}	1×10^{-1}	1000
CIRCLES	10^{-3}	1×10^{-1}	100
RCV1	10^{-5}	5×10^{-1}	202
COVERTYPE	10^{-4}	5×10^{-3}	5810
AIRLINE	10^{-3}	2×10^{-2}	581
ELECTRICITY	10^{-4}	1×10^{-1}	1333
POWERSUPPLY	10^{-3}	1×10^{-1}	299

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 7.

D. Additional Experimental Results

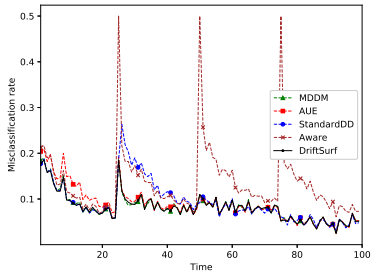
This section contains experimental results under both training strategies of equal computational power for each model and equal computational power for each algorithm, which is divided among its models. Additionally, we report results for a sensitivity analysis of the threshold in DriftSurf, results for DriftSurf without the greedy approach during the reactive state, results for single-pass SGD and an oblivious algorithm using STRSAGA, results for each algorithm when SGD is used as the update process instead of STRSAGA, results when using Hoeffding Trees and Naive Bayes classifiers as the base learners, and results of each algorithm under a 95%-recovery-time metric.

D.1. Equal Computational Power for Each Model

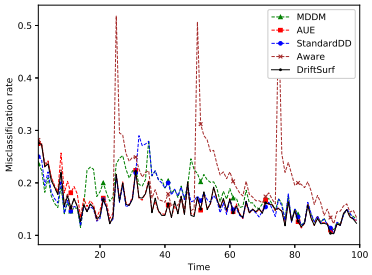
First, we compare each algorithm under the setting where at every time step, each algorithm uses $\rho = 2m$ update steps (gradient computations) to update each of its models. In this case, the total computational power used varies per algorithm. For example, at each time step AUE maintains an ensemble of ten models, while MDDM maintains just one (and DriftSurf maintains either one or two), so AUE uses ten times the total computation of MDDM. (Later in §D.2, we will study a different setting where the available computational power for each algorithm is divided equally among all of its models, in order to account for the varying computational efficiency of each algorithm.)

We present the misclassification rates at each time step over the new batch in Figure 6, and the average misclassification rate over all time steps is summarized in Table 8. (These results are a superset of those presented in Figure 1 and Table 2 from §6). The advantage of DriftSurf over MDDM is most evident on the noisy versions of SEA (also shown in Figure 7), and on CoverType, Electricity, and PowerSupply. The drift detection method MDDM (and similarly over StandardDD) 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.

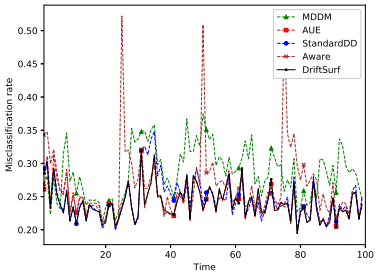
For sharp drifts when immediately switching to a new model is desirable, we observe, most evident on SINE1, that MDDM is the fastest to adapt, followed shortly by DriftSurf, then PL, with AUE lagging behind. CoverType also is a clear example where DriftSurf and StandardDD adapt faster than AUE (but MDDM suffered as previously mentioned). For these drifts, MDDM and StandardDD naturally lead because they are using a new model when they accurately detect a 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. 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



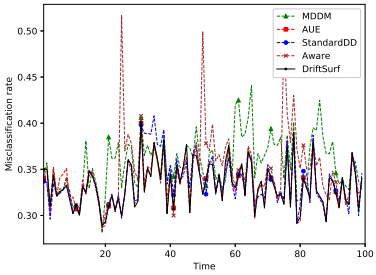
(a) SEA0



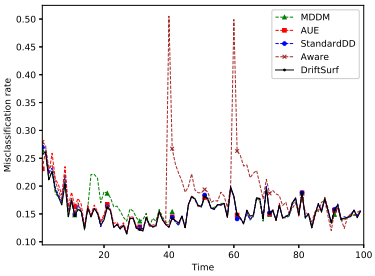
(b) SEA10



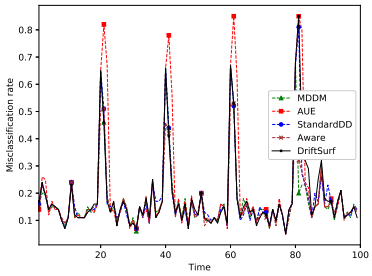
(c) SEA20



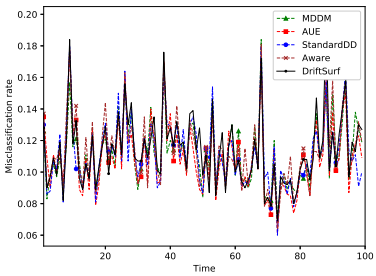
(d) SEA30



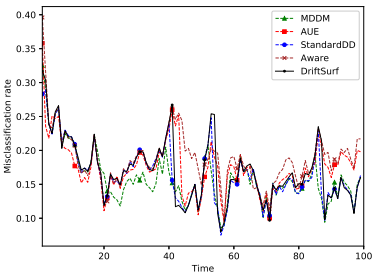
(e) SEA-gradual



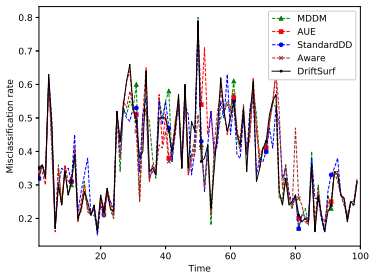
(f) SINE1



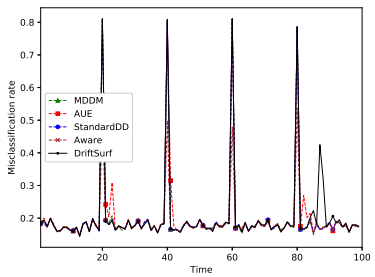
(g) HyperPlane-slow



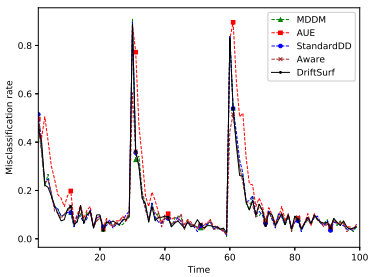
(h) HyperPlane-fast



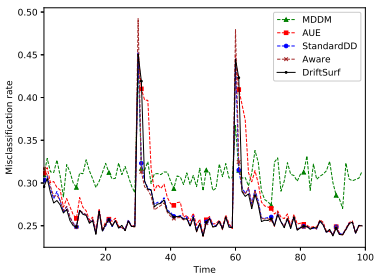
(i) Circles



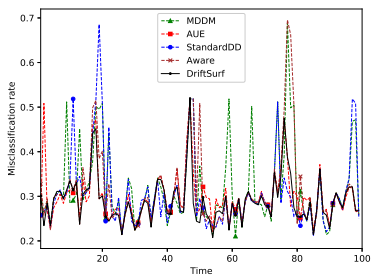
(j) Mixed



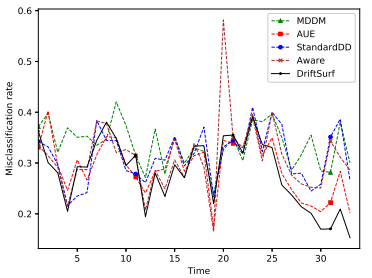
(k) RCV1



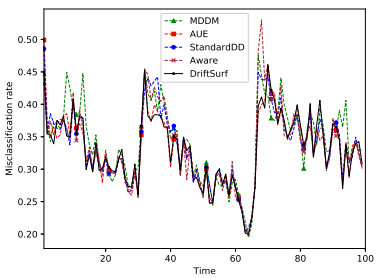
(l) CoverType



(m) PowerSupply



(n) Electricity



(o) Airline

Figure 6: Misclassification rate over time ($\rho = 2m$ for each model)

