Large-Scale Machine Learning over Streaming Data

Ellango Jothimurugesan

CMU-CS-22-150

December 2022

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under award numbers 1725663, 1919223, and 2211882, and the U.S. Army Contracting Command under award number W911NF20D0002. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: machine learning, streaming, concept drift, federated learning

Abstract

This thesis introduces new techniques for efficiently training machine learning models over continuously arriving data to achieve high accuracy, even under changes in the data distribution over time, known as concept drift. First, we address the case of IID data with STRSAGA, an optimization algorithm based on variance-reduced stochastic gradient descent that can incorporate incrementally arriving data and efficiently converges to statistical accuracy. Second, we address the case of non-IID data over time with DriftSurf. Previous work on drift detection generally rely on threshold parameters that are difficult to set, making them less practical without prior knowledge of the magnitude and rate of change. DriftSurf improves the robustness of traditional drift detection tests through a stable-state/reactive-state process, and attains higher statistical accuracy whenever an efficient optimizer like STRSAGA is used. Third, we address the case of non-IID data both over time and distributed in space in the federated learning setting with FedDrift. We empirically show that previous centralized drift adaptation and previous personalized federated learning methods are ill-suited under staggered drifts. FedDrift is the first algorithm explicitly designed for both dimensions of heterogeneity, and accurately identifies distinct concepts by learning a time-varying clustering, which enables collaborative training despite drifts. We show the presented algorithms are effective through theoretical competitive analyses and experimental studies that demonstrate higher accuracy on benchmark datasets over the prior state-of-the-art.

Acknowledgments

I am grateful for the guidance and support that I received from all of the faculty, staff, and fellow students whom I was fortunate to work with in the course of my graduate studies. I thank Phil Gibbons who advised me throughout this journey; it has been an incredible opportunity to learn from him, both in computer science and character. I thank all of my hard-working and talented collaborators who made this research possible, with special thanks to Ashraf Tahmasbi whom I worked closely with throughout. I thank Srikanta Tirthapura and Goce Trajcevski for their mentorship. Finally, I thank my thesis committee for their suggestions and feedback.

Contents

1 Introduction **1**

2 Preliminaries **5**

3 Learning from IID data with STRSAGA **9**
3.1 Related Work 10
3.2 Algorithm 11
3.2.1 Analysis 12
3.2.2 Proof of Lemma 3 15
3.3 Competitive Analysis of STRSAGA on Specific Arrival Distributions 16
3.3.1 Additional Proof Details 19
3.4 Experimental Setup 22
3.4.1 Datasets and Parameters 23
3.5 Experimental Results 24

4 Learning from non-IID data over time with DriftSurf **29**
4.1 Related Work 30
4.2 Problem Setup 31
4.3 Algorithm 32
4.4 Analysis 35
4.4.1 Stationary Environment 36
4.4.2 In Presence of Abrupt Drifts 37
4.5 Proofs 37
4.5.1 In a Stationary Environment 38
4.5.2 In Presence of Abrupt Drifts 41
4.6 Experimental Setup 45
4.6.1 Algorithms Evaluated 45
4.6.2 Datasets 47
4.6.3 Training and Hyperparameters 49
4.7 Main Experimental Results 50
4.8 Additional Experimental Results 53
4.8.1 Equal Computational Power for Each Model 54
4.8.2 Equal Computational Power for Each Algorithm 57
4.8.3 Sensitivity Analysis of δ 62

4.8.4 Evaluation of Greedy Reactive State 63
4.8.5 Comparison to 1PASS-SGD and Oblivious 65
4.8.6 Using SGD as the Update Process 65
4.8.7 Using Hoeffding Trees and Naive Bayes as Base Learners 66
4.8.8 Recovery Time Analysis 67

5 Learning from non-IID data over time and distributed in space with FedDrift **71**
5.1 Related Work 72
5.2 Problem Setup 73
5.3 Multiple-Model Training in FL 74
5.4 Clustering Algorithms 76
5.4.1 Special Case: One New Concept at a Time 76
5.4.2 General Case 77
5.5 Experimental Setup 79
5.5.1 Datasets 81
5.5.2 Experimental Parameters 82
5.6 Main Experimental Results 85
5.7 Additional Experimental Results 87

6 Conclusion 97

Bibliography 99

List of Figures

1.1 Simplistic drifts studied in prior work. (left) Simultaneous timing. (right) One majority concept. 2
1.2 Distributed drift pattern (2 concepts). 2
1.3 Distributed drift pattern (4 concepts). 2
1.4 Class distribution over time in FMoW. The drift viewed globally (left) is small relative to the localized drift for Africa (right). 3

3.1 Depiction of the errors assuming that an algorithm A is c-risk-competitive to DYNASAGA(ρ). The total error of DYNASAGA(ρ) is at least the statistical error bound $\mathcal{H}(t_i^D)$, and the sub-optimality of A is at most a factor of c of $\mathcal{H}(t_i^D)$. . . . 14
3.2 Sub-optimality under skewed arrivals with $M = 8\lambda$. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs. 24
3.3 Sensitivity analysis. The first plot varies the skew M/λ for a fixed processing rate ρ/λ, and the second two plots vary the processing rate for a fixed skew. Results are plotted for time steps $i = 25, 50, 75, 100$ over a stream of the RCV dataset of 100 time steps. The median is taken over 9 runs. 25
3.4 Sub-optimality under Poisson arrivals with mean λ. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs. 26
3.5 Test loss under skewed arrivals with $M = 8\lambda$. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs. 27
3.6 Test loss under Poisson arrivals with mean λ. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs. 27
3.7 Additional plots of sub-optimality for MovieLens100k. 28
3.8 Additional plots of test loss for MovieLens100k. 28

4.1 A drift happens at time t_d. DriftSurf recovers by time t_r. The current predictive model is initialized at time t_e. 44
4.2 Misclassification rate over time for CoverType, PowerSupply, and Electricity . . 51
4.3 CoverType (update steps divided among each model) 51
4.4 All datasets, DriftSurf and StandardDD under varying threshold δ 51
4.5 RCV1, DriftSurf and DriftSurf (no-greedy) 51
4.6 Misclassification rate over time ($\rho = 2m$ for each model) 55
4.7 Average misclassification rate for SEA dataset with different levels of noise . . . 56
4.8 CoverType dataset, comparing different drift detectors ($\rho = 2m$ for each model) . 57

4.9 Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm) comparing Aware, DriftSurf, AUE, and MDDM 59
4.10 Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm) comparing Aware, DriftSurf, AUE ($k = 2$), and Condor 60
4.11 Misclassification rate over time ($\rho = 2m$ divided among all models of each algorithm) 61
4.12 δ-sensitivity ($\rho = 2m$ for each model) 64
4.13 Misclassification rate over time ($\rho = 2m$ for each model) 65

5.1 Simplistic drifts studied in prior work. (left) Simultaneous timing. (right) One majority concept. 71
5.2 Distributed drift pattern (2 concepts). 71
5.3 Distributed drift pattern (4 concepts). 71
5.4 Class distribution over time in FMoW. The drift viewed globally (left) is small relative to the localized drift for Africa (right). 80
5.5 Average accuracy of each drift detection-based algorithm under varying thresholds δ. 84
5.6 Accuracy at each time (averaged across clients) on CIRCLE-2. 86
5.7 The clustering learned by FedDrift on MNIST-4. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept. 86
5.8 The clustering learned by CFL-W on SINE-2. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept. 91
5.9 The clustering learned by FedDrift-Eager on MNIST-4. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept. 91
5.10 The accuracy-communication trade-off on MNIST-4 for FedDrift-Eager, FedDrift, and FedDrift-C. Each algorithm is evaluated under various selections of the splitting/merging threshold δ between 0.02 and 0.20, indicated by color. The vertical axis is the average test accuracy across clients and time, omitting drifts. (1 trial) . 93
5.11 The clustering learned on SINE-2 when $\delta = 0.01$. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept. 93
5.12 Test accuracy of selected algorithms at each time on SINE-2, SEA-2, MNIST-2, SEA-4, MNIST-4, and FMoW. Vertical lines represent standard deviations. . . . 96

List of Tables

3.1 Datasets for logistic regression 23
3.2 Datasets for matrix factorization 23

4.1 Commonly used symbols pertaining to DriftSurf 31
4.2 Basic statistics of datasets 47
4.3 Details of drifts in datasets 49
4.4 Hyperparameters and batch sizes 50
4.5 Average misclassification rate (equal number of update steps for each model) . . 52
4.6 Average misclassification rate (update steps divided among each model) 53
4.7 Average misclassification rate ($\rho = 2m$ for each model) 54
4.8 Average misclassification rate ($\rho = 4m$ divided among all models of each algorithm) 58
4.9 Average misclassification rate ($\rho = 2m$ divided among all models of each algorithm) 58
4.10 Average misclassification rate - DriftSurf vs DriftSurf (no-greedy) ($\rho = 2m$ for each model) 63
4.11 Average misclassification rate - update process: SGD ($\rho = 2m$ for each model) . 66
4.12 Average misclassification rate - update process: SGD ($\rho = 2m$ divided among all models of each algorithm) 67
4.13 Average misclassification rate - base learner: HT ($\rho = 2m$ divided among all models of each algorithm) 68
4.14 Average misclassification rate - base learner: NB 69
4.15 95%-recovery-time ($\rho = 2m$ for each model) 69

5.1 Commonly used symbols pertaining to FedDrift 74
5.2 Training parameters 82
5.3 Average test accuracy (%) across all clients and time, omitting drifts (5 trials) . . 87
5.4 Average test accuracy (%) across clients and time, omitting drifts (5 trials), extended results 89
5.5 Average test accuracy (%) across clients and time, including drifts (5 trials), extended results 90
5.6 Average test accuracy (%) under random drift patterns, omitting drifts (5 trials) . 94
5.7 Average test accuracy (%) under random drift patterns, including drifts (5 trials) . 94

Chapter 1

Introduction

We consider streaming data arrivals that continually arrive over time. From these training data, our goal is to learn a model with highly *accurate* predictions at each time on unlabeled test data. Periodically recomputing the model from scratch (i.e., batch retraining) upon the arrival of new data points incurs significant computational cost. We seek techniques for *efficient* incremental training of the model as data arrive. Furthermore, the model training should be *drift-robust*, adapting to unexpected changes in the distribution of data over time.

Oblivious algorithms presume that data are drawn from a stable underlying distribution, and that training data in the past are sufficiently similar to the test data in the future. But in real-world data, the distribution is often non-stationary. User sentiment and preferences can change drastically due to external environments such as the pandemic and macroeconomics [31, 52]. Data collected by passive devices such as cameras also experience various data changes due to unexpected weather or novel objects [5, 84]. Without adaptation to concept drifts, model accuracy degrades over time.

Adaptive algorithms can typically be classified into three major categories: weighted sampling, ensembles, and drift detection [30], each representing different trade-offs among the desired criteria of accuracy, efficiency, and drift-robustness. A comprehensive review can be found in §4.1, but here we give a brief overview. Weighted sampling techniques focus the training on more recent data points, and are guaranteed to adapt to drifts, but their myopic strategy suffers from low statistical accuracy in the absence of drift, making them less suitable for general application. Ensemble methods consist of both long-trained models that have high accuracy in the absence of drift and newer models that have high accuracy after drift, but the maintenance of a full ensemble may be inefficient in practice when training large models such as deep neural networks (or come at a penalty in accuracy if training time is divided among the models).

The third category of adaptive algorithms is drift detection, which track 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 replaced with a model trained solely on the data going forward. Perfect drift detection would enable efficient training for a model with high prediction accuracy both in the presence and absence of drifts. The challenge is that drift detection is not robust to the variety of patterns, magnitudes, and rates of change encompassed by concept drifts in practice. Different drift detection tests are preferred depending on whether a drift is abrupt or gradual, and the user-defined threshold

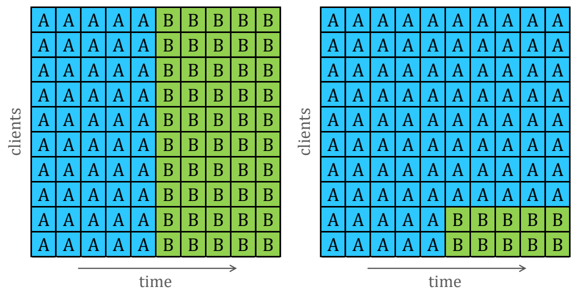


Figure 1.1: Simplistic drifts studied in prior work. (left) Simultaneous timing. (right) One majority concept.

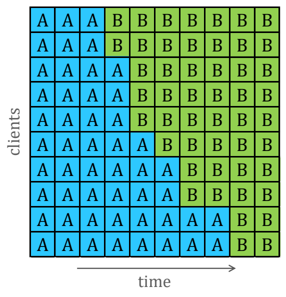


Figure 1.2: Distributed drift pattern (2 concepts).

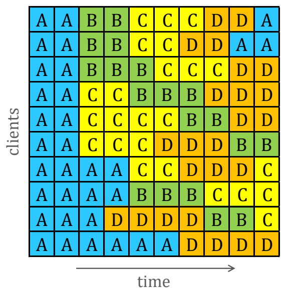


Figure 1.3: Distributed drift pattern (4 concepts).

parameter in most tests governs a trade-off between the detection accuracy and speed [30]. But concept drift is inherently unpredictable [1], and with no prior knowledge of the nature of change, choosing the right tests and parameters is hard (as demonstrated in §4.8). Getting it wrong comes at 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.

In addition to consideration of the classical drift adaptation problem in a centralized learning setting, this thesis initiates a general study of distributed concept drifts in the Federated Learning (FL) setting. FL is a popular machine learning paradigm that enables collaborative training without sharing the raw training data generated at each individual client. Concept drift in FL poses new fundamental challenges when data is heterogeneous both over time and across different clients. When different clients experience the drift at different times, no single global model can perform well for all clients. Similarly, when multiple concepts exist simultaneously, no centralized training decision works well for all clients.

Several recent works have recognized the problem of FL under concept drift but only under restrictive settings such as (i) drifts occurring simultaneously in time (e.g., Figure 1.1(left)) by centrally tuning the learning rate [15], or (ii) drifts with only minor deviations from a majority concept (e.g., Figure 1.1(right)) by tuning regularization to suppress updates from drifting clients [18, 62].

