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 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

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 $t = 1, 2, \dots$, 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	= $\lvert\mathbf{X}_t\rvert$, 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:

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

where $\mathcal{H}(n) = hn^{-\alpha}$, for a constant h and $1/2 \le \alpha \le 1$. 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 $t_{d_1}, t_{d_2}, \dots$ 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

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 ← 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 ← 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 ← 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
```

is executed when DriftSurf is in the stable state, and Algorithm 2, which is executed when DriftSurf is in the reactive 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:

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

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:

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

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 $\frac{1}{2} \le \alpha \le 1$, 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 $c_4 = (2c_3 - 8)c_1^2 p_s^2 + 6c_1 p_s$ (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).*

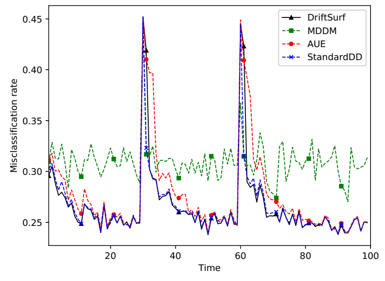
At a high level, ϵ_r and ϵ_s, respectively, capture the error rates in false negatives in drift detection and false positives in the stationary period afterwards. The full proof is in Appendix B.2.

6. Experimental Results

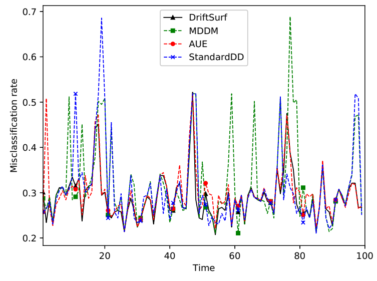
In this section, we present experimental results on datasets with drifts that (i) empirically confirm the advantage of DriftSurf's stable-state / reactive-state approach over Standard Drift Detection (StandardDD), (ii) empirically confirm the risk-competitiveness of DriftSurf with Aware, and (iii) show 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.

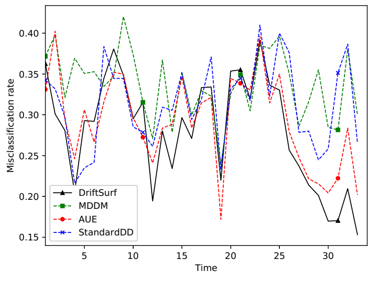
In our experiments, a batch of data points arrives at each time step. We first evaluate the performance of each algorithm by measuring the misclassification rate over this batch, and then each algorithm gains access to the labeled data to update their model(s); i.e., test-then-train. The base learner 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

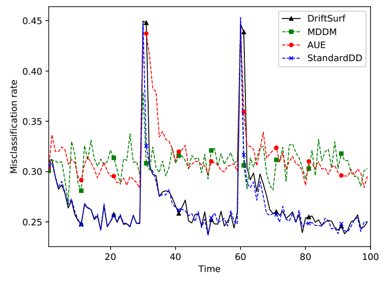


Figure 2: CoverType (update steps divided among each model)

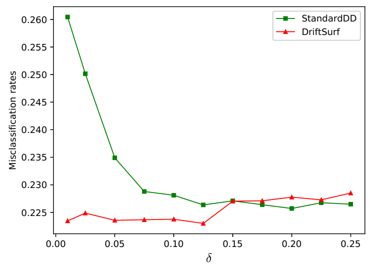


Figure 3: All datasets, DriftSurf and StandardDD under varying threshold δ

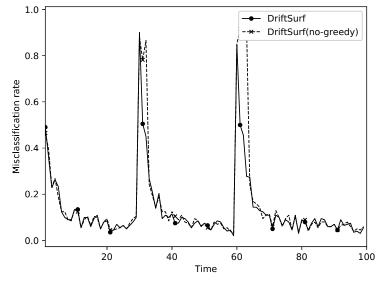


Figure 4: RCV1, DriftSurf and DriftSurf (no-greedy)

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.