Table 8: Total average of misclassification rate ($\rho = 2m$ for each model)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	MDDM-A	MDDM-E	AUE	PL	1PASS-SGD	OBL
SEA0	0.137	0.086	0.097	**0.086**	0.090	0.087	0.093	**0.085**	0.131	0.110
SEA10	0.197	**0.160**	0.168	0.180	0.166	0.172	0.163	0.161	0.188	0.176
SEA20	0.264	**0.243**	0.249	0.289	0.278	0.289	0.245	**0.243**	0.267	0.254
SEA30	0.350	**0.335**	0.338	0.358	0.358	0.352	0.337	0.337	0.348	0.338
SEA-GRADUAL	0.177	0.159	0.160	0.165	0.167	0.174	0.162	**0.157**	0.196	0.173
HYPER-SLOW	0.116	0.118	0.116	0.116	0.117	0.116	**0.112**	0.118	0.139	0.170
HYPER-FAST	0.191	0.173	0.168	**0.163**	**0.163**	0.164	0.179	0.188	0.177	0.280
SINE1	0.171	0.187	0.184	0.176	**0.175**	0.178	0.212	0.193	0.223	0.477
MIXED	0.192	0.204	0.204	0.204	0.204	**0.203**	0.209	0.219	0.208	0.455
CIRCLES	0.368	**0.371**	0.377	0.372	0.375	0.372	0.379	0.373	0.385	0.508
RCV1	0.121	**0.125**	0.126	**0.125**	0.130	0.130	0.167	0.148	0.276	0.468
COVERTYPE	0.267	0.268	**0.267**	0.311	0.311	0.313	0.279	0.287	0.298	0.321
AIRLINE	0.338	0.334	0.338	0.345	0.346	0.348	**0.333**	**0.333**	0.340	0.359
ELECTRICITY	0.315	**0.290**	0.320	0.344	0.339	0.341	0.296	0.291	0.347	0.302
POWERSUPPLY	0.309	**0.292**	0.308	0.322	0.315	0.329	0.301	0.306	0.307	0.312

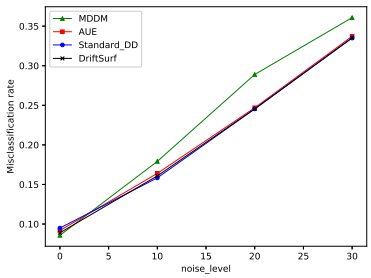


Figure 7: Total average of misclassification rate for SEA dataset with different levels of noise

negatively impact the predictions. On SINE1, we observe that the ensemble of two models, PL, adapts faster than AUE. At every time step, PL trains a new model over the latest batch, similarly to AUE, but differs in that it the latest model is either switched to as the predictive model, or discarded. That is, PL uses exclusively an old or new model, instead of a weighted combination over ten models like in AUE.

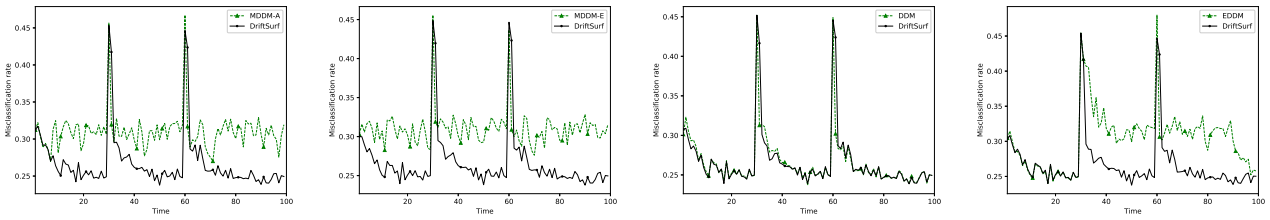
There are two major advantages of DriftSurf and AUE not immediately switching to the latest model: (i) there are drifts for which switching to a new model is not desired because the older model can still provide good accuracy, and (ii) delaying the switch to a new model can be desired if the new model 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, would 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 8, 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 and AUE are the best performers, followed by StandardDD and MDDM, and then 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.

The Hyperplane-slow and Hyperplane-fast datasets 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. Furthermore, on the remaining datasets with gradual drift, SEA-gradual and Circles, that contain stationary periods and drift periods instead of the continual drift of Hyperplane, DriftSurf is the best performer.

Table 8 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 performance 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 8, we observed DDM accurately detected the two drifts, but EDDM also suffered with continual false positives.



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

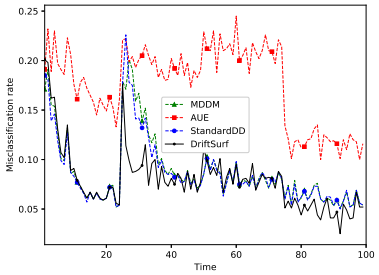
Figure 8: CoverType dataset, comparing different drift detectors ($\rho = 2m$ for each model)

Table 8 also includes results for PL, the ensemble of 2 models. As noted earlier, by choosing exclusively an old or new model it can adapt faster than AUE on datasets like SINE1 or RCV1. However, PL's performance is still short of DriftSurf on SINE1, as well as on PowerSupply, because PL may falsely switch to the new model in the absence of drift, while the condition DriftSurf reduces false switches. Furthermore, PL is short of DriftSurf on RCV1 because PL only gives the new model one time step to be switched in before being discarded, requiring multiple tries to switch after the drift.

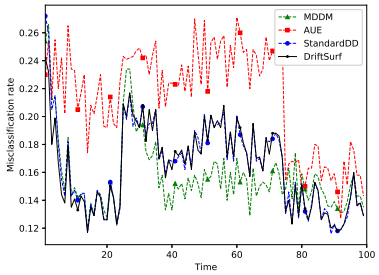
D.2. Equal Computational Power for Each Algorithm

Next, we present results for the training strategy where each algorithm has access to ρ 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 9 for the comparison of DriftSurf, Aware, MDDM, and AUE and in Figure 10 for the additional algorithmic comparisons against two ensemble methods, AUE ($k = 2$) and Condor. The average over time is in Table 9. For the case $\rho = 2m$, the misclassification rate at each time is shown in Figure 11, and the average over time is in Table 10.