This work is the first to study more general settings arising in distributed drifts. Consider the distributed drift pattern depicted in Figure 1.2. This is representative of an emerging trend (e.g., a breaking news event) that affects different clients at different times (e.g., due to their lag in learning of the news). For example, consider a next word prediction app in the period when "war" emerges as the popular next word after "Ukraine" or "slap" emerges after "Will Smith". Even for this simple case of a single staggered transition between two concepts, all classes of centralized drift adaptation (weighted sampling, ensembles, and drift detection) and prior FL techniques suffer in accuracy (as demonstrated in §5.6).

We also consider more challenging cases, as depicted in Figure 1.3, where multiple concepts emerge at the same time and concept drifts may be recurring (a.k.a. periodic).

The challenge of distributed drift in real-world data is exemplified in the Functional Map

[1]Distribution changes that are predictable like seasonal variations may be accounted for with additional featurization.

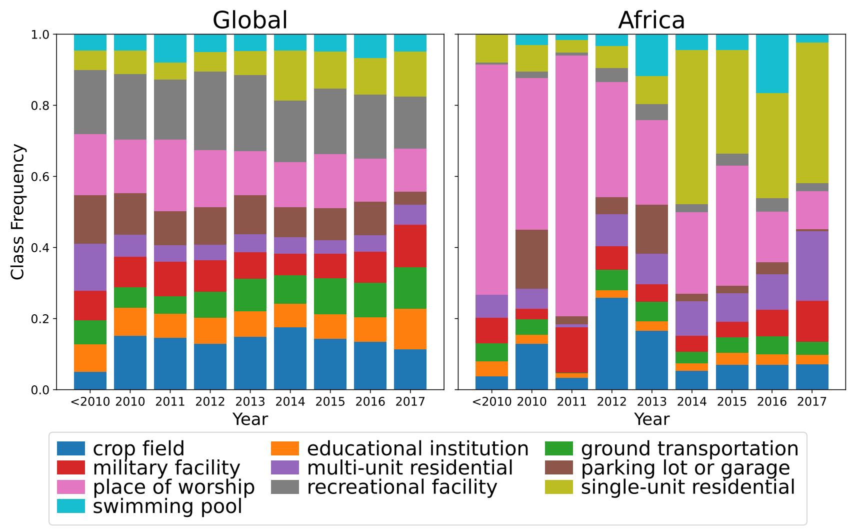


Figure 1.4: Class distribution over time in FMoW. The drift viewed globally (left) is small relative to the localized drift for Africa (right).

of the World (FMoW) dataset adapted from the WILDS benchmark [20, 52]. The task is to classify the building type or land use from a satellite image, where images are over 5 major geographical regions (Africa, Americas, Asia, Europe, and Oceania) and across 16 years. Concept drift due to human activity and environmental processes degrades predictive accuracy over time. For the 10 most common classes, Figure 1.4 shows how the class distribution in Africa changes more rapidly over time, such as a reduction in places of worship and an increase in single-unit residential buildings. However, the class distribution viewed globally is relatively slow-changing. Our evaluation shows that the model trained on the global dataset only achieves 48% accuracy on Africa after the major drift at 2014, compared to 66% on the rest of the world. This real-world example highlights the necessity to mitigate concept drift differently across regions, and existing centralized solutions cannot address this fundamental challenge.

Given the aforementioned challenges with prior drift adaptation, machine learning practitioners may still resort to periodically retraining their models, which comes at high computational cost and suffers from accuracy degradations from drifts between retraining (or even suffers perpetually under some distributed drifts). This thesis takes a step towards ending that practice.

Thesis Statement: The efficient training of models that maintain high accuracy even in the presence of drifts can be achieved via an incremental model update strategy, which attains risk comparable to offline learning, while mitigating the uncertainty of traditional drift detection through a stable-state/reactive-state process and hierarchical clustering, which accurately identify drifts staggered across time and space.

The organization of this thesis is as follows.

- Chapter 2 introduces common notation and terminology, defines the objective of high prediction accuracy at each time, and sets up the framework for competitive analysis by introducing the definition of *risk-competitive*.
- Chapter 3 considers the case when all data points in the stream are independent and

identically distributed (IID), and introduces the algorithm STRSAGA. While the IID case is of independent interest for streams known to not contain drifts, it is also an important building block for the subsequent chapters on drift adaptation—given the accurate detection of changepoints, a suffix of the stream of can be identified as data from a (mostly) common concept. The challenge is that the training data points are only observed sequentially and the optimization objective of empirical risk minimization over the expanding set of points is a moving target. Overcoming this challenge, STRSAGA efficiently incorporates arriving data points into the model it maintains, and risk-competitive analysis shows that STRSAGA achieves comparable accuracy to that of an offline algorithm that has access to all training data points in advance. Furthermore, even when data points do not arrive at a steady rate, STRSAGA remains risk-competitive under common arrival distributions, including Poisson arrivals and many classes of skewed arrivals. Experimental results support these analytical findings, and also show that STRSAGA outperforms natural streaming data versions of both SGD and SSVRG.

- Chapter 4 considers non-IID data arrivals over time, and introduces the algorithm DriftSurf. While traditional drift detection may be brittle under a variety of types of concept drift encountered in practice, DriftSurf robustly extends drift detection by its incorporation into a broader stable-state/reactive-state process. DriftSurf accounts for the uncertainty in the application of drift detection by using the trigger to transition to a reactive state, which reacts quickly to true drifts while eliminating most false positives. The advantage to this approach is that a small threshold can be universally applied (corresponding to aggressive drift detection) to achieve a high detection rate, while mitigating the false positive rate of standalone drift detection. Risk-competitive analysis shows that DriftSurf attains comparable accuracy to an idealized algorithm with oracle access to when drift occurs (permitting that both algorithms are trained accurately; e.g., using STRSAGA). Experimental results show that DriftSurf generally outperforms a state-of-the-art drift-detection-based method MDDM and a state-of-the-art ensemble methods AUE, while operating at computational efficiency comparable to standalone drift detection.
- Chapter 5 considers non-IID data arriving over time and distributed in space in the FL setting, and introduces the algorithm FedDrift. To adapt robustly to a broader variety of distributed drift patterns than considered in previous work, FedDrift solves a time-varying clustering problem to learn from data generally heterogeneous in both dimensions. FedDrift draws upon ideas from DriftSurf to manage the uncertainty of drift detection. Local detection is applied to quickly isolate drifted clients and hierarchical clustering is leveraged to slowly and safely merge clients once common concepts are confidently identified. The hierarchical clustering process in FedDrift also mitigates false positives in detection (which could occur frequently when multiplied across clients). Experimental results show that FedDrift achieves similar accuracy to an idealized algorithm with oracle access to when drift occurs for each client, and that it outperforms state-of-the-art FL algorithms Adaptive-FedAvg, IFCA, and CFL, and state-of-the-art centralized adaptive algorithms AUE and KUE.
- Chapter 6 concludes the thesis and lays out open problems and directions for future work.

Chapter 2

Preliminaries

This chapter defines symbols and terms used in the context of the centralized learning setting considered in Chapters 3 and 4. The notation for the extension to data and models across multiple clients in the FL setting differs and is deferred to Chapter 5 where it is first used.

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. 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 $n_{t_1,t_2} = |\mathcal{S}_{t_1,t_2}|$. For the case $t_1 = 1$, we omit the first subscript; e.g., $\mathcal{S}_{t_2}$ and n_{t_2}.

Each $\mathbf{X}_t$ consists of data points drawn from a distribution (concept) 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. Drift is categorized as either *abrupt* when change occurs across a single time step, or *gradual* for a transition over multiple time steps [30].

The model being trained is a member of a class of functions $\mathcal{F}$. A function in this class is parameterized by a vector of weights $\mathbf{w} \in \mathbb{R}^d$. Our goal is to learn a model with high prediction accuracy at each time step t; i.e., 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 $\mathbf{w}$ at point $\mathbf{x}$. Thus, the objective at each time t is:

$$\min_{\mathbf{w}_t \in \mathcal{F}} \mathbb{E}_{\mathbf{x}\sim I_t}[f_{\mathbf{x}}(\mathbf{w}_t)]. \tag{2.1}$$

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 $\mathcal{R}^*_{\mathcal{S}} = \mathcal{R}_{\mathcal{S}}(\mathbf{w}^*_{\mathcal{S}})$.

In order to quantify the error in the expected risk from empirical risk minimization, we use a uniform convergence bound [10, 11]. 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:

$$\mathbb{E}[\sup_{\mathbf{w}\in\mathcal{F}} |\mathcal{R}_I(\mathbf{w}) - \mathcal{R}_{\mathcal{S}}(\mathbf{w})|] \leq \mathcal{H}(n)/2 \tag{2.2}$$

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 [11].

Following Bottou and Bousquet [11], we define *sub-optimality* as follows.

Definition 1. *The sub-optimality of an algorithm A over training data $\mathcal{S} = \mathcal{S}_{t_1,t_2}$ is the difference between A's empirical risk and the optimal empirical risk:*

$$\texttt{SUBOPT}_{\mathcal{S}}(A) := \mathcal{R}_{\mathcal{S}}(\mathbf{w}) - \mathcal{R}_{\mathcal{S}}(\mathbf{w}^*_{\mathcal{S}})$$

where $\mathbf{w}$ is the solution returned by A on $\mathcal{S}$ and $\mathbf{w}^*_{\mathcal{S}}$ is the empirical risk minimizer over $\mathcal{S}$.

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 $\mathcal{F}$. (The total error for $\mathcal{F}$ of a function $\mathbf{w}$ learned from a sample $\mathcal{S}$ drawn from a distribution I refers to $\mathbb{E}[\mathcal{R}_I(\mathbf{w})] - \inf_{\mathbf{w}' \in \mathcal{F}} \mathcal{R}_I(\mathbf{w}')$.) Therefore, we focus on reducing the sub-optimality to balance with $\mathcal{H}(n)$—there is no guaranteeable benefit to minimizing the empirical risk further. Note that although $\mathcal{H}(n)$ is only an upper bound on the statistical error, Bottou and Bousquet remark "it is often accepted that these upper bounds give a realistic idea of the actual convergence rates" [11].

Given this principle, we define our notion of *risk-competitiveness* as follows.

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

$$\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(A)\right] \leq c\mathcal{H}(n_B)$$

where t_d is the time step of the most recent drift (said to be 1 if there are no drifts) and $n_B \leq n_{t_d,t}$ is the total number of data points read by algorithm B, all from $\mathcal{S}_{t_d,t}$.

The above definition in terms of a general algorithm B will be made explicit in §3.2.1 and §4.4 for particular instances of an algorithm B. For example, in the case of IID data arrivals, c-risk-competitiveness to the empirical risk minimizer as algorithm B at time t simplifies to $\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_t}(A)\right] \leq c\mathcal{H}(n_t)$. Another example is in the case of the stationary period after a concept drift, c-risk-competitiveness to an algorithm with oracle knowledge of when the drift occurred is the bound $\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_{t_d,t}}(A)\right] \leq c\mathcal{H}(n_{t_d,t})$. In both cases, the expected sub-optimality is bounded within a constant factor of our stated goal of the statistical error bound $\mathcal{H}()$.

The motivating idea in the definition is that if we believe $\mathcal{H}()$ to be a tight bound on the statistical error, and hence, a lower bound on the total error, then $\mathcal{H}(n_B)$ is a lower bound on the error of algorithm B, and thus constant factor risk-competitiveness implies the ratio of errors of algorithm A to algorithm B is constant.

Assumptions and Limitations Finally, we mention some broad assumptions and limitations in the study of algorithms in this thesis.

- We consider algorithms that, at each time t, have access to all the data points so far $\mathcal{S}_{1,t}$. In contrast to single-pass online learning, our setting permits resampling relevant older data, enabling higher accuracy than what can be achieved in a single pass. For this reason, we use risk-competitive analysis rather than the classic regret analysis in online learning [1]. Of course, unlimited memory is a simplifying assumption and memory is eventually bounded. More realistically, the model can access some window of the most recent W time steps of data, $\mathcal{S}_{t-W,t}$, and the risk-competitive analyses presented in this thesis can be readily generalized with reference to this windowed stream segment.

[1] Since our initial conference publication of STRSAGA, regret analysis has since been performed by other authors [44] for the online learning setting.

- We seek high prediction accuracy at each time (Eq 2.1), yet our high-probability risk-competitive results are established asymptotically, providing no guarantee at some time steps. Theoretical analyses are supplemented with experimental studies that present the average accuracy over time, which is a natural metric, but may not be the most appropriate in some applications. For instance, it could be preferable to have a stable accuracy that is lower on average but with a better worst-case compared to a system with a higher average accuracy but is susceptible to catastrophic costs at some times. Evaluation under other metrics are considered briefly (§4.8.8), but not comprehensively studied in this thesis.
- Our criteria (§1) are accuracy, efficiency, and drift-robustness. Risk-competitive analysis establishes high accuracy, with respect to stationary periods after drifts (and can be with respect to a budget of gradient computations as in §3.2.1). To consider the trade-offs between the desired criteria, experimental studies are indispensable. For algorithms that train multiple models and incur higher cost, empirical results are also presented normalized for training time or communication. Furthermore, while the definition of risk-competitive is valid for drifts both abrupt and gradual, the guarantee on accuracy only after drift renders it less interesting for slow, gradual drifts. Our experimental study measures accuracy over all time across datasets encompassing a variety of drift magnitudes, rates, and patterns, and we characterize drift-robustness as achieving consistent performance across datasets and with low sensitivity of hyperparameters. We choose datasets for evaluation to match those considered by papers on prior state-of-the-art algorithms for fair comparisons, and assume that high performance over datasets studied in the literature corresponds to practicality.

Chapter 3

Learning from IID data with STRSAGA

This chapter considers the maintenance of a model over IID data arrivals. At any point in time, the goal is to fit the model to the training data points observed so far, in order to accurately predict the labels of unobserved test data. Such a model is never "complete" but instead needs to be continuously updated as newer training data points arrive.

We refer to algorithms that assume the set of training data are all available in advance as *offline algorithms*. Upon the arrival of new data points, offline algorithms must recompute the model from scratch, which is infeasible due to high computational costs. We refer to algorithms that can efficiently update the model as more data arrive as *streaming data algorithms*. Such efficiency should not come at the expense of accuracy—ideally, the accuracy of the model maintained through such updates should be close to that obtained if we were to build a model from scratch using all the training data points seen so far.

Fitting a model $\mathbf{w}$ is usually cast as optimization problem. For the stream of training data points $\mathcal{S}_i$ of n_i data points observed through time i, the goal is to minimize the empirical risk $\mathcal{R}_{\mathcal{S}_i}(\mathbf{w}) = \frac{1}{n_i}\sum_{\mathbf{x}\in\mathcal{S}_i} f_{\mathbf{x}}(\mathbf{w})$, where $f_{\mathbf{x}}$ is the loss of $\mathbf{w}$ at data point $\mathbf{x}$. In modern machine learning, Stochastic Gradient Descent (SGD) [9, 75] methods are widely used for optimization. Variance-reduced SGD such as SVRG [45] and SAGA [23] have emerged more recently and attracted interest for their faster convergence rates. These algorithms operate by iteratively sampling a data point from the set of training data and using the gradient of its loss to determine an update direction.

For variance-reduced SGD, simply expanding this set from which points are sampled to include all new arrivals can result in poor convergence (discussed in §3.2), depending on the number of arrivals. Thus, even though these algorithms use only a single data point at a time, they still assume a fixed set of training data, and are still offline algorithms.

The optimization goal of a streaming data algorithm is to maintain a model using all the data points that have arrived so far, such that the model's empirical risk is close to the empirical risk minimizer (ERM) over those data points. The challenges include (i) because the training data is changing at each time step, the ERM on streaming data is a "moving target"; (ii) the ERM is an optimal solution that cannot be realized in limited processing time, while a streaming data algorithm is not only limited in processing time, but is also presented the data points only sequentially; (iii) with increasing arrival rates, it becomes increasingly difficult for the streaming data algorithm to keep up with the ERM; and (iv) data points may not arrive at a steady rate: the numbers of points arriving at different points in time can be highly skewed.

This chapter presents STRSAGA (§3.2), a streaming data algorithm that overcomes these challenges. Risk-competitive analysis compares the accuracy of STRSAGA to a state-of-the-art offline algorithm DYNASAGA [21] for variance-reduced SGD, which is given all the data points in advance. We show that given the same computational power, the accuracy of STRSAGA is competitive to DYNASAGA at each point in time, under certain conditions on the schedule of input arrivals which we lay out in §3.3. We show that these conditions are satisfied by a number of common arrival distributions, including Poisson arrivals and many classes of skewed arrivals.

Experimental results (§3.5) for two machine learning tasks, logistic regression and matrix factorization, on two real data sets each, support our analytical findings: the sub-optimality of STRSAGA on data points arriving over time (under various input arrival distributions) is almost always comparable to the offline algorithm DYNASAGA, when each algorithm is given the same computational power. We also show that STRSAGA significantly outperforms natural streaming data versions of both SGD and SSVRG [27].

3.1 Related Work

Stochastic Gradient Descent (SGD) [75] and its extensions are used extensively in practice for learning from large datasets. While an iteration of SGD is cheap relative to an iteration of a full gradient method, its variance can be high. To control the variance, the learning rate of SGD must decay over time, resulting in a sublinear convergence rate.

The previous decade has seen a breakthrough with variance-reduced versions of SGD that achieve linear convergence on strongly convex objective functions, generally by incorporating a correction term in each update step that approximates a full gradient, while still ensuring each iteration is efficient like SGD. SAG [76] was among the first variance reduction methods proposed and achieves linear convergence rate for smooth and strongly convex problems. SAG requires storage of the last gradient computed for each data point and uses their average in each update. SAGA [23] improves on SAG by eliminating a bias in the update. Stochastic Variance-Reduced Gradient (SVRG) [45] is another variance reduction method that does not store the computed gradients, but periodically computes a full-data gradient, requiring more computation than SAGA. Semi-Stochastic Gradient Descent (S2GD) [54] is a variant of SVRG where the gaps between full-data gradient computations are of random length and follow a geometric law. CHEAPSVRG [79] is another variant of SVRG. In contrast with SVRG, it estimates the gradient through computing the gradient on a subset of training data points rather than all the data points. However, all of the above variance-reduced methods require $O(n \log n)$ iterations to guarantee convergence to statistical accuracy (to yield a good fit to the underlying data) for n data points. DYNASAGA [21] achieves statistical accuracy in only $O(n)$ iterations.

The main observation underlying DYNASAGA is that the linear convergence rate of variance-reduced SGD depends on the sample size n. Specifically for SAGA [23], assuming losses are μ-strongly convex and with L-Lipschitz continuous gradients, each iteration improves the objective by a factor of $\left(1 - \frac{\mu}{2(\mu n+L)}\right)$, which decreases in n. Given the higher optimization errors for larger sample sizes, DYNASAGA achieves fast convergence by using a controlled schedule to gradually increase its set of samples for SAGA iterations.

The above algorithms for variance-reduced SGD are offline, and assume the entire dataset is available beforehand. Streaming SVRG (SSVRG) [27] is an algorithm that handles streaming data arrivals, and processes them in a single pass through data, using limited memory. (Underlying their work too is that the convergence rate for SVRG varies in the number of samples.) In our experimental study, we found STRSAGA to be significantly more accurate than SSVRG. Further, our analysis of STRSAGA shows that it handles arrival distributions which allow for burstiness in the stream, while SSVRG is not suited for this case. In many practical situations, restricting a streaming data algorithm to use limited memory is overly restrictive and as our results show, leads to worse accuracy.

Since the initial conference publication of STRSAGA, other authors [69] have adapted STR-SAGA for multi-core parallelism, employing a lock-free strategy similar to Hogwild [73] that works well for sparse gradients. STRSAGA has also since been applied for local client training in the Federated Learning setting by other authors [44, 87], who also demonstrate the applicability of STRSAGA for training other models such as CNNs.

3.2 Algorithm

We present STRSAGA for updating a model with varianced-reduced SGD over data arriving over time. SGD and variance-reduced versions like SAGA [23] work by iteratively sampling a point from a training set T and using the gradient of its loss to determine an update direction. We let $\rho \geq 1$ denote the number of such iterations that can be performed in a single time step.

One option to handle streaming data arrivals is to simply expand the set T from which further sampling is conducted, by adding all the new arrivals. We consider this strategy for SGD in our evaluation in §3.5. However, for variance-reduced SGD, the optimization error increases in the size of T due to a slower convergence rate (as noted in §3.1). With uncontrolled increases in the size of T (due to either a high mean arrival rate or temporary bursts in the number of arrivals), the corresponding sub-optimality over T of the algorithm increases, so that the function that is finally computed may have poor accuracy.

To handle this, we use an idea from DYNASAGA [21], which increases the size of the training set T in a controlled manner, according to a schedule. Upon increasing the size of T, further increases are placed on hold until a sufficient number of SAGA steps have been performed on the current state of T. By using this idea, DYNASAGA was able to achieve statistical accuracy earlier than SAGA. However, DYNASAGA is still an offline algorithm that assumes that all training data is available in advance.

STRSAGA deals with streaming arrivals as follows. Arriving points from the next set of points $\mathbf{X}_i$ are added to a buffer `Buf`. The effective sample set T is expanded in a controlled manner, similar to DYNASAGA. However, instead of choosing new points from a static training set, such as in DYNASAGA, STRSAGA chooses new points from the dynamically changing buffer `Buf`. If `Buf` is empty, then available CPU cycles are used to perform further steps of SAGA. After any time step, it is possible that STRSAGA may have trained over only a subset of the points that are available in `Buf`, but this is to ensure that the optimization error on the subset that has been trained is balanced with the statistical error of the effective sample size. Algorithm 1 depicts the steps taken to process the zero or more points $\mathbf{X}_i$ arriving at time step i. Before any input is

Algorithm 1 STRSAGA: Process a set of training points $\mathbf{X}_i$ that arrived in time step i to update the model $\mathbf{w}$.

```
// $\rho$ is the number of update steps that can be performed
// $\eta$ is the learning rate
// $T_i$ is the effective sample set
$T_i \leftarrow T_{i-1}$
Add $\mathbf{X}_i$ to Buf {Buf is the set of training points not added to $T_i$ yet}
for $j \leftarrow 1$ to $\rho$ do
  if Buf is non-empty & $j$ is even then
    Move a single point, $p$, from Buf to $T_i$
    $\alpha(p) \leftarrow 0$ {$\alpha(p)$ is the prior gradient of $p$, initialized to $0$}
  else
    Sample a point $p$ uniformly from $T_i$
  end if
  $A \leftarrow \sum_{x \in T_i} \alpha(x)/|T_i|$ {$A$ 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
```

seen, the algorithm initializes buffer Buf to empty, effective sample T_0 to empty, and function $\mathbf{w}_0$ to random values. STRSAGA as described here uses the basic framework of DYNASAGA, of adding one training point to T_i every two steps of SAGA (the linear schedule in [21]), and both algorithms borrow variance-reduction steps from SAGA (lines 8-9 in Algorithm 1 and using the running average A of all gradients).

The time complexity of Algorithm 1 is on the order of ρ times the cost of a gradient computation. Storing the samples and model parameters requires $O(n_i D + d)$ space, where D is the dimension of the data and d is the dimension of the model. Storing the prior gradients $\alpha(p)$ for each data point incurs an additive $O(n_i d)$; for linear models, this cost is reduced to $O(n_i)$ since each gradient is a scalar multiple of the corresponding data point.

3.2.1 Analysis

Suppose data points $\mathcal{S}_i$ have been seen until time step i, and $n_i = |\mathcal{S}_i|$. We first note that the time taken to process a set of training points $\mathbf{X}_i$ is dominated by the time taken for ρ iterations of SAGA. Ideally, the empirical risk of the solution returned by STRSAGA is close to that of the ERM over $\mathcal{S}_i$. However, this is not possible in general. Suppose the number of points arriving at each time step i were much greater than ρ, the number of iterations of SAGA that can be performed at each step. Then not even an offline algorithm such as DYNASAGA that has all points at the beginning of time could be expected to match the ERM within the available time. In what follows, we present a competitive analysis, where the performance of STRSAGA is compared with that of an offline algorithm that has all data available to it in advance. We consider two offline algorithms, ERM and DYNASAGA(ρ), described below.

Algorithm ERM sees all of $\mathcal{S}_i$ at the beginning of time, and has infinite computational power to minimize the empirical risk. A streaming data algorithm has two obstacles if it has to compete with ERM: (i) Unlike ERM, a streaming data algorithm does not have all data in advance, and (ii) Unlike ERM, a streaming data algorithm has limited computational power. It is clear that no streaming data algorithm can do better than ERM. We can practically approach the performance of ERM through executing DYNASAGA until convergence is achieved.

Algorithm DYNASAGA(ρ) sees all of $\mathcal{S}_i$ at the beginning of time, and is given ρ iterations of gradient computations in each step. The parenthetical ρ denotes this algorithm is the extension of the original DYNASAGA [21], parameterized by the available amount of processing time. The algorithm DYNASAGA performs $2n_i$ steps of gradient computations on $\mathcal{S}_i$ and then terminates, while DYNASAGA(ρ) performs ρi steps, where if $\rho i > 2n_i$, the additional steps are uniformly over $\mathcal{S}_i$. The computational power of DYNASAGA(ρ) over i time steps matches that of a streaming data algorithm. However, DYNASAGA(ρ) is still more powerful than a streaming data algorithm, because it can see all data in advance. In general, it is not possible for a streaming data algorithm to compete with DYNASAGA(ρ) either—one issue being that streaming arrivals may be very bursty. Consider the extreme case when all of $\mathcal{S}_i$ arrives in the ith time step, and there were no arrivals in time steps 1 through $i-1$. An algorithm for streaming data has only ρ gradient computation steps that it can perform on n_i points, and its earlier $\rho(i-1)$ gradient steps had no data to use. In contrast, DYNASAGA(ρ) can perform ρi gradient steps on $\mathcal{S}_i$, and achieve a smaller empirical risk.

Each algorithm STRSAGA, DYNASAGA(ρ), and ERM, after seeing $\mathcal{S}_i$, has trained its model on a subset $T_i \subseteq \mathcal{S}_i$. We call this subset the "effective sample set". Let $t_i^{\mathsf{STR}}, t_i^D$, and t_i^{ERM} denote the sizes of the effective sample sets of STRSAGA, DYNASAGA(ρ), and ERM, respectively, after i time steps. The following lemma shows that the expected sub-optimality of DYNASAGA(ρ) over $\mathcal{S}_i$ is related to t_i^D.

Lemma 1 (Lemma 5 in [21]). *After i time steps, $t_i^D = \min\{n_i, \rho i/2\}$, and $t_i^{\mathsf{ERM}} = n_i$. The expected sub-optimality of* DYNASAGA*(ρ) over $\mathcal{S}_i$ after i time steps is $O(\mathcal{H}(t_i^D))$.*

Our goal is for a streaming data algorithm to achieve an empirical risk that is close to that of an offline algorithm; i.e., risk-competitiveness.

Definition 3. *For $c \geq 1$, a streaming data algorithm A is said to be c-risk-competitive to* DYNASAGA*(ρ) at time step i if $\mathbb{E}\left[\mathtt{SUBOPT}_{\mathcal{S}_i}(A)\right] \leq c\mathcal{H}(t_i^D)$. Similarly, A is said to be c-risk-competitive to* ERM *at time step i if $\mathbb{E}\left[\mathtt{SUBOPT}_{\mathcal{S}_i}(A)\right] \leq c\mathcal{H}(n_i)$.*

Note that the expected sub-optimality of A is compared with $\mathcal{H}(t_i^D)$ and $\mathcal{H}(n_i)$, which are upper bounds on the statistical errors of DYNASAGA(ρ) and ERM respectively. If $\mathcal{H}()$ is a tight bound on the statistical error, and hence, a lower bound on the total error, then c-risk-competitiveness to DYNASAGA(ρ) implies that the expected sub-optimality of the algorithm A is within a factor of c of the total risk of DYNASAGA(ρ), as illustrated in Figure 3.1. We next show if a streaming data algorithm is risk-competitive with respect to DYNASAGA(ρ) then it is also risk-competitive with respect to ERM, under certain conditions.