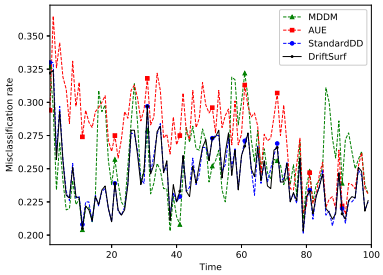
Let us discuss a few differences from the previous case where each model was trained with ρ steps. We generally observe lower relative accuracy for AUE, and especially so after drifts. (The exceptions are on Circles and PowerSupply, where the extra training iterations do not matter as much; compare to the fast convergence of Aware after a reset.) This is because AUE is an ensemble of 10 models, and so each model is trained at most 1/5 of the steps that the models of DriftSurf get, and



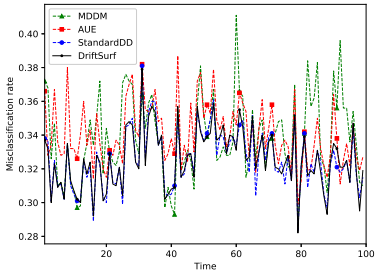
(a) SEA0



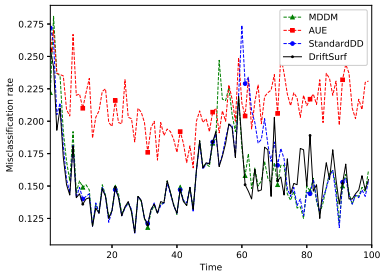
(b) SEA10



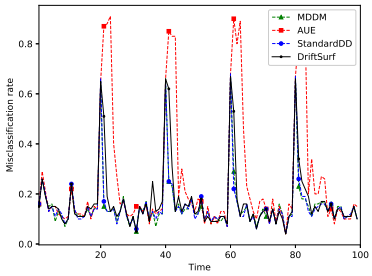
(c) SEA20



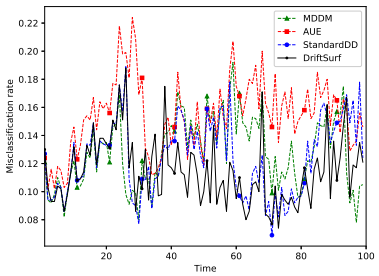
(d) SEA30



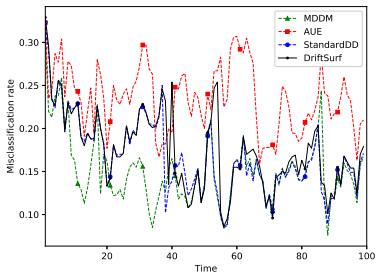
(e) SEA-gradual



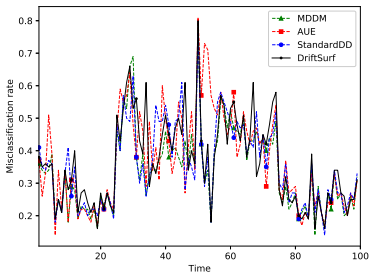
(f) SINE1



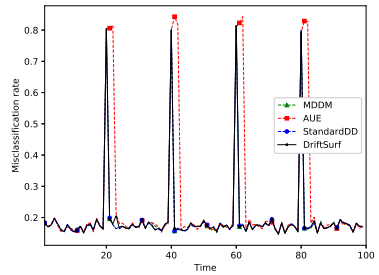
(g) HyperPlane-slow



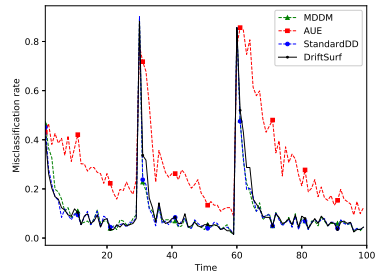
(h) HyperPlane-fast



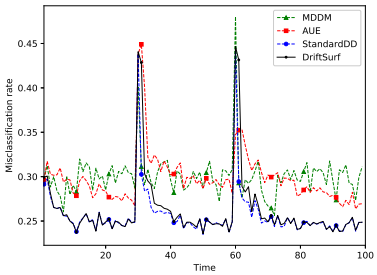
(i) Circles



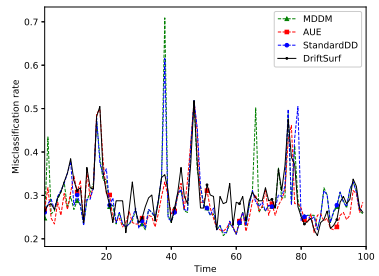
(j) Mixed



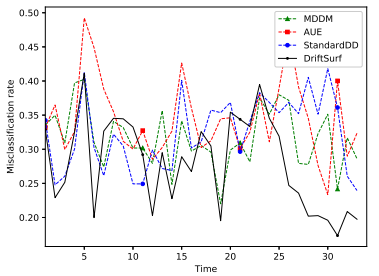
(k) RCV1



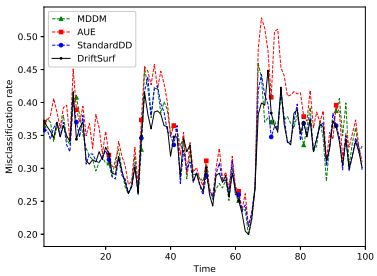
(l) CoverType



(m) PowerSupply

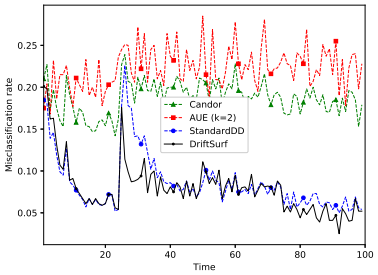


(n) Electricity

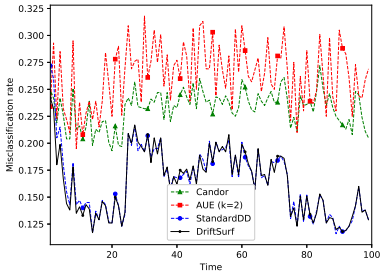


(o) Airline

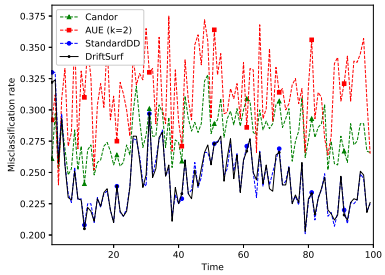
Figure 9: Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm) comparing Aware, DriftSurf, AUE and MDDM



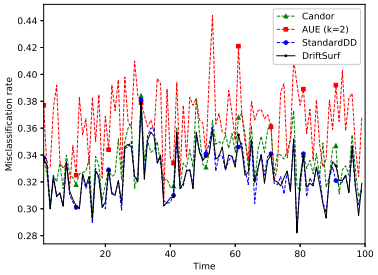
(a) SEA0



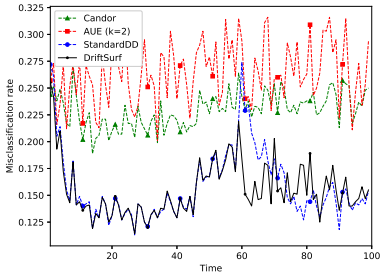
(b) SEA10



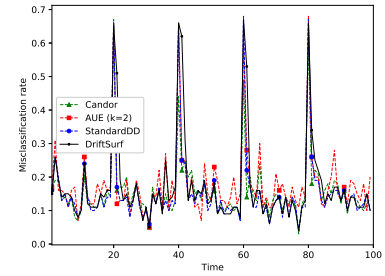
(c) SEA20



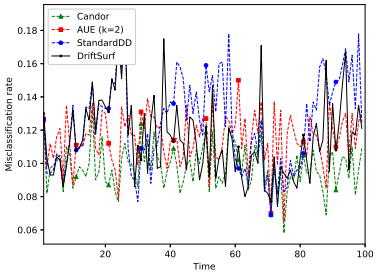
(d) SEA30



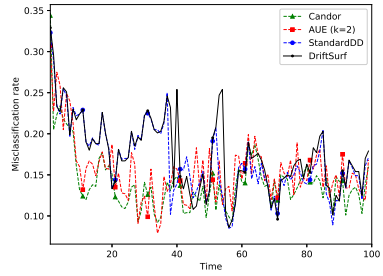
(e) SEA-gradual



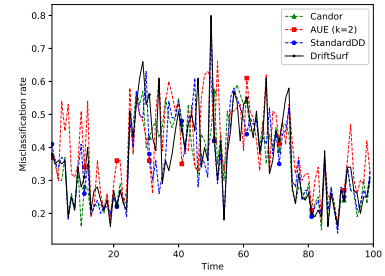
(f) SINE1



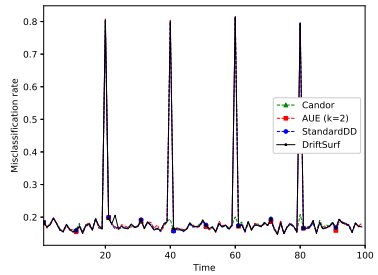
(g) HyperPlane-slow



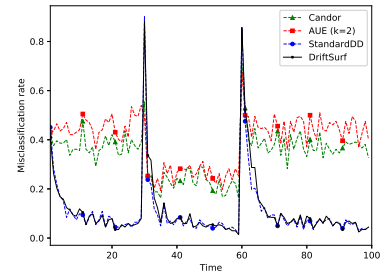
(h) HyperPlane-fast



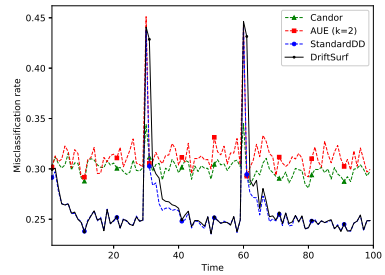
(i) Circles



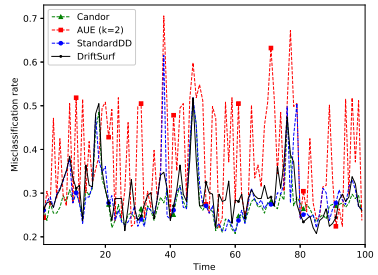
(j) Mixed



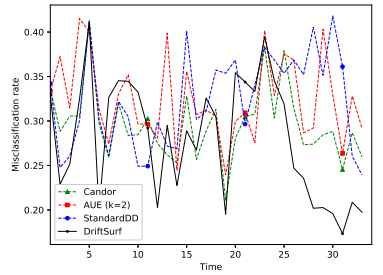
(k) RCV1



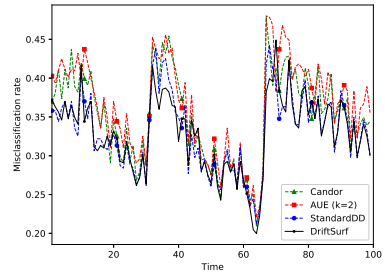
(l) CoverType



(m) PowerSupply



(n) Electricity



(o) Airline

Figure 10: Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm) comparing Aware, DriftSurf, AUE with $k = 2$ and Condor

Table 9: Total average of misclassification rate ($\rho = 4m$ divided among all models of each algorithm)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE	AUE (k=2)	CONDOR	PL
SEA0	0.120	**0.084**	0.091	0.092	0.179	0.226	0.192	0.085
SEA10	0.179	0.167	0.169	**0.160**	0.218	0.269	0.234	**0.160**
SEA20	0.256	0.247	0.247	0.258	0.280	0.320	0.283	**0.242**
SEA30	0.334	**0.327**	**0.327**	0.341	0.342	0.365	0.338	0.338
SEA-GRADUAL	0.170	**0.159**	0.162	0.160	0.215	0.267	0.232	0.161
HYPER-SLOW	0.145	0.119	0.128	0.132	0.158	0.120	**0.103**	0.117
HYPER-FAST	0.222	0.177	0.171	0.154	0.238	0.154	**0.144**	0.191
SINE1	0.149	0.176	0.158	**0.157**	0.263	0.181	0.159	0.192
MIXED	0.188	0.201	0.200	0.200	0.254	0.203	**0.182**	0.217
CIRCLES	0.345	0.365	0.357	**0.341**	0.372	0.424	0.360	0.370
RCV1	0.101	0.116	**0.110**	0.113	0.310	0.404	0.341	0.137
COVERTYPE	0.260	0.264	**0.259**	0.302	0.301	0.314	0.303	0.287
AIRLINE	0.335	**0.330**	0.333	0.337	0.360	0.366	0.353	0.332
ELECTRICITY	0.310	**0.289**	0.332	0.324	0.348	0.326	0.300	**0.289**
POWERSUPPLY	0.303	0.300	0.294	0.292	0.284	0.393	**0.282**	0.300

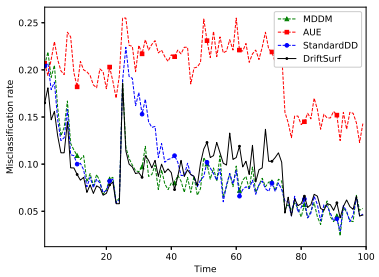
Table 10: Total average of misclassification rate ($\rho = 2m$ divided among all models of each algorithm)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE	AUE (k=2)	CONDOR	PL
SEA0	0.133	0.094	0.097	**0.089**	0.201	0.230	0.200	0.131
SEA10	0.197	**0.161**	0.163	0.183	0.237	0.275	0.238	0.194
SEA20	0.266	**0.249**	0.253	0.283	0.291	0.327	0.292	0.271
SEA30	0.352	**0.337**	0.339	0.360	0.354	0.381	0.345	0.352
SEA-GRADUAL	0.174	**0.160**	0.161	0.172	0.24	0.273	0.239	0.188
HYPER-SLOW	0.117	0.130	0.117	**0.116**	0.191	0.166	0.122	0.185
HYPER-FAST	0.191	0.188	0.168	**0.164**	0.278	0.211	0.166	0.222
SINE1	0.168	0.209	0.180	**0.178**	0.309	0.246	0.179	0.207
MIXED	0.191	0.204	0.204	0.204	0.259	0.204	**0.182**	0.229
CIRCLES	0.368	**0.369**	0.380	0.372	0.401	0.415	0.384	0.388
RCV1	0.120	0.143	**0.128**	0.131	0.403	0.467	0.401	0.195
COVERTYPE	0.267	0.271	**0.267**	0.313	0.317	0.330	0.312	0.280
AIRLINE	0.338	0.348	**0.338**	0.351	0.369	0.380	0.365	0.369
ELECTRICITY	0.311	**0.308**	0.319	0.339	0.364	0.363	0.313	0.354
POWERSUPPLY	0.311	**0.307**	0.311	0.309	0.313	0.463	0.338	0.342