Lemma 2. *If a streaming data algorithm A is c-risk-competitive to* DYNASAGA*(ρ) at time step i, then A is $c \cdot \max\left(\left(\frac{2\widetilde{\lambda}_i}{\rho}\right)^{\alpha}, 1\right)$-risk-competitive to* ERM *at time step i, where $\widetilde{\lambda}_i = \left(\frac{n_i}{i}\right)$ and $1/2 \leq \alpha \leq 1$ is the exponent in the statistical error upper bound $\mathcal{H}(n) = hn^{-\alpha}$.*

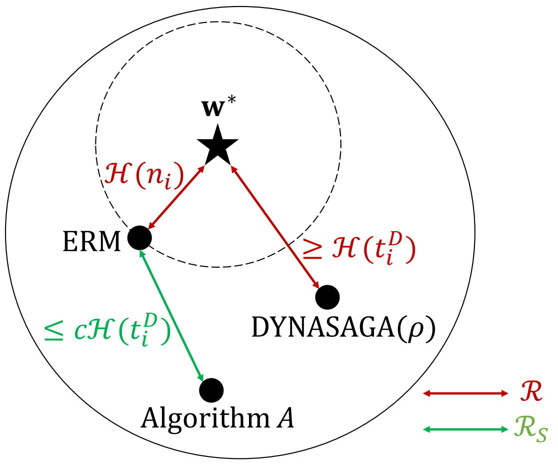


Figure 3.1: Depiction of the errors assuming that an algorithm A is c-risk-competitive to DYNASAGA(ρ). The total error of DYNASAGA(ρ) is at least the statistical error bound $\mathcal{H}(t_i^D)$, and the sub-optimality of A is at most a factor of c of $\mathcal{H}(t_i^D)$.

Proof. From Definition 3, $\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_i}(A)\right] \leq c\mathcal{H}(t_i^D)$. And by Lemma 1, $t_i^D = \min(n_i, \rho i/2)$. In the first case $t_i^D = n_i$, $\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_i}(A)\right] \leq c\mathcal{H}(t_i^D) = c\mathcal{H}(n_i)$. Otherwise, $t_i^D = \rho i/2$, and $\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_i}(A)\right] \leq c\mathcal{H}(t_i^D) = c\left(\frac{2\widetilde{\lambda}_i}{\rho}\right)^{\alpha}\mathcal{H}(n_i)$. □

Discussion: $\widetilde{\lambda}_i = (n_i/i)$ is the average rate of arrivals in a time step. We expect the ratio $(\widetilde{\lambda}_i/\rho)$ to be a small constant. If this ratio is a large number, much greater than 1, the total number of arrivals over i time steps far exceeds the number of gradient computations the algorithm can perform over i time steps. This rate of arrivals is unsustainable, because most practical algorithms such as SGD and variants, including SVRG and SAGA, require more than one gradient computation for each training point. Hence, the above lemma implies that if A is $O(1)$-risk-competitive to DYNASAGA(ρ), then it is also $O(1)$-risk-competitive to ERM, under reasonable arrival patterns.

Finally, Lemma 3 bounds the expected sub-optimality of STRSAGA over its effective sample set T_i. In §3.3, we will show how to apply the following result to establish the risk-competitiveness of STRSAGA.

Lemma 3. *Suppose all $f_{\mathbf{x}}$ are convex and their gradients are L-Lipschitz continuous, and that $\mathcal{R}_{T_i}$ is μ-strongly convex. At the end of each time step i, the expected sub-optimality of* STRSAGA *over T_i is*

$$\mathbb{E}\left[\texttt{SUBOPT}_{T_i}(\textsf{STRSAGA})\right] \leq \mathcal{H}(t_i^{STR}) + 2\left(\mathcal{R}(\mathbf{w}_0) - \mathcal{R}(\mathbf{w}^*)\right)\left(\frac{L}{\mu}\right)^3\left(\frac{1}{t_i^{STR}}\right)^2.$$

If we additionally assume that the condition number L/μ is bounded by a constant at each time, the above simplifies to $\mathbb{E}\left[\texttt{SUBOPT}_{T_i}(\textsf{STRSAGA})\right] \leq (1+o(1))\mathcal{H}(t_i^{STR})$.

3.2.2 Proof of Lemma 3

Throughout, we assume that all $f_{\mathbf{x}}$ are convex and their gradients are L-Lipschitz continuous, and that $\mathcal{R}_{\mathcal{S}}$ is μ-strongly convex for the set of training samples $\mathcal{S}$. In addition, we will use $\mathbf{U}$, as defined by Daneshmand et al. [21] in their analysis of DYNASAGA.

$$\mathbf{U}(t,n) = \min \begin{cases} \rho_n \mathbf{U}(t-1,n) \\ \min\limits_{m<n} [\mathbf{U}(t,m) + \dfrac{n-m}{n}\mathcal{H}(m)], \end{cases} \tag{3.1}$$

where ρ_n is the convergence rate of SAGA and defines as $\rho_n = 1 - \min(\frac{1}{n}, \frac{\mu}{L})$. And, the initial error $\mathbf{U}(0,m) = \zeta$ is defined as:

$$\zeta := \frac{4L}{\mu}[\mathcal{R}(\mathbf{w}_0) - \mathcal{R}(\mathbf{w}^*)].$$

We will use the following results from Daneshmand et al. [21].

Lemma 4. (THEOREM 3 IN [21]) *Suppose the expected sub-optimality of an algorithm A over a training set $\mathcal{T} \subseteq \mathcal{S}$ is bounded as $\mathbb{E}[\texttt{SUBOPT}_{\mathcal{T}}(A)] \le \epsilon$. Then the expected sub-optimality of A over $\mathcal{S}$ is bounded by $\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}}(A)] \le \epsilon + \frac{n-m}{n}\mathcal{H}(m)$, where $|\mathcal{T}| = m, |\mathcal{S}| = n$.*

Lemma 5. (PROPOSITION 4. IN [21]) *The expected sub-optimality of* DYNASAGA *over a training set $\mathcal{S}$ at iteration t is*

$$\mathbb{E}[\texttt{SUBOPT}_{\mathcal{S}}(\mathsf{DYNASAGA})] \le \mathbf{U}(t,n).$$

where the expectation is taken over randomness of $\mathcal{S}$.

Lemma 6. (LEMMA 5 IN [21]) *For $\mathcal{H}(n) = cn^{-\alpha}$,where $1/2 \le \alpha \le 1$,*

$$\mathbf{U}(2n,n) \le \mathcal{H}(n) + \frac{\zeta}{2}\left(\frac{L}{\mu n}\right)^2.$$

The expected sub-optimality of STRSAGA over its effective sample set can be bounded similarly in terms of the function $\mathbf{U}$.

Lemma 7. *At the end of each time step i, the expected sub-optimality of* STRSAGA *over T_i is*

$$\mathbb{E}[\texttt{SUBOPT}_{T_i}(\mathsf{STRSAGA})] \le \mathbf{U}(2t_i^{STR}, t_i^{STR}).$$

Proof. The proof is similar to the proof of Proposition 4 in [21]. Note that performing extra steps of SAGA when the `Buf` is empty does not weaken the bound. □

The proof of Lemma 3 immediately follows by substitution of Lemma 7.

Proof. The expected sub-optimality is bounded by the $\mathbf{U}$ function by Lemma 7, and we have a bound on $\mathbf{U}$ by Lemma 6. Therefore,

$$\begin{aligned}\mathbb{E}[\texttt{SUBOPT}_{T_i}(\mathsf{STRSAGA})] &\le \mathbf{U}(2t_i^{\mathrm{STR}}, t_i^{\mathrm{STR}}) \\ &\le \mathcal{H}(t_i^{\mathrm{STR}}) + \frac{\zeta}{2}\left(\frac{L}{\mu t_i^{\mathrm{STR}}}\right)^2 \\ &= \mathcal{H}(t_i^{\mathrm{STR}}) + 2\left(\mathcal{R}(\mathbf{w}_0) - \mathcal{R}(\mathbf{w}^*)\right)\left(\frac{L}{\mu}\right)^3\left(\frac{1}{t_i^{\mathrm{STR}}}\right)^2.\end{aligned}$$

□

Similarly, we can also prove a bound on the expected sub-optimality of DYNASAGA(ρ).

Lemma 8. *At the end of each time step i, the expected sub-optimality of* DYNASAGA*(ρ) over $\mathcal{S}_i$ is*

$$\mathbb{E}\left[\mathtt{SUBOPT}_{\mathcal{S}_i}(\mathsf{DYNASAGA}(\rho))\right] \leq \left(\max\left(1, \left(\frac{2\widetilde{\lambda}_i}{\rho}\right)^{1+\alpha}\right) + o(1)\right)\mathcal{H}(n_i)$$

where $\widetilde{\lambda}_i = \left(\frac{n_i}{i}\right)$ and $n_i = |\mathcal{S}_i|$.

Proof. According to Lemma 5, the expected sub-optimality of DYNASAGA(ρ) over the sample set $\mathcal{S}_i$ of size n_i after t iterations is bounded by $\mathbf{U}(t, n_i)$. As mentioned earlier, the Algorithm DYNASAGA(ρ) has limited computational power and can performs only ρi steps of SAGA. Thus,

$$\mathbb{E}\left[\mathtt{SUBOPT}_{\mathcal{S}_i}(\mathsf{DYNASAGA}(\rho))\right] \leq \mathbf{U}(\rho i, n_i)$$

If $\widetilde{\lambda}_i \leq \rho/2$, then

$$\begin{aligned}\mathbb{E}\left[\mathtt{SUBOPT}_{\mathcal{S}_i}(\mathsf{DYNASAGA}(\rho))\right] &\leq \mathbf{U}(\rho i, n_i)\\ &\leq \mathbf{U}(2\widetilde{\lambda}_i i, n_i) = \mathbf{U}(2n_i, n_i)\\ &\leq \mathcal{H}(n_i) + \frac{\zeta}{2}\left(\frac{L}{\mu n_i}\right)^2\end{aligned}$$

If $\widetilde{\lambda}_i > \rho/2$, then $n_i = (\widetilde{\lambda}_i)i > (\rho/2)\,i$. Let T be a subset of $\mathcal{S}_i$ such that $|T| = (\rho/2)\,i$, then Lemma 4 results

$$\begin{aligned}\mathbb{E}\left[\mathtt{SUBOPT}_{\mathcal{S}_i}(\mathsf{DYNASAGA}(\rho))\right] &\leq \mathbb{E}\left[\mathtt{SUBOPT}_{T_i}(\mathsf{DYNASAGA}(\rho))\right] + \frac{\widetilde{\lambda}_i i - (\rho/2)\,i}{(\rho/2)\,i}\mathcal{H}\left((\rho/2)i\right)\\ &\leq \mathbf{U}(\rho i, (\rho/2)\,i) + \frac{\widetilde{\lambda}_i i - (\rho/2)\,i}{(\rho/2)\,i}\mathcal{H}((\rho/2)\,i)\\ &\leq \mathcal{H}((\rho/2)\,i) + \frac{\zeta}{2}\left(\frac{L}{\mu(\rho/2)i}\right)^2 + \left(\frac{2\widetilde{\lambda}_i}{\rho} - 1\right)\mathcal{H}((\rho/2)\,i)\\ &= \left(\frac{2\widetilde{\lambda}_i}{\rho}\right)\mathcal{H}\left(\frac{\rho}{2\widetilde{\lambda}_i}n_i\right) + \frac{\zeta}{2}\left(\frac{L}{\mu(\rho/2)i}\right)^2\\ &= \left(\frac{2\widetilde{\lambda}_i}{\rho}\right)^{1+\alpha}\mathcal{H}(n_i) + \frac{\zeta}{2}\left(\frac{2\widetilde{\lambda}_i}{\rho}\right)^2\left(\frac{L}{\mu n_i}\right)^2.\end{aligned}$$

□

3.3 Competitive Analysis of STRSAGA on Specific Arrival Distributions

Lemma 3 shows that the expected sub-optimality of STRSAGA over its effective sample set T_i is $O(\mathcal{H}(t_i^{\mathrm{STR}}))$ (note t_i^{STR} is not equal to n_i the number of points so far). However, our goal is to show that STRSAGA is risk-competitive to DYNASAGA(ρ) at each time step i; i.e., the expected sub-optimality of STRSAGA over $\mathcal{S}_i$ is within a factor of $\mathcal{H}(t_i^D)$. The connection between the two depends on the relation between t_i^{STR} and t_i^D. This relation is captured using sample-competitiveness, which is introduced in this section. Although not every arrival distribution provides sample-competitiveness, we will show a number of different patterns of arrival distributions that do provide this property. To model different arrival patterns, we consider a general arrival model where the number of points arriving in time step i is a random variable x_i which is independently drawn from distribution $\mathcal{P}$ with a finite mean λ. We consider arrival distributions of varying degrees of generality, including Poisson arrivals, skewed arrivals, general arrivals with a bounded maximum, and general arrivals with an unbounded maximum. The proofs of some results about specific distributions, as well as the full statements of prior theorems and bounds referenced below, is deferred to §3.3.1.

Throughout, we will assume that all $f_{\mathbf{x}}$ are convex and their gradients are L-Lipschitz continuous, that the empirical risk objective $\mathcal{R}_{\mathcal{S}}$ is μ-strongly convex, and that the condition number L/μ is bounded by a constant at each time.

Definition 4. *At time i, STRSAGA is said to be k-sample-competitive to DYNASAGA(ρ) if $t_i^{STR}/t_i^D \geq k$.*

Lemma 9. *If STRSAGA is k-sample-competitive to DYNASAGA(ρ) at time step i, then it is c-risk-competitive to DYNASAGA(ρ) at time step i with $c = k^{-\alpha}(2 + o(1))$.*

Proof. Let ${T_i}^{\text{STR}}$ and ${T_i}^D$ denote the effective samples that were used at iteration i for STRSAGA and DYNASAGA(ρ), respectively. We know that ${T_i}^{\text{STR}}, {T_i}^D \subseteq \mathcal{S}_i$. Using Lemma 4 in §3.2.2, we have: $\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_i}(\text{STRSAGA})\right] \leq \mathbb{E}\left[\texttt{SUBOPT}_{T_i}(\text{STRSAGA})\right] + \frac{n_i - {t_i}^{\text{STR}}}{n_i}\mathcal{H}({t_i}^{\text{STR}})$.