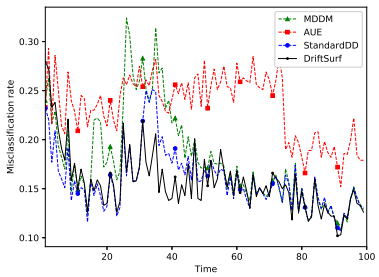
only 1/10 of the models for MDDM and Aware. DriftSurf now dominates AUE in average misclassification rate on each dataset except for PowerSupply. The relative performance of PL to DriftSurf is generally similar in the $\rho = 4m$ case, but in the $\rho = 2m$ case, PL does relatively worse, losing its edge on Electricity and the SEA datasets. Because PL only gives a new model one chance to perform well before discarding it, the restricted training iterations of the new model makes it significantly harder to adapt.

We observe DriftSurf compares favorably to MDDM and StandardDD on the same datasets as it did in the undivided ρ case. However, MDDM's and StandardDD'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 and StandardDD as before, because their relative additional training per model means that their new models catch up faster. For Hyperplane-fast, the relative additional training for MDDM was advantageous. 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.

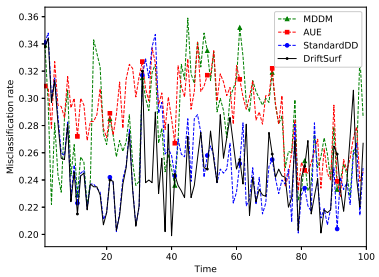
In Tables 9 and 10, we present results for a variation on AUE that is limited to only two experts, which we refer to as AUE ($k = 2$). In our comparison of each algorithm when enforcing equal computation time, dividing the ρ steps equally among a total of ten experts in the original AUE is unsurprisingly 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



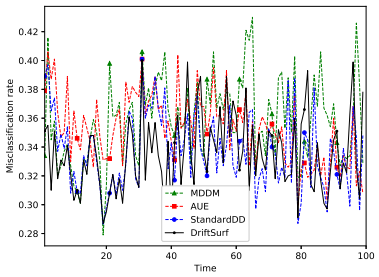
(a) SEA0



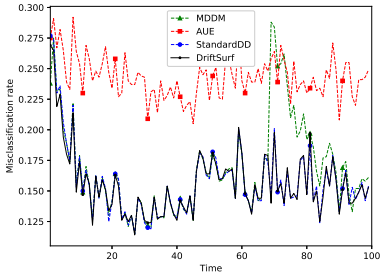
(b) SEA10



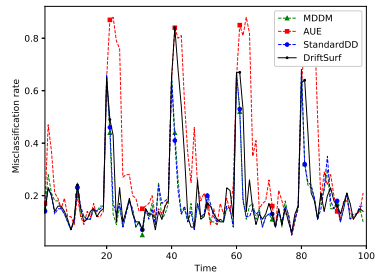
(c) SEA20



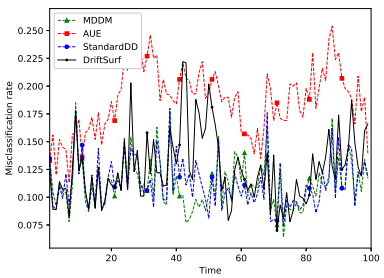
(d) SEA30



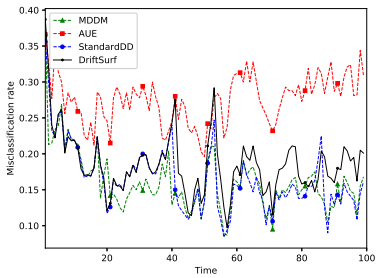
(e) SEA-gradual



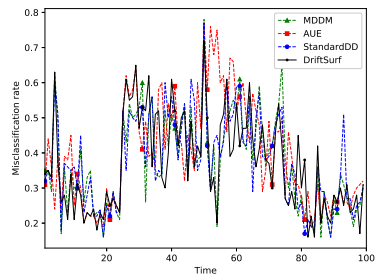
(f) SINE1



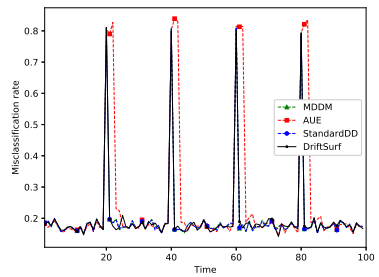
(g) HyperPlane-slow



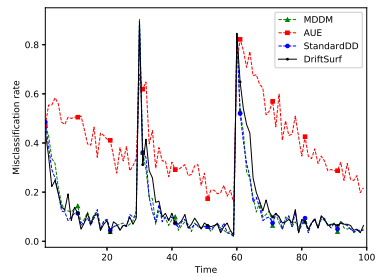
(h) HyperPlane-fast



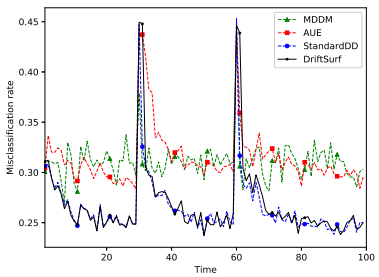
(i) Circles



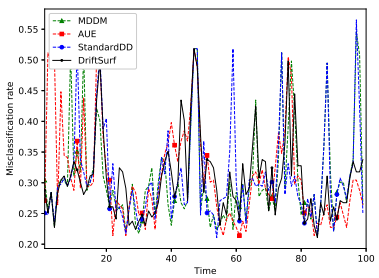
(j) Mixed



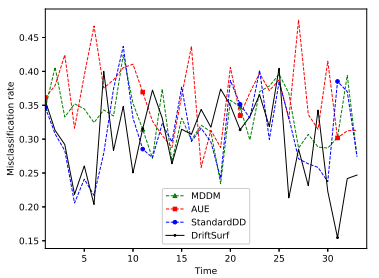
(k) RCV1



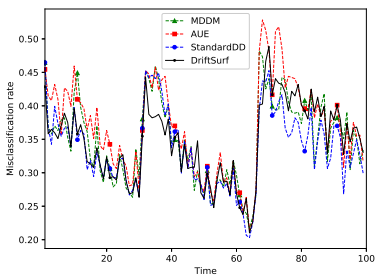
(l) CoverType



(m) PowerSupply



(n) Electricity



(o) Airline

Figure 11: Misclassification rate over time ($\rho = 2m$ divided among all models of each algorithm)

to DriftSurf. We observe that AUE ($k = 2$) performs better than AUE on five datasets: Hyperplane-slow, Hyperplane-fast, SINE1, Mixed, 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 and Mixed datasets 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.

In Tables 9 and 10, we present results for another ensemble method Condor, which is better suited for the setting studied in this section normalizing the computational power because it only requires training a single model at a time. Another distinctive feature of Condor is that it uses biased regularization during training to anchor the newest model closer to the weighted ensemble average from the previous time step. For these two factors, we expect that Condor is better at adapting to drift at the expense of stationary performance, which is exemplified by its high accuracy on the Mixed, PowerSupply and the continually drifting Hyperplane datasets and its relative improvement over AUE on some other datasets including SINE1, Circles, Airline, Electricity and PowerSupply.

D.3. Sensitivity Analysis of δ

Choosing the right threshold is a key challenge for any drift detection technique. However, one of the key strengths of DriftSurf is its resilience to imprecision in detection. In this experiment, we compared DriftSurf to StandardDD (the baseline drift detection algorithm that uses condition 2 to decide whether to reset the model) under a range of settings of δ and plotted the misclassification rate for each dataset in Figure 12.

As we observed in §6 from the results averaged over the datasets in Figure 3, we can generally choose a small value of δ in DriftSurf to detect subtle drifts while not sacrificing performance between drifts because the reactive state catches most false positives.

Over all variations of the SEA dataset, RCV1, Electricity, and Airline, larger values of δ improve the performance since the permitted variation within the results will not be mistaken as drifts. This is true for both StandardDD, especially, while DriftSurf's performance is more stable because the reactive state corrects the false positive detections either by switching to the old model or by early exiting.

For the Hyperplane datasets (fast and slow) with continuous gradual drifts, StandardDD outperforms DriftSurf because resetting the model after any drift detection tends to improve the performance. There is an exception to this for the Hyperplane-slow under very small δ, where the reactive state length of DriftSurf limits excessive switching. As δ increases, neither DriftSurf and StandardDD detect changes, and as a result, they perform the same.

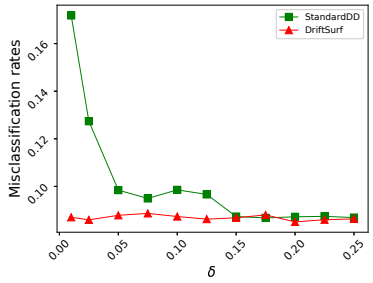
In some other datasets with large, abrupt drifts (SINE1, Mixed, CoverType), StandardDD outperforms DriftSurf, especially for larger values for δ. In such cases, the right thing to do is to reset the model, but DriftSurf suffers from a delayed reaction to such drifts it has to enter reactive state and leave the reactive state with a new model. These delays in reacting to actual drifts make StandardDD outperform DriftSurf even for smaller choices of δ on SINE1.

D.4. Evaluation of Greedy Reactive State

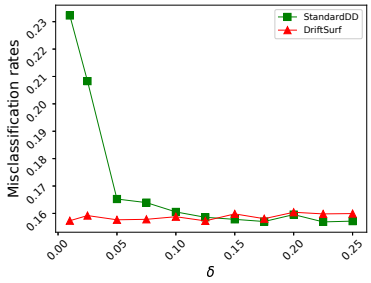
This section includes results for the comparison of DriftSurf to DriftSurf (no-greedy). Recall that DriftSurf uses greedy prediction in the reactive state, meaning that the predictive model used at one time step is the model with better performance from the previous time step, while DriftSurf (no-greedy) only uses the older model during the reactive state, and may only switch to the new model at the end of the reactive state. In Table 11 we observe that DriftSurf performs similar or better across each dataset, with the biggest improvements on the SINE1, RCV1, and Mixed datasets that we earlier observed MDDM and Aware perform well on because it is desirable to immediately switch to the new model after the large, abrupt drift.

D.5. Comparison to 1PASS-SGD and Oblivious

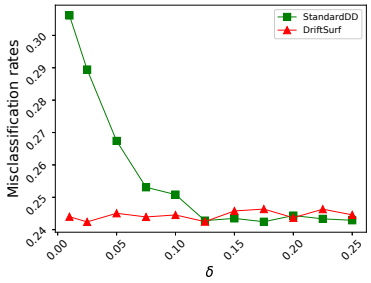
Figure 13 shows the comparison to 1PASS-SGD and the oblivious algorithm (OBL) for the RCV1 and Electricity datasets at each time. The time average misclassification rate for each dataset are in Table 8. In the case of the large, abrupt drift in RCV1, we observe that 1PASS-SGD and especially oblivious have poor performance after drift. The oblivious algorithm