Using Lemma 3, we can rewrite the above inequality as

$$\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_i}(\text{STRSAGA})\right] \leq (1 + o(1))\mathcal{H}({t_i}^{\text{STR}}) + \frac{n_i - {t_i}^{\text{STR}}}{n_i}\mathcal{H}({t_i}^{\text{STR}}) \leq (2 + o(1))\mathcal{H}({t_i}^{\text{STR}}).$$

If STRSAGA is k-sample-competitive to DYNASAGA(ρ), then the result follows:

$$\mathbb{E}\left[\texttt{SUBOPT}_{\mathcal{S}_i}(\text{STRSAGA})\right] \leq (2+o(1))\mathcal{H}({t_i}^{\text{STR}}) \leq (2+o(1))\mathcal{H}(k \cdot {t_i}^D) = k^{-\alpha}(2+o(1))\mathcal{H}({t_i}^D).$$

□

Constant Arrival Rate. We first consider the case where $x_i = \lambda$ for each i, so that the number of arrivals in each time step is the same.

Lemma 10. *For a constant arrival rate, STRSAGA is $(2+o(1))$-risk-competitive to DYNASAGA(ρ) at any time step.*

Proof. The result follows from Lemma 9 because STRSAGA is 1-sample-competitive to DYNASAGA(ρ) at every time step. Recall STRSAGA moves one point from `Buf` (if available) every two iterations of SAGA. In the processing-limited regime $\rho/2 \leq \lambda$, $t_i^{\text{STR}} = \rho i/2 = t_i^D$, and in the sample-limited regime $\lambda < \rho/2$, $t_i^{\text{STR}} = \lambda i = t_i^D$. □

For non-trivial arrival distributions, our proof strategy is to first establish sample-competitiveness through the following lemma, and then apply Lemma 9 to show risk-competitiveness.

Lemma 11. *At time step i, suppose the streaming arrivals satisfy: $n_{i/2} \geq kn_i$. Then,* STRSAGA *is* $\min\{k, \frac{1}{2}\}$*-sample-competitive to* DYNASAGA*(ρ) at time step i.*

Proof. We first bound t_i^{STR}. At time $i/2$, at least kn_i points have arrived. In Algorithm 1, at time $i/2$, these points are either in the `Buf` or already in the effective sample $T_{i/2}^{\text{STR}}$. For every two iterations of SAGA, the algorithm moves one point from `Buf` (if available) to the effective sample, thus increasing the size of the effective sample set by 1. In the $i/2$ time steps from $i/2+1, \ldots, i$, STRSAGA can perform $\rho i/2$ iterations of SAGA. Within these iterations, it can move $\rho i/4$ points to T_i^{STR}, if available in the buffer. Hence, the effective sample size for STRSAGA at time i is: $t_i^{\text{STR}} \geq \min\{\rho i/4, kn_i\}$. We know $t_i^D = \min\{n_i, \rho i/2\}$.

We consider four cases. In the first case, (1) if $\rho i/4 < n_{i/2}$ and $n_i < \rho i/2$, then $t_i^D = n_i$ and $t_i^{\text{STR}} \geq \rho i/4$. In this case, we have $t_i^{\text{STR}} \geq \rho i/4 > n_i/2 = t_i^D/2$. The other three cases, (2) $\rho i/4 < n_{i/2}$ and $n_i \geq \rho i/2$, (3) $\rho i/4 \geq n_{i/2}$ and $n_i < \rho i/2$, and (4) $\rho i/4 \geq n_{i/2}$ and $n_i \geq \rho i/2$, can be handled similarly. □

Skewed Arrivals with a Bounded Maximum. We next consider an arrival distribution parameterized by integer $M \geq \lambda$, where the number of arrivals per time step can either be high (M) or zero. More precisely, $x_i = M$ with prob. $\frac{\lambda}{M}$ and $x_i = 0$ with prob. $1 - \frac{\lambda}{M}$. Thus, $E[x_i] = \lambda$. For $M > \lambda$, this models bursty arrival distributions with a number of "quiet" time steps with no arrivals, combined with an occasional burst of M arrivals. We have the following result for skewed arrivals.

Lemma 12. *For a skewed arrival distribution with maximum M and mean λ,* STRSAGA *is* $6^\alpha(2+o(1))$*-risk-competitive to* DYNASAGA*(ρ), with probability at least $1-\epsilon$, at any time step $i > \frac{16M}{\lambda}\ln\frac{1}{\epsilon}$.*

Note that as M increases, arrivals become more bursty, and it takes longer for the algorithm to be competitive, with a high confidence.

General Arrivals with a Bounded Maximum. We next consider a more general arrival distribution with a maximum of M arrivals, and a mean of λ. $x_i = j$ with probability p_j for $j = 0, \ldots, M$, such that $\sum_j p_j = 1$ and $E[x_i] = \lambda$, for an integer $M > 0$.

Lemma 13. *For a general arrival distribution with mean λ and maximum M, at any time step $i > (\frac{16M}{\lambda} + \frac{8}{3})\ln\frac{1}{\epsilon}$,* STRSAGA *is* $8^\alpha(2+o(1))$*-risk-competitive to* DYNASAGA*(ρ), with probability at least $1-\epsilon$.*

General Arrivals with an Unbounded Maximum. More generally, the number of arrivals in a time step may not have a specified maximum. The arrival distribution can have a finite mean, despite a small probability of reaching arbitrarily large values. We consider a sub-class of such distributions where all the polynomial moments are bounded, as in the following *Bernstein's condition* with parameter b: The random variable x_i has mean λ, variance σ^2, and $|\mathbb{E}\left[(x_i - \lambda)^k\right]| \leq \frac{1}{2}k!\sigma^2 b^{k-2}$ for all integers $k \geq 3$ [65].

Lemma 14. *For any arrival distribution with mean λ, bounded variance σ^2 and satisfying Bernstein's condition with parameter b,* STRSAGA *is* $8^\alpha(2+o(1))$*-risk-competitive to* DYNASAGA*(ρ), with probability at least $1-\epsilon$, at any time step $i > \max((16(\frac{\sigma}{\lambda})^2 + \frac{8}{3})\ln\frac{1}{\epsilon}, 2((\frac{\sigma}{\lambda})^2 + \frac{b}{\lambda})\ln\frac{1}{\epsilon})$.*

Poisson Arrivals. We next consider the case where the number of points arriving in each time step follows a Poisson distribution with mean λ, i.e., $\mathbb{P}(x_i = k) = \frac{e^{-\lambda}\lambda^k}{k!}$ for integer $k \geq 0$.

Lemma 15. *For Poisson arrival distribution with mean λ,* STRSAGA *is $8^\alpha(2 + o(1))$-risk-competitive to* DYNASAGA*(ρ) with probability at least $1 - \epsilon$, at any time step $i > \frac{16}{\lambda} \ln \frac{1}{\epsilon}$.*

3.3.1 Additional Proof Details

Skewed Arrivals with a Bounded Maximum. We prove concentration of n_i and $n_{i/2}$ using a Chernoff bound. We use the following form of the Chernoff bound from Mitzenmacher and Upfal [67] tailored to the Poisson distribution.

Theorem 1. *(Chernoff Bound) Let $X_1, ..., X_n$ be independent Poisson trials such that $\mathbb{P}(X_i) = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then the following bounds hold:*

- *For* $0 < \delta < 1$,
$$\mathbb{P}(X \leq (1-\delta)\mu) \leq e^{-\mu\delta^2/2}$$
- *For* $0 < \delta \leq 1$,
$$\mathbb{P}(X \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3}$$

Lemma 16. *For a skewed arrival distribution with mean λ and parameterized by M, for $i > \frac{3M}{\delta^2\lambda} \ln \frac{1}{\epsilon}$, with probability at least $1 - \epsilon$, we have $n_i \leq (1+\delta)\lambda i$, where $0 < \delta \leq 1$.*

Proof. Let Y_i denotes the number of non-empty arrivals in time steps $1, \dots, i$. Y_i follows the binomial distribution with parameters $n = i$, and $p = \lambda/M$, i.e., $Y_i \sim B(i, \lambda/M)$ and $\mathbb{E}[Y_i] = \lambda i/M$. By Chernoff (Theorem 1), for $0 < \delta \leq 1$:

$$\mathbb{P}(Y_i \geq (1+\delta)\lambda i/M) \leq e^{-\frac{\delta^2\lambda i}{3M}} \leq \epsilon$$

On the other hand, we have n_i, the number of arrivals in the first i time steps, is $M \cdot Y_i$. Thus, for $i > \frac{3M}{\delta^2\lambda} \ln \frac{1}{\epsilon}$ with probability at least $1 - \epsilon$, we have $n_i \leq (1+\delta)\lambda i$. □

Lemma 17. *For a skewed arrival distribution with mean λ and parameterized by M, for $i > \frac{4M}{\delta^2\lambda} \ln \frac{1}{\epsilon}$, with probability at least $1 - \epsilon$, we have $n_{i/2} \geq (1-\delta)\lambda i/2$, where $0 < \delta < 1$.*

Proof. Same as Lemma 16, let Y_i denotes the number of non-empty arrivals in time steps $1, \dots, i$. Y_i follows the binomial distribution with parameters $n = i$, and $p = \lambda/M$, i.e., $Y_i \sim B(i, \lambda/M)$ and $\mathbb{E}[Y_i] = \lambda i/M$. By the Chernoff bound (Theorem 1), for $0 < \delta < 1$:

$$\mathbb{P}\left(Y_{i/2} \leq (1-\delta)\frac{\lambda i}{2M}\right) \leq e^{-\frac{\delta^2\lambda i}{4M}} \leq \epsilon$$

On the other hand, we have $n_{i/2}$, total number of arrivals in the first $i/2$ time steps, is $M \cdot Y_{i/2}$. Thus, for $i > \frac{4M}{\delta^2\lambda} \ln \frac{1}{\epsilon}$ with probability at least $1 - \epsilon$, we have $n_{i/2} \geq (1-\delta)\lambda i/2$. □

Now we can prove Lemma 12, repeated below.

Lemma 12. *For a skewed arrival distribution with maximum M and mean λ,* STRSAGA *is $6^\alpha(2 + o(1))$-risk-competitive to* DYNASAGA*(ρ), with probability at least $1 - \epsilon$, at any time step $i > \frac{16M}{\lambda} \ln \frac{1}{\epsilon}$.*

Proof. By setting $\delta = 1/2$ in Lemma 16, for $i > \frac{12M}{\lambda} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_i \leq \left(\frac{3}{2}\right) \lambda i$. On the other hand, by setting $\delta = 1/2$ in Lemma 17, for $i > \frac{16M}{\lambda} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_{i/2} \geq \lambda i/4$. Therefore, using union bound we can conclude with probability at least $1-2\epsilon$, we have $n_{i/2} \geq \frac{1}{6} n_i$ for $i > \frac{16M}{\lambda} \ln \frac{1}{\epsilon}$. As a result, using Lemma 11, STRSAGA and DYNASAGA(ρ) are at least $\frac{1}{6}$-sample-competitive and therefore by Lemma 9, STRSAGA is $6^{\alpha}(2 + o(1))$-risk-competitive with DYNASAGA(ρ). □

General Arrivals with a Bounded Maximum. In order to prove concentration bounds for n_i and $n_{i/2}$, the plan is to use Bernstein's inequality [65], which lets us bound the sum of independent random variables in a more flexible manner than Chernoff bounds (for random variables that are not necessarily binary valued), in conjunction with a bound on the variance of the distribution.

Observation 1. *Let $x_1, x_2, \ldots, x_n$ be independent random variables such that $\mathbb{E}[x_i] = \lambda$ and the range of these random variables is $\{0, 1, , \ldots, M\}$, then the variance of x_i is no more than $\lambda(M - \lambda)$.*

Proof.

$$
\begin{aligned}
Var[x_i] &= \sum_{j=0}^{M} p_j (j-\lambda)^2 = \left(\sum_{j=0}^{M} j^2 p_j \right) - \lambda^2 \\
&\leq M \sum_{j=0}^{M} j p_j - \lambda^2 = M\lambda - \lambda^2
\end{aligned}
$$

□

We use Bernstein's inequality from Massart [65].

Theorem 2. *(Bernstein's Inequality) Let $x_1, x_2, \ldots, x_n$ be independent bounded random variables such that $\mathbb{E}[x_i] = 0$ and $x_i \leq M$ with probability 1 and let $\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} Var[x_i]$. Then for any $a \geq 0$ we have:*

$$
\mathbb{P}\left(\frac{1}{n} \sum_{j=1}^{n} x_i \geq a \right) \leq e^{-\frac{na^2}{2\sigma^2 + 2Ma/3}}
$$

By Bernstein's inequality, we can show the following.

Lemma 18. *For any general arrival distribution with mean λ and bounded maximum M, for $i > \frac{2(k+2)}{3(k-1)^2} \frac{M}{\lambda} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_i \leq k\lambda i$, for any $k > 1$.*

Proof. According to Observation 1, we have $Var[x_i] \leq M\lambda$. Let's define random variable $z_i = x_i - \lambda$. We have $\mathbb{E}[z_i] = 0$ and $Var[z_i] = Var[x_i] \leq M\lambda$. By Bernstein's inequality (Theorem 2), and setting a to λ we have:

$$\mathbb{P}\left(n_i \geq k\lambda i\right) = \mathbb{P}\left(\frac{1}{i}(n_i - \lambda i) \geq (k-1)\lambda\right) = \mathbb{P}\left(\frac{1}{i}\sum_{j=1}^{i}(x_j - \lambda) \geq (k-1)\lambda\right)$$
$$= \mathbb{P}\left(\frac{1}{i}\sum_{j=1}^{i} z_j \geq (k-1)\lambda\right) \leq e^{-\frac{i(k-1)^2\lambda^2}{2M\lambda + 2M(k-1)\lambda/3}} = e^{-\frac{3(k-1)^2}{2(k+2)}\frac{\lambda}{M}i}$$

For $i > \frac{2(k+2)}{3(k-1)^2}\frac{M}{\lambda}\ln\frac{1}{\epsilon}$, this probability is at most ϵ. □

We also make use of the following theorem from Bercu et al. [4].