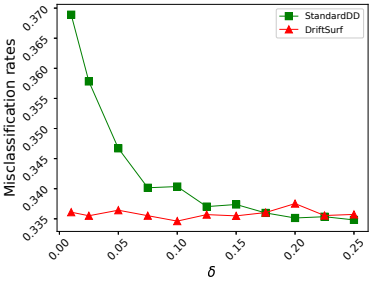
(a) SEA0



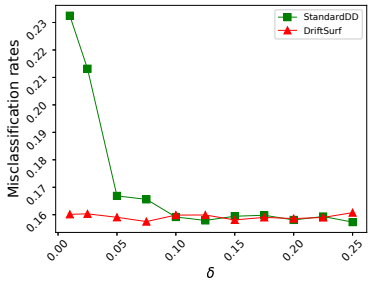
(b) SEA10



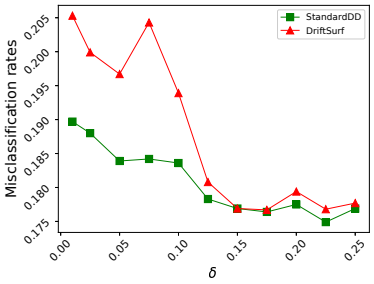
(c) SEA20



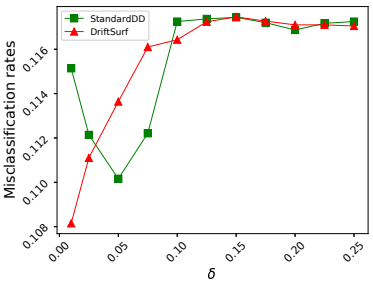
(d) SEA30



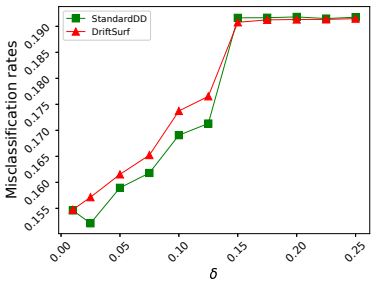
(e) SEA-gradual



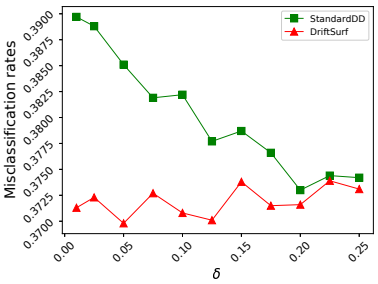
(f) SINE1



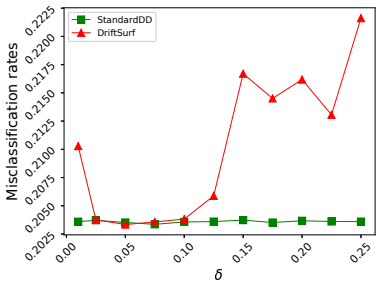
(g) HyperPlane-slow



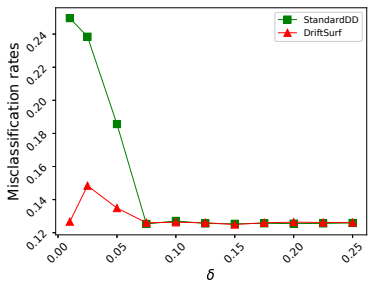
(h) HyperPlane-fast



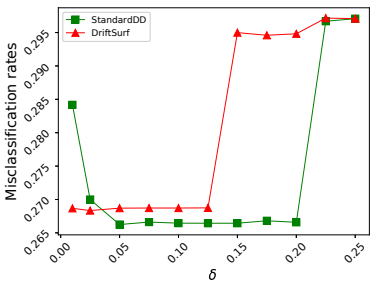
(i) Circles



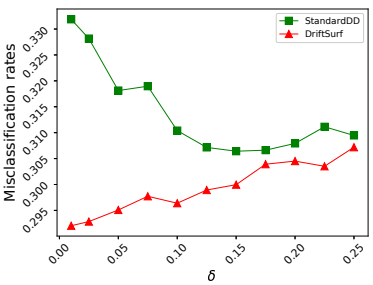
(j) Mixed



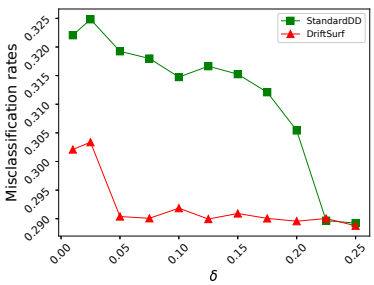
(k) RCV1



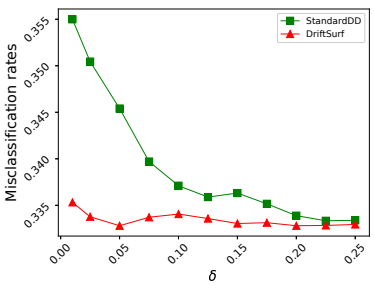
(l) CoverType



(m) PowerSupply



(n) Electricity



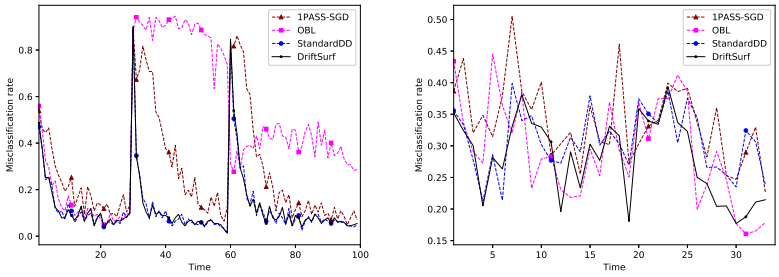
(o) Airline

Figure 12: δ-sensitivity ($\rho = 2m$ for each model)

Table 11: Total average of misclassification rate - DriftSurf vs DriftSurf (no-greedy) ($\rho = 2m$ for each model)

DATASET	DriftSurf	DriftSurf (NO-GREEDY)
SEA0	0.087	**0.085**
SEA10	0.161	**0.158**
SEA20	0.247	**0.246**
SEA30	**0.335**	0.336
SEA-GRADUAL	**0.158**	0.159
HYPER-SLOW	**0.117**	0.118
HYPER-FAST	**0.173**	0.177
SINE1	**0.191**	0.220
MIXED	**0.204**	0.238
CIRCLES	**0.371**	0.376
RCV1	**0.134**	0.158
COVERTYPE	**0.267**	0.273
AIRLINE	**0.333**	**0.333**
ELECTRICITY	**0.284**	0.287
POWERSUPPLY	**0.303**	**0.303**

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.



(a) RCV1 (b) Electricity

Figure 13: Misclassification rate over time ($\rho = 2m$ for each model)

D.6. 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 variance-reduced update step. We study the impact of the choice of the update process on the performance. We re-run the previous experiments using SGD instead of STRSAGA. Table 12 shows the average misclassification rate for the case where $\rho = 2m$ update steps are used for each model.

As the results presented in Table 12 suggest, AUE, unlike the previous experiment, outperforms MDDM and DriftSurf for the majority of the studied datasets. The reason is that AUE mitigates the high variance of SGD. MDDM, StandardDD, and DriftSurf all 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

Table 12: Total average of misclassification rate - update process: SGD ($\rho = 2m$ for each model)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE
SEA0	0.170	0.119	**0.115**	0.127	0.125
SEA10	0.217	0.186	0.191	0.197	**0.184**
SEA20	0.279	0.271	0.276	0.296	**0.263**
SEA30	0.360	0.352	0.360	0.382	**0.340**
SEA-GRADUAL	0.205	**0.179**	0.193	0.216	0.188
HYPER-SLOW	0.169	0.155	0.146	0.140	**0.124**
HYPER-FAST	0.272	0.201	0.203	**0.179**	0.204
SINE1	0.194	**0.199**	0.216	0.200	0.239
MIXED	0.194	**0.207**	0.209	0.209	0.242
CIRCLES	0.362	0.389	0.390	0.386	**0.362**
RCV1	0.151	**0.154**	0.155	0.162	0.208
COVERTYPE	0.274	0.276	**0.273**	0.326	0.286
AIRLINE	0.356	0.351	0.353	0.359	**0.343**
ELECTRICITY	0.335	0.337	0.359	0.348	**0.299**
POWERSUPPLY	0.350	0.321	0.390	0.365	**0.300**

Table 13: Total average of misclassification rate - update process: SGD ($\rho = 2m$ divided among all models of each algorithm)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE
SEA0	0.163	0.127	0.125	**0.123**	0.193
SEA10	0.229	**0.186**	0.198	0.197	0.238
SEA20	0.283	**0.267**	0.279	0.311	0.287
SEA30	0.366	**0.353**	0.355	0.376	0.358
SEA-GRADUAL	0.204	**0.187**	0.188	0.196	0.236
HYPER-SLOW	0.169	0.155	0.149	**0.143**	0.158
HYPER-FAST	0.269	0.211	0.191	**0.185**	0.274
SINE1	0.200	0.241	0.212	**0.205**	0.302
MIXED	0.195	0.207	**0.206**	0.209	0.262
CIRCLES	0.306	0.371	0.389	**0.367**	0.429
RCV1	0.146	0.168	**0.156**	0.161	0.437
COVERTYPE	0.275	0.282	**0.275**	0.323	0.316
AIRLINE	0.354	0.365	**0.353**	0.366	0.370
ELECTRICITY	0.343	**0.316**	0.342	0.350	0.354
POWERSUPPLY	0.336	0.318	0.348	0.356	**0.316**

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 13 suggest that the variance-reduction effect of AUE is not able to overcome the limited training.

STRSAGA because of its variance-reduced update step achieves a faster convergence rate in comparison to SGD. Difference between the reported results in Table 8 and Table 12 confirms the advantage of using STRSAGA over SGD as the update process.