Theorem 3. *Let $x_1, x_2, \ldots, x_n$ be a finite sequence of independent and non-negative random variables with finite variances. Denote $S_n = x_1 + x_2 + \ldots + x_n$ and $V_n = Var(S_n)$. Then, for any $a \geq 0$,*

$$\mathbb{P}\left(S_n \leq \mathbb{E}\left[S_n\right] - a\right) \leq e^{-\frac{a^2}{2V_n + W_n}}$$

where

$$W_n = \frac{1}{3}\sum_{k=1}^{n}\left(\frac{{m_k}^2 - v_k}{m_k}\right)^2, \quad m_k = \mathbb{E}\left[x_k\right] \quad \text{and} \quad v_k = Var(x_k)$$

Given the above bound, we can bound $n_{i/2}$ for the bounded maximum case.

Lemma 19. *For any general arrival distribution with mean λ and bounded maximum M, for $i > \frac{12M/\lambda + 2}{3(1-2k)^2}\ln\frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_{i/2} \geq k\lambda i$, for any $k < \frac{1}{2}$.*

Proof. By Theorem 3,

$$\mathbb{P}\left(n_{i/2} \leq k\lambda i\right) = \mathbb{P}\left(n_{i/2} \leq \left(\lambda\frac{i}{2} - \lambda i(\frac{1}{2} - k)\right)\right) \leq e^{-\frac{a^2}{2V_n + W_n}} \leq e^{-\frac{\lambda^2 i^2 (1/2-k)^2}{\lambda M i + \lambda^2 i/6}} = e^{-\frac{3(1-2k)^2}{12M/\lambda + 2}i}$$

Thus, for $i > \frac{12M/\lambda + 2}{3(1-2k)^2}\ln\frac{1}{\epsilon}$ with probability at least $1-\epsilon$, we have $n_{i/2} \geq k\lambda i$. □

Now we can prove Lemma 13, repeated below.

Lemma 13. *For a general arrival distribution with mean λ and maximum M, at any time step $i > (\frac{16M}{\lambda} + \frac{8}{3})\ln\frac{1}{\epsilon}$,* STRSAGA *is $8^\alpha(2+o(1))$-risk-competitive to* DYNASAGA*(ρ), with probability at least $1-\epsilon$.*

Proof. Similar to the proof of Lemma 12, by setting $k = 2$ in Lemma 18 and $k = 1/4$ in Lemma 19. □

General Arrivals with an Unbounded Maximum. We will use the following theorem from Massart [65].

Theorem 4. *Let $x_1, x_2, \ldots, x_n$ be independent random variables with mean λ and variance σ^2 satisfying the Bernstein condition with parameter b, $|\mathbb{E}\left[(x_i - \lambda)^k\right]| \leq \frac{1}{2}k!\sigma^2 b^{k-2}$ for all integers $k \geq 3$. Then for any $t \geq 0$ we have:*

$$\mathbb{P}\left(\sum_{i}^{n}(x_i - \lambda) \geq t\right) \leq e^{-\frac{t^2}{2(\sigma^2 + bt)}}$$

A bound n_i follows from the above theorem.

Lemma 20. *For any arrival distribution x_i with $\mathbb{E}[x_i] = \lambda$ and variance σ^2 that satisfies Bernstein's condition with parameter b, for $i > \frac{2(\sigma^2+b\lambda)}{(k-1)^2\lambda^2} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_i \leq k\lambda i$, for any $k > 1$.*

Proof. According to Theorem 4, we have:

$$Pr[n_i \geq k\lambda i] = Pr[(n_i - \lambda i) \geq (k-1)\lambda i] \leq e^{-\frac{(k-1)^2\lambda^2}{2(\sigma^2+b\lambda)}i}$$

when $i \geq \frac{2(\sigma^2+b\lambda)}{(k-1)^2\lambda^2} \ln \frac{1}{\epsilon}$, this probability is at most ϵ. □

Lemma 21. *For any arrival distribution x_i with mean λ and variance σ^2, for $i > \frac{12(\sigma/\lambda)^2+2}{3(1-2k)^2} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_{i/2} \geq k\lambda i$ for any $k < \frac{1}{2}$.*

Proof. Similar to the proof of Lemma 19. □

Now we can prove Lemma 14, repeated below.

Lemma 14. *For any arrival distribution with mean λ, bounded variance σ^2 and satisfying Bernstein's condition with parameter b,* STRSAGA *is $8^\alpha(2+o(1))$-risk-competitive to* DYNASAGA*(ρ), with probability at least $1-\epsilon$, at any time step $i > \max((16(\frac{\sigma}{\lambda})^2 + \frac{8}{3}) \ln \frac{1}{\epsilon}, 2((\frac{\sigma}{\lambda})^2 + \frac{b}{\lambda}) \ln \frac{1}{\epsilon})$.*

Proof. Similar to the proof of Lemma 12, by setting $k = 2$ in Lemma 20 and $k = 1/4$ in Lemma 21. □

Poisson Arrivals. The following two lemmas follow immediately from the Chernoff bound in Theorem 1.

Lemma 22. *For $i > \frac{3}{\lambda\delta^2} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_i \leq (1+\delta)\lambda i$ for any $0 < \delta \leq 1$.*

Lemma 23. *For $i > \frac{4}{\lambda\delta^2} \ln \frac{1}{\epsilon}$, with probability at least $1-\epsilon$, we have $n_{i/2} \geq (1-\delta)\lambda i/2$ for any $0 < \delta < 1$.*

Now we can prove Lemma 15, repeated below.

Lemma 15. *For Poisson arrival distribution with mean λ,* STRSAGA *is $8^\alpha(2+o(1))$-risk-competitive to* DYNASAGA*(ρ) with probability at least $1-\epsilon$, at any time step $i > \frac{16}{\lambda} \ln \frac{1}{\epsilon}$.*

Proof. Similar to the proof of Lemma 12, by setting $\delta = 1$ in Lemma 22 and $\delta = 1/2$ in Lemma 23. □

3.4 Experimental Setup

We conduct a set of experiments on real world datasets streamed in under various arrival distributions. We consider two optimization problems that arise in supervised learning, logistic regression (convex) and matrix factorization (nonconvex). For logistic regression, we use the A9A [24] and RCV1.binary [58] datasets, and for matrix factorization, we use two datasets of user-item ratings from Movielens [36]. These static training data are converted into streams, by ordering them by a random permutation, and defining an arrival rate λ dependent on the dataset size. Training data arrive over the course of 100 time steps. We consider skewed arrivals parameterized by $M = 8\lambda$ and Poisson arrivals.

At each time step i, a streaming data algorithm has access to ρ gradient computations to update the model. We examine the case for $\rho/\lambda = 1$ and $\rho/\lambda = 5$. We compare STRSAGA against the offline algorithm DYNASAGA(ρ), which is run from scratch at each time i using ρi steps on S_i. We also compare against two streaming data algorithms, SGD, and for the case $\rho/\lambda = 1$, the single-pass algorithm SSVRG.[1] For the streaming data version of SGD under an expanding set of points $\mathcal{S}_i$, at each time step i we first visit points from $\mathcal{S}_i$ that have not been seen yet, and spend any remaining processing time to sample uniformly from all of $\mathcal{S}_i$. (Across all datasets, this strategy was either better or indistinguishable from just directly sampling from all of $\mathcal{S}_i$.) For our implementation of SSVRG, we have relaxed the memory limitation of the original streaming algorithm by introducing a buffer to store points that have arrived but not yet been processed. With this additional storage, we allow SSVRG to make progress during time steps even when no new points arrive, and hence make for a fairer comparison when data points do not arrive at a steady rate.

3.4.1 Datasets and Parameters

Details of the 4 real-world datasets we used are given in Tables 3.1 and 3.2. We reserve 10% of each dataset for testing and use the remaining 90% for training.

Table 3.1: Datasets for logistic regression

Dataset	Size	Number of Features
RCV1.BINARY	20242	47236
A9A	32561	123

Table 3.2: Datasets for matrix factorization

Dataset	Users	Movies	Date Range	Rating Scale	Density
MovieLens100K	943	1682	9/1997-4/1998	1-5, stars	6.30%
MovieLens1M	6040	3706	4/2000-2/2003	1-5, stars	4.47%

The loss function for the binary classification task is L2-regularized logistic loss. For a data point (x, y), the corresponding loss is $f_{(x,y)}(\mathbf{w}) = \log(1 + \exp(-y\mathbf{w}^T x)) + \frac{\mu}{2}||\mathbf{w}||_2^2$. For collaborative filtering, we solve the matrix factorization problem of finding two rank-10 matrices, $\mathbf{w} = (L, R)$, so that LR^T approximates the known elements of the data matrix M. The regularized loss function for the data point M_{ij} is $f_{(i,j)}(\mathbf{w}) = ((LR^T)_{ij} - M_{ij})^2 + \frac{\mu}{2}(||L||_F^2 + ||R||_F^2)$. The

[1]We consider SSVRG a $\rho/\lambda = 1$ algorithm, because for most data points it receives, it uses 1 gradient computation, and only for an $o(1)$ fraction of the data points does it require 2 gradient computations.

rank 10 for matrix factorization was chosen for good validation set error after optimizing with SGD after a single pass over the static dataset.

We used grid search to determine the values of the regularization factor μ and step size η that optimized the validation set error after passing once fully over the static training set. 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 setting of μ for each dataset is $\mu_{\text{A9A}} = 10^{-3}$, $\mu_{\text{RCV}} = 10^{-5}$, $\mu_{\text{MovieLens}} = 10^{-1}$. For SGD, we used a constant step size, which performed better than a decaying step size of the form $\eta_t = \eta_0/(1 + \eta_0 \mu t)$.

3.5 Experimental Results

We empirically confirm the competitiveness of STRSAGA with the offline algorithm DYNASAGA(ρ) under skewed arrivals and Poisson arrivals, and also find that STRSAGA outperforms SGD and SSVRG.

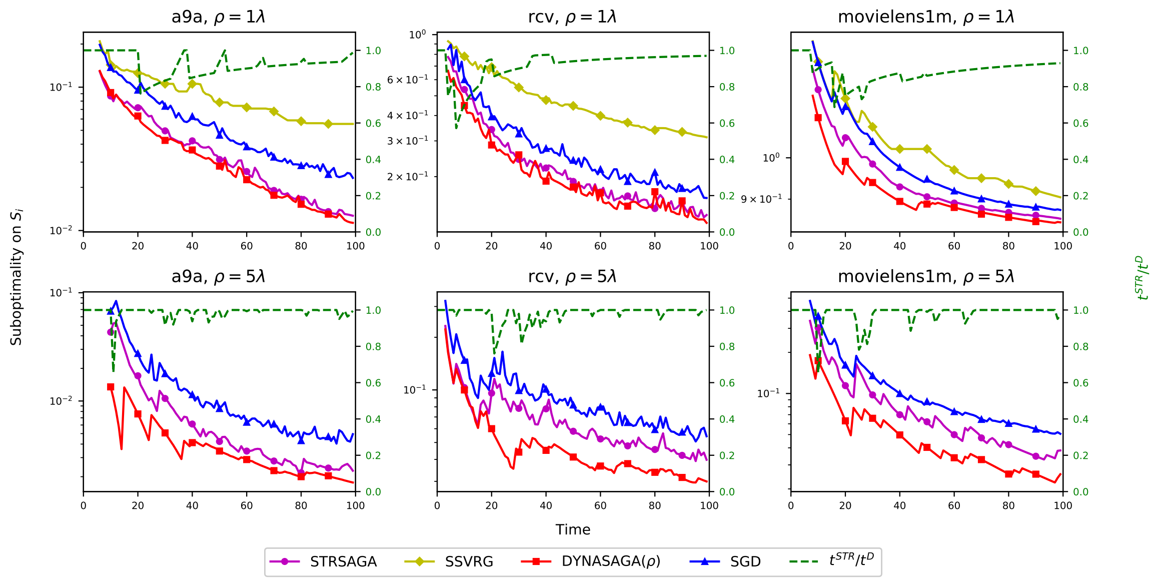


Figure 3.2: Sub-optimality under skewed arrivals with $M = 8\lambda$. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs.

Figure 3.2 shows the sub-optimality of each algorithm and the sample-competitive ratio for STRSAGA under skewed arrivals. The dips in the sample-competitive ratio represent the arrival of a large group of points, and correspondingly at those times, the sub-optimality spikes, because there are now many new points added to S_i that have yet to be processed. We observe that the sample-competitive ratio improves over the lifetime of the stream and tends towards 1, outperforming our pessimistic theoretical analysis. Furthermore, as the sample-competitive ratio increases, the risk-competitiveness of STRSAGA improves so that the sub-optimality of STRSAGA is comparable to that of the offline DYNASAGA(ρ), which is the best we can do given limited computational power. In Figure 3.2, we also observe that STRSAGA outperforms both our streaming data version of SGD, due to the faster convergence rate when using SAGA steps with reduced variance, and also SSVRG, showing the benefit of revisiting data points, even when the processing rate is constrained at $\rho = 1\lambda$.

To better understand the impact of the skewed arrival distribution on the performance of STRSAGA, we did three experiments in Figure 3.3, showing the following results. (1) As M/λ increases, the arrivals become more bursty and it takes longer for STRSAGA to be sample-competitive, and as a result, risk-competitive to DYNASAGA(ρ). Note that the far left endpoint, for skewed arrival parameterized with $M = \lambda$, is the case of constant arrivals. (2) We observe that there is an intermediate point for ρ/λ where it is more difficult to be sample-competitive, but at the extremes the ratio tends towards 1. This is because for large ρ/λ, whenever a big group of points arrives they can all be processed quickly. On the other hand, for small ρ/λ, at any time i, both STRSAGA and the offline algorithm are still processing points that arrived at some time significantly before i, and so a large variance in the amount of fresh arrivals at the tail of the stream can be tolerated. (3) The bound on sub-optimality we showed earlier is dependent on the number of data points processed so far. As we see, as time passes and STRSAGA sees more data points, its sub-optimality on $\mathcal{S}_i$ improves. Additionally as ρ/λ increases, STRSAGA has more steps available to incorporate newly arrived data points and becomes more resilient to bursty arrivals.

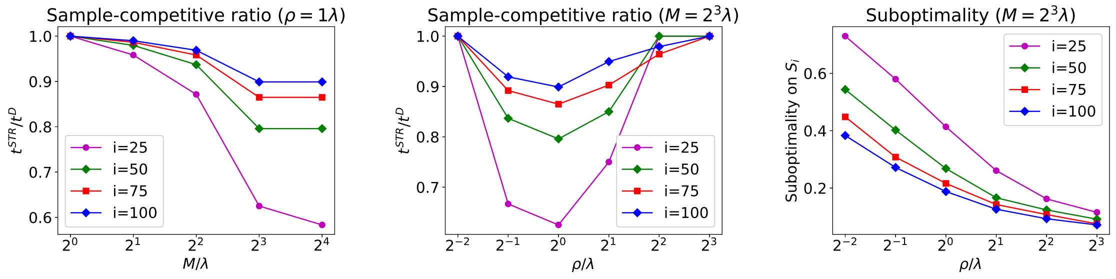


Figure 3.3: Sensitivity analysis. The first plot varies the skew M/λ for a fixed processing rate ρ/λ, and the second two plots vary the processing rate for a fixed skew. Results are plotted for time steps $i = 25, 50, 75, 100$ over a stream of the RCV dataset of 100 time steps. The median is taken over 9 runs.

We also show results for sub-optimality for Poisson arrivals in Figure 3.4 and test loss in Figure 3.6. The median sample-competitive ratio is 1 from the beginning of the stream, which is significantly better than the ratio we showed analytically. Note that the curves for STRSAGA and DYNASAGA(ρ) coincide for $\rho/\lambda = 1$ when STRSAGA is sample-competitive at all points in the stream, since the two algorithms are identical in this regime. Again we find that STRSAGA outperforms SGD and SSVRG.

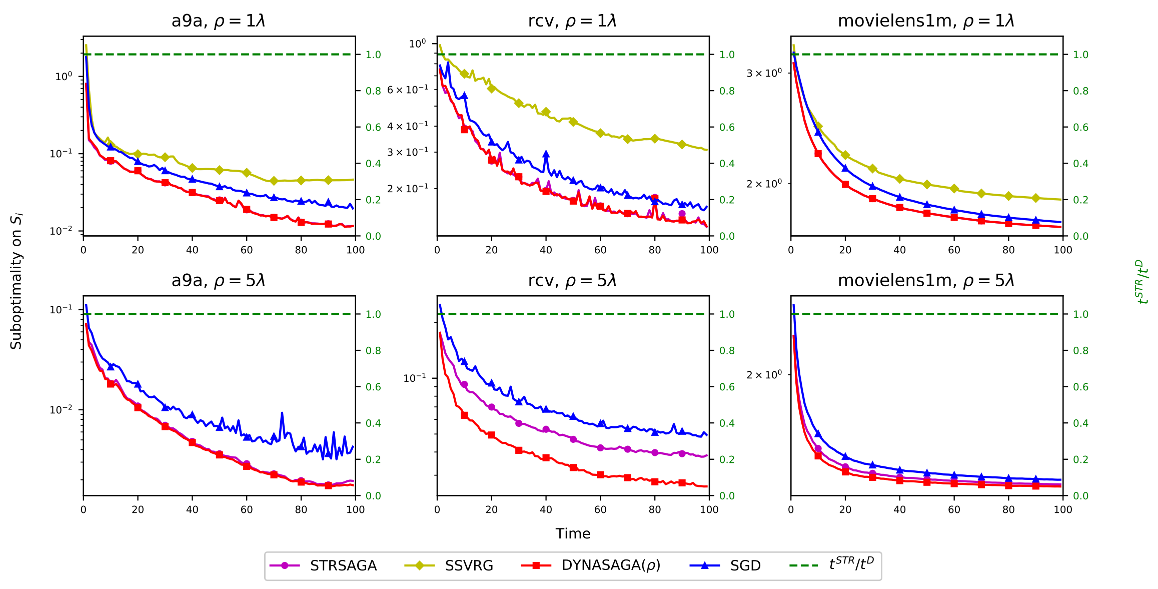


Figure 3.4: Sub-optimality under Poisson arrivals with mean λ. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs.

Additional Results Above, we showed only the sub-optimality. Figure 3.5 shows the test loss under skewed arrivals, and Figure 3.6 shows the test loss under Poisson arrivals. We observe the accuracy of STRSAGA is comparable with the offline algorithm and yields a more accurate model than either SGD or SSVRG.

Finally, we include plots for the MovieLens100K dataset omitted above, for which results are generally similar to those for the larger MovieLens1M dataset presented previously. Sub-optimality is shown in Figure 3.7 and test loss in Figure 3.8.

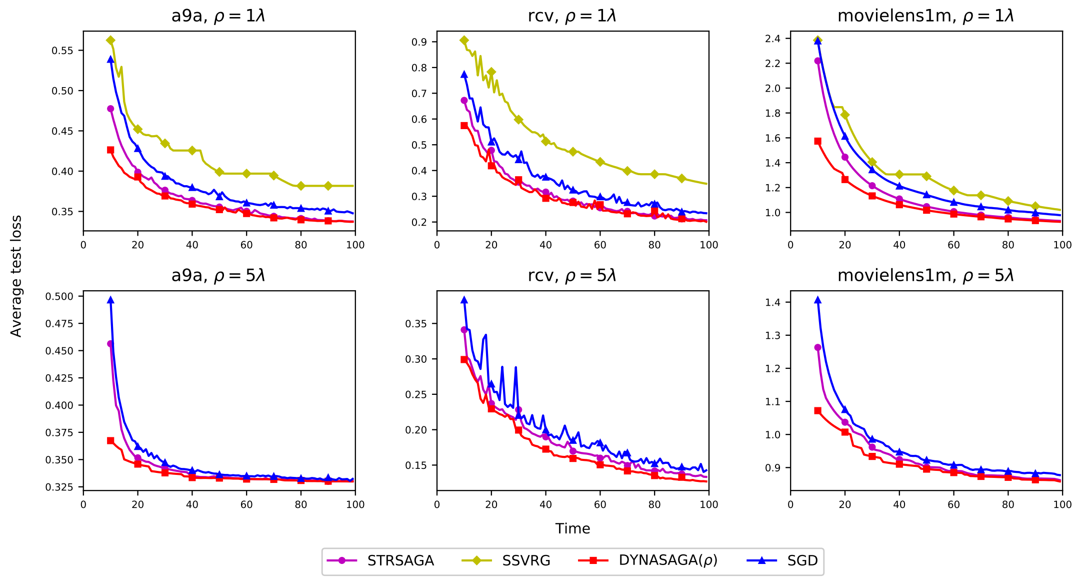


Figure 3.5: Test loss under skewed arrivals with $M = 8\lambda$. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs.

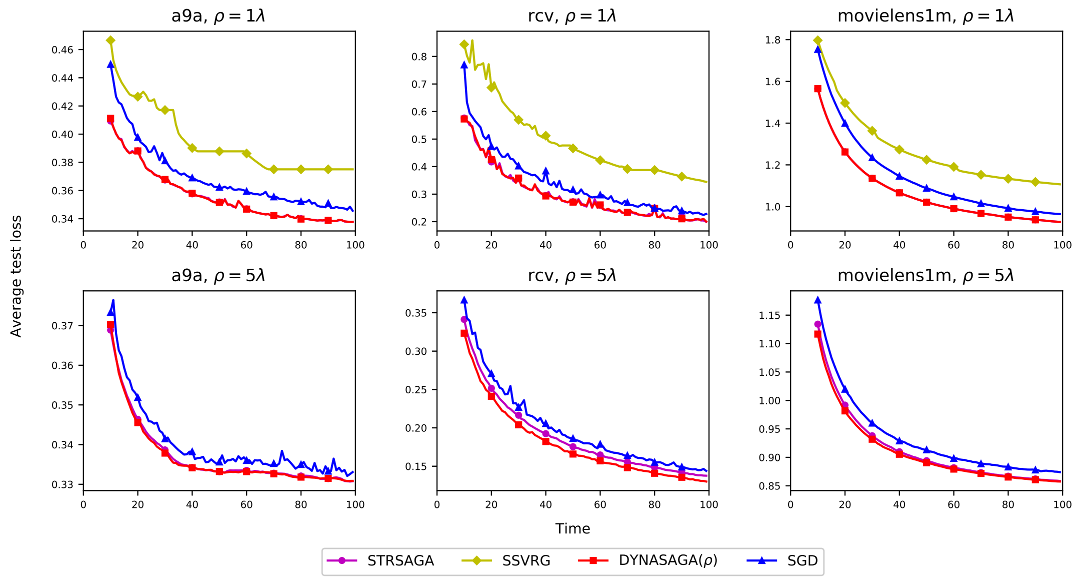


Figure 3.6: Test loss under Poisson arrivals with mean λ. Top row is processing rate $\rho = 1\lambda$, and bottom row is $\rho = 5\lambda$. The median is taken over 5 runs.

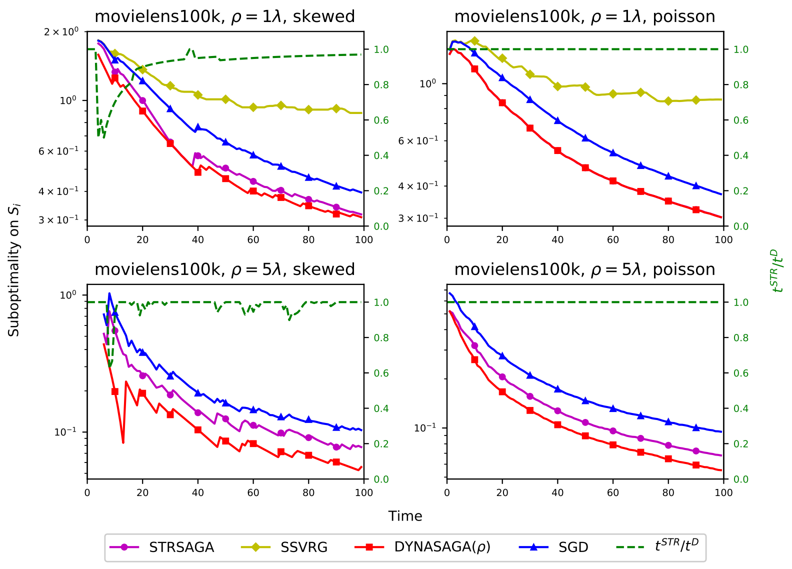


Figure 3.7: Additional plots of sub-optimality for MovieLens100k.

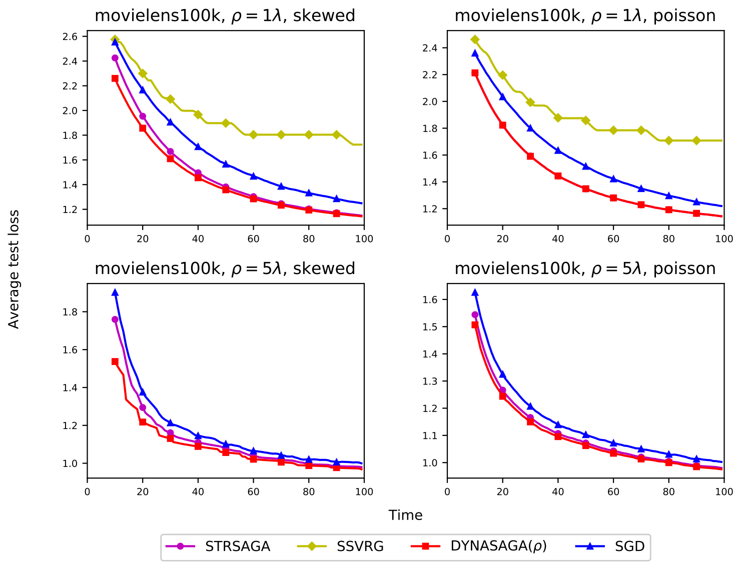


Figure 3.8: Additional plots of test loss for MovieLens100k.

Chapter 4

Learning from non-IID data over time with DriftSurf

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.

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 [13]. Our results on 10 synthetic and real-world datasets with concept drifts show that DriftSurf generally outperforms both MDDM and AUE.

4.1 Related Work

Most adaptive learning algorithms can be classified into three major categories: Weighted sampling, ensembles, and drift detection. Weighted sampling can take the form of a step function, such as the family of FLORA algorithms [88] that 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 [56] or exponential [40, 51] decay. Weighted sampling algorithms are guaranteed to adapt to drifts, but at a cost in accuracy in the absence of drift.

Ensemble methods, such as DWM [53], Learn++.NSE [26], AUE [13], DWMIL [61], DTEL [83], Diversity Pool [19], and Condor [90]. 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 Paired Learners (PL) [2], 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.

Finally, there are drift detection methods, which can be further divided into two groups. There are tests that detect degradation in prediction accuracy with respect to a given model, which includes tests such as DDM [28], EDDM [3], ADWIN [6], PERM [35], FHDDM [70], and MDDM [72]. Alternatively, tests can detect change in the underlying data distribution [50, 78]. The connection between the two approaches is made in [41]. In this thesis, 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 [35], 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 [1], in which a slow but accurate

Table 4.1: Commonly used symbols pertaining to DriftSurf

$\mathbf{X}_t$	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 4.1 for entering the reactive state
δ'	the threshold in condition 4.2 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

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.

4.2 Problem Setup

We seek an adaptive learning algorithm A with high prediction accuracy at each time step. Let $\mathbf{w}$ be the solution learned by an algorithm A over stream segment $\mathcal{S} = \mathcal{S}_{t_1,t_2}$. Recall from §2 that 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).

We formalize the latter as A being "risk-competitive" with an idealized algorithm Aware that has oracle knowledge of when drifts occur. At each drift, Aware restarts the predictive model to a random initial point and trains it over data points that arrive after the drift.