D.7. Using Hoeffding Trees and Naive Bayes as Base Learners

For the experiments in §D.1–§D.6, the base learner across each algorithm is a logistic regression model using either STRSAGA or SGD as the update process. In this section, we study two additional base learners: Hoeffding Trees (HT), and Naive Bayes (NB).

When using HT as the base learner, we also compare against an additional adaptive learning algorithm, Hoeffding Adaptive

Table 14: Total average of misclassification rate - base learner: HT ($\rho = 2m$ divided among all models of each algorithm)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE	HAT
SEA0	0.038	**0.032**	**0.032**	0.045	0.053	0.037
SEA10	0.139	**0.134**	0.136	0.139	0.143	0.140
SEA20	0.233	**0.230**	0.231	0.231	0.234	0.233
SEA30	0.327	**0.324**	0.329	0.329	0.327	0.326
SEA-GRADUAL	0.141	0.133	0.135	**0.131**	0.148	0.135
HYPER-SLOW	0.162	0.140	0.126	0.153	**0.123**	0.149
HYPER-FAST	0.245	0.173	**0.159**	0.168	0.162	0.173
SINE1	0.170	0.194	**0.176**	**0.176**	0.251	0.329
MIXED	0.178	0.192	0.193	**0.191**	0.240	0.195
CIRCLES	0.191	**0.173**	0.182	0.178	0.184	0.186
RCV1	0.139	0.158	**0.142**	0.177	0.171	0.188
COVERTYPE	0.226	**0.221**	0.238	0.261	0.251	**0.221**
AIRLINE	0.388	0.378	0.377	0.379	0.378	**0.376**
ELECTRICITY	0.260	0.255	0.274	0.265	**0.248**	0.268
POWERSUPPLY	0.286	0.283	0.279	0.281	0.282	**0.276**

Table 15: Total average of misclassification rate - base learner: NB

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE
SEA0	0.062	0.056	0.055	0.083	**0.054**
SEA10	0.150	0.144	**0.143**	0.144	**0.143**
SEA20	0.239	0.235	**0.233**	**0.233**	0.234
SEA30	0.328	0.326	0.323	0.325	**0.324**
SEA-GRADUAL	0.154	**0.147**	0.148	**0.147**	**0.147**
HYPER-SLOW	0.146	0.118	0.110	0.131	**0.109**
HYPER-FAST	0.257	0.161	**0.148**	0.160	0.151
SINE1	0.166	0.203	0.176	**0.171**	0.238
MIXED	0.180	**0.193**	**0.193**	**0.193**	0.254
CIRCLES	0.187	**0.174**	**0.174**	0.177	0.178
RCV1	0.124	0.131	0.138	**0.130**	0.150
COVERTYPE	0.311	0.314	**0.310**	0.311	0.312
AIRLINE	0.386	0.379	0.377	0.378	**0.376**
ELECTRICITY	0.274	**0.259**	0.263	0.262	**0.259**
POWERSUPPLY	0.284	**0.278**	**0.278**	**0.278**	0.284

Tree (HAT) (Bifet & Gavaldà, 2009). HAT continually maintains the starting HT throughout time, but reacts to drift by swapping out subtrees based on an internal drift detection module. This represents a more granular approach to adapting to drift, compared to replacing the entire model in traditional drift detection. We use the implementation of HAT available in scikit-multiflow using their default hyperparameters (Montiel et al., 2018).

The results when using HT are shown in Table 14 for the setting where the computation available to each algorithm is divided among all its models. Note, however, that HAT is a single-pass algorithm, and only used half the number of available update steps that each other algorithm used. We observe that DriftSurf is an effective algorithm for this base learner, too. As we saw with other base learners, DriftSurf performs especially well on RCV1 and CoverType where it quickly switches to a new model after the drift and concentrates its processing power on the new model, but that DriftSurf loses to the large ensemble that AUE has on the continually drifting Hyperplane datasets. We observe that HAT's granular adaptation to drift compares favorably on the real datasets of Airline and Powersupply, but suffers on the sharpest drifts like in SINE1 and RCV1. For the NB base learner, the results are shown in Table 15. With NB, there is no advantage to repeated sampling of earlier visited points, and so we compare each algorithm as they run a single pass over the data. In this setting, AUE's ensemble of 10 experts is the overall best-performer, using more computation than DriftSurf and MDDM, although AUE was the slowest at adapting to the sharp drifts on SINE1 and RCV1.

D.8. Recovery Time Analysis

In this paper, we defined *recovery time* as the number of time steps after a drift before switching to a model trained only over the new distribution. However, an alternative metric for recovery we consider in this section is the 95%-recovery-time metric proposed by (Shaker & Hüllermeier, 2015). The 95%-recovery-time is the length of a time interval called the recovery phase. The recovery phase starts when the predictive model's performance drops below 95% of the performance attained over the first distribution, and the recovery phase ends when the model's performance recovers up to 95% of the performance curve that is ultimately obtained over the new distribution.

Table 16 shows the 95%-recovery-time over each dataset. When there is more than one drift in a dataset, the reported recovery time is the average over all the drifts. Note that no recovery time is reported for Hyperplane datasets because they contain a continually gradual drift that last throughout the entire stream, and the metric is not defined for such drifts.

The recovery time in some cases is reported to be 0. This is because the performance never dropped below 95% of the performance curve over the original distribution. Also, it is worth mentioning that the recovery time of Aware is regularly longer the other algorithms (while under our definition of recovery, Aware recovers immediately by resetting the model at the time of drift). This occurs because restarting from scratch after a drift means that Aware always enters the recovery phase, while the other algorithms may not enter the recovery phase if their performances did not drop below 95% of that under the original distribution.

We observe that DriftSurf overall performs well on the 95%-recovery-time metric. On RCV1 and CoverType, DriftSurf outperforms AUE which takes several time steps to shift its weight towards the newer models. Furthermore, on CoverType, DriftSurf outperforms MDDM, which suffers significant false positives throughout the stream.

Table 16: 95%-recovery-time ($\rho = 2m$ for each model)

DATASET	Aware	DriftSurf	StandardDD	MDDM-G	AUE
SEA0	10.67	**0.33**	3.67	**0.33**	**0.33**
SEA10	9.33	**0.33**	**0.33**	6.00	**0.33**
SEA20	5.00	**0.00**	**0.00**	4.33	**0.00**
SEA30	2.33	**0.33**	**0.33**	3.00	**0.33**
SEA-GRADUAL	4.50	**0.00**	**0.00**	7.00	**0.00**
SINE1	2.75	**3.00**	3.25	**3.00**	3.50
MIXED	1.00	**1.00**	**1.00**	**1.00**	2.00
CIRCLES	1.00	1.00	1.00	**0.83**	1.00
RCV1	10.00	**8.50**	11.50	9.50	13.50
COVERTYPE	3.50	**4.00**	**4.00**	19.50	9.00
AIRLINE	12.50	**10.50**	**10.50**	**10.50**	**10.50**
ELECTRICITY	1.00	**0.00**	**0.00**	**0.00**	**0.00**
POWERSUPPLY	3.00	1.33	**1.00**	2.00	1.67
AVG	5.12	**2.33**	2.81	5.15	3.24