Definition 5. *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 $n_{t_d,t} = |\mathcal{S}_{t_d,t}|$.*

Risk-competitiveness to Aware implies A is asymptotically optimal in terms of its total error, despite concept drifts.

Table 4.1 summarizes the symbols commonly used throughout this chapter.

4.3 Algorithm

We present our algorithm DriftSurf for adaptively learning from streaming data that may experience drift. Incremental learning algorithms like STRSAGA (§3) 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 §4.1, prior work [3, 6, 28, 35, 70, 72] 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 2, which is executed when DriftSurf is in the stable state, and Algorithm 3, 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,

Algorithm 2 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 4.1 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 3 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**
```

given by the following condition:

$$\mathcal{R}_{\mathbf{X}_t}(\mathbf{w}_b) > \mathcal{R}_b + \delta, \text{where } b = \underset{b \in b1, b2}{\arg\min}\, \mathcal{R}_b \tag{4.1}$$

and δ is a predetermined threshold that represents the tolerance in performance degradation (the selection of δ is discussed in §4.7), 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 4.1 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 (for example, using STRSAGA). 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 3), DriftSurf checks that condition 4.1 still holds (to handle a corner

Algorithm 3 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 2, except that $W1$, $W2$ are the two most-
// recent windows started *before* the current reactive state.
**if** condition 4.1 does NOT hold **then** {Early exit}
    state $\leftarrow$ stable
    execute Algorithm 2 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 4.2 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**
```

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 4.2 holds:

$$\mathcal{R}_T(\mathbf{w}'_f) < \mathcal{R}_T(\mathbf{w}_b) - \delta', \text{where } b = \underset{b \in b1, b2}{\arg\min} \mathcal{R}_b \tag{4.2}$$

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 4.1 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 2 and 3 are generic in the individual base learner. For the experimental evaluation in §4.7, we focus on base learners where the update process is STRSAGA (§3). The time and space complexity of DriftSurf is within a constant factor of the individual base learner.

4.4 Analysis

In this section, we show that DriftSurf achieves goals **G1** and **G2** from §4.2.

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 asymptotically by Lemma 3 in §3.2.1.

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 model snapshots 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.

4.4.1 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 24. *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 (Section 4.5.1), 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 25 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 4.2.

Lemma 25. *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 (Section 4.5.1), 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 $2W + c_4/(\epsilon - c_3) \le t < t_{d_1}$, 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 25).*

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 $2W+c_4/(\epsilon-c_3) \le t < t_{d_1}$. This is a special case of the forthcoming Theorem 5 in §4.4.2.

In addition, it follows from Lemma 24 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 4.1 holds (this algorithm is DriftSurf without the reactive state).

Lemma 26. *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.

4.4.2 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 §4.5.2. 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 27. *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 28. *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 27 and 28, 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 34 in §4.5.2.

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 5. *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* $t_{d_i} + 3W + c_4/(\epsilon_s - c_3) \le t < t_{d_{i+1}}$*, 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 §4.5.2.

4.5 Proofs

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

Observation 2. *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 2 follows from Assumption 1 and applying Equation 2.2 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 6. *(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 7. *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 §4.4 that establish the conditions under which DriftSurf is risk-competitive with Aware both in a stationary environment and in the presence of abrupt drifts.

4.5.1 In a Stationary Environment

In §4.4.1, 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 29. *(Generalized statement of Lemma 24.) 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 7 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 6 (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 4.1, which we later show in Lemma 31. 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 30. In the proof we will use the following theorem (Bennett's inequality).

Theorem 8. *(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 30. *(Generalized statement of Lemma 25.) 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 2 and the assumption $\mathcal{R}_I^* = 0$.

We apply Theorem 8 (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 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 25 (same as in Lemma 30)).*

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 4.1 holds.

Lemma 31. *(Generalized statement of Lemma 26.) 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 29. Therefore,

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

□

4.5.2 In Presence of Abrupt Drifts

For the case of abrupt drift, we first bound the recovery time for DriftSurf through Lemmas 32 and 34, and then establish risk-competitiveness after recovery in Theorem 5.

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 32. *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 32, 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 33 addresses this problem.

Lemma 33. *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 [43] and by setting parameter t (defined in their proof) to $\frac{p_d}{2}$. □

Now, by putting together the results of Lemma 32 and 33, we can now provide a probabilistic upper bound on the recovery time as follows:

Lemma 34. *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 32, 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 33, 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 33 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 37.

On the other hand, we have:

Lemma 35. *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 36. *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 36, and denoting the probability of not exiting the reactive state by ϵ_3, we can generalize Lemma 34 as follows:

Lemma 37. *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* $\epsilon'_r = \epsilon_1 + \epsilon_2 + \epsilon_3$, $\frac{1}{\epsilon_1} \leq 1 + \frac{(\frac{W q_d}{rg} - 1)^2}{1 - q_d}$, $\epsilon_2 \geq e^{-\frac{W}{rg}}$, $g = \frac{4 \ln 2}{p_d r} + 1$*, and* $\epsilon_3 \leq p_d$.

One more fact we will use before we prove Theorem 5 on risk-competitiveness is the the following Lemma from [21], which is a consequence of the generalization bound in Equation 2.2.

Lemma 38. (THEOREM 3 IN [21]) *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 5 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 5. *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 34 (and where c_3 and c_4 are the same constants in Corollary 1).*

Proof. By Lemma 34, 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.

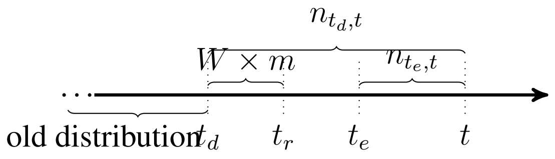


Figure 4.1: A drift happens at time t_d. DriftSurf recovers by time t_r. The current predictive model is initialized at time t_e.

To show DriftSurf is $\frac{7}{4^{1-\alpha}}$-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 38 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 37.

4.6 Experimental Setup

We conduct experiments comparing DriftSurf against (i) Standard Drift Detection (StandardDD) to evaluate the effectiveness of DriftSurf's stable-state / reactive-state approach, (ii) the idealized Aware, (iii) the state-of-the-art drift detection MDDM, and (iv) the state-of-the-art 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 §4.6.1.

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 §4.6.2.

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 (§3) as the update process. More details on this base learner, hyperparameter settings, and additional base learners, are provided in § 4.6.3. All reported results of the misclassification rates represent the median over five trials.

4.6.1 Algorithms Evaluated

In our experimental evaluation, we compare our algorithm DriftSurf to MDDM [72] and AUE [13], 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" [72]. 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 sec:dsurf-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 [13].

Another ensemble algorithm we compare against is the Paired Learners (PL) algorithm [2]. 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 §4.3, the threshold for condition 4.1 is $\delta = 0.1$, and the threshold for condition 4.2 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 §4.4). 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. Note that 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 (§3.2). 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 [90]. 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.

Table 4.2: 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

4.6.2 Datasets

Our experiments use the 5 synthetic, 2 semi-synthetic and 3 real-world datasets shown in Table 4.2 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 [8]: This dataset is generated using the Massive Online Analysis (MOA) framework. There are three attributes in $[0, 10]$. The label is determined by $x_1 + x_2 \leq \theta_j$ where j corresponds to 4 different concepts, $\theta_1 = 9, \theta_2 = 8, \theta_3 = 7, \theta_4 = 9.5$ (the third attribute x_3 is not correlated with the label). We synthetically generated 25000 points from each concept in the order 3, 2, 4, 1, following the example from the MOA manual. We experimented on four different datasets varying the amount of noise, SEA0, SEA10, SEA20, SEA30, corresponding to 0%, 10%, 20%, and 30% of the labels being swapped during the generation of the dataset. SEA-gradual is generated by generating samples from two concepts (the first two concepts discussed above) during the drift period.
- Hyperplane [8]: 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 [71]: 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 [71]: 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 [71]: 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 [59]: 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 [24]: 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) [42]: 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 [37]: 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 [22]: This real world data set contains records of hourly power supply of

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

The type of drift in each dataset is detailed in Table 4.3. 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 [13]. Still, we had to mark the drift times for the implementation of Aware, which resets the model whenever drifts occur. We chose these times by observing the misclassification rates of an oblivious algorithm that is not designed to adapt to drift, and noting for which time steps there was a significant increase in misclassifications on the newly arrived batch.

Table 4.3: 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]

4.6.3 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 $f_{(x,y)}(\mathbf{w}) = \log(1+\exp(-y\mathbf{w}^T x))+\frac{\mu}{2}||\mathbf{w}||_2^2$.

Recall there are two hyperparameters relevant for STRSAGA for logistic regression, 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.)

As with the experiments run in §3.4, the parameter choices of μ and η were chosen by grid search, optimizing for validation set error. The parameters we chose are given in Table 4.4. 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 [68], 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: $r = 6, \delta = 0.05, \delta' = \delta/10, W = 50$.

Table 4.4: Hyperparameters and batch sizes

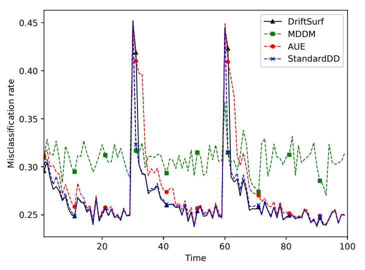
DATASET	REGULARIZATION μ	STEP SIZE η	BATCH SIZE m
SEA (ALL)	10^{-2}	1×10^{-3}	1000
HYPER-SLOW	10^{-3}	1×10^{-1}	1000
HYPER-FAST	10^{-3}	1×10^{-2}	1000
SINE1	10^{-3}	2×10^{-1}	100
MIXED	10^{-3}	1×10^{-1}	1000
CIRCLES	10^{-3}	1×10^{-1}	100
RCV1	10^{-5}	5×10^{-1}	202
COVERTYPE	10^{-4}	5×10^{-3}	5810
AIRLINE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

In general, the batch size is determined by the rate of arrival of new data points, and hence not a hyperparameter to be tuned. For simplicity, we assume that data arrive over the course of b time steps in equally-sized batches containing $m = (\text{dataset size})/b$ points, where $b = 100$ for all datasets other than Electricity. For the case of Electricity, we defined the number of time steps $b = 34$ so that one time step corresponds to 28 days of the collected data, and was a scale where we could visually observe drift in the results. The resulting batch sizes are shown in the last column of Table 4.4.

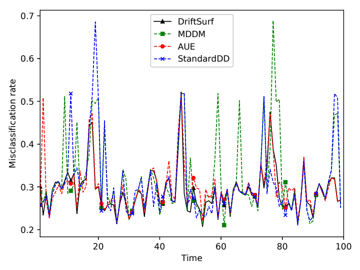
4.7 Main Experimental Results

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.

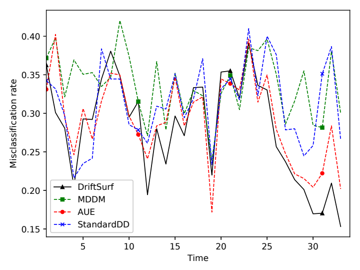
We present the misclassification rates at each time step on the CoverType, PowerSupply, and Electricity datasets (see § 4.8.1 for other datasets) in Figure 4.2. A drift occurs at times 30 and 60 in CoverType, at times 17, 47, and 76 in PowerSupply, and at time 20 in Electricity. We observe DriftSurf 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



(a) CoverType



(b) PowerSupply



(c) Electricity

Figure 4.2: Misclassification rate over time for CoverType, PowerSupply, and Electricity

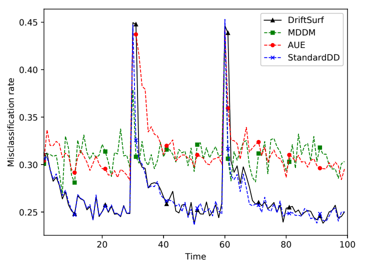


Figure 4.3: CoverType (update steps divided among each model)

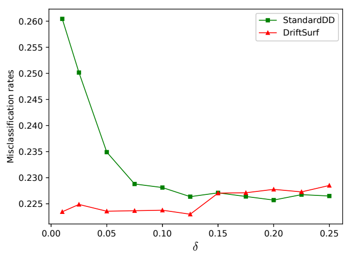


Figure 4.4: All datasets, DriftSurf and StandardDD under varying threshold δ

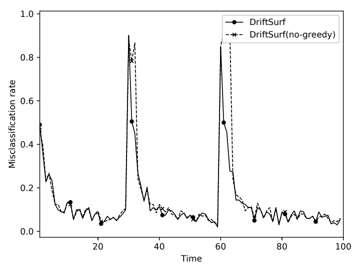


Figure 4.5: RCV1, DriftSurf and DriftSurf (no-greedy)

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 4.5 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 4.5. For some datasets (Airline, Hyper-Slow) AUE outperforms DriftSurf. A factor is the different computational power (e.g., number of gradient

Table 4.5: Average 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

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 §4.8.2.) The misclassification rates for each dataset are presented in Table 4.6, where we observe DriftSurf dominates AUE across all datasets. The CoverType dataset is visualized in Figure 4.3 (compare to Figure 4.2a 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.

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 24 and 27 in §4.4), which can be difficult to tune without knowing the frequency and magnitude of drifts in advance. Across a range of δ, Figure 4.4 shows the misclassification rates for DriftSurf compared to StandardDD, averaged across the datasets in Table 4.5 (see §4.8.3 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

Table 4.6: Average misclassification rate (update steps divided among each model)

ALGORITHM / DATASET	AUE	MDDM	Stand-ardDD	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

visualized on the RCV1 dataset in Figure 4.5, 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 §4.8.4, where we observe that DriftSurf performs equal or better than DriftSurf (no-greedy) on 11 of the 13 datasets in Table 4.5, and, averaging over all the datasets, has a misclassification rate of 0.221 compared to 0.229.

§4.8 contains additional experimental results. In §4.8.5, 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. §4.8.6 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. §4.8.7 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, §4.8.8 reports additional numerical results on the recovery time of each algorithm.

4.8 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.

Table 4.7: Average misclassification rate ($\rho = 2m$ for each model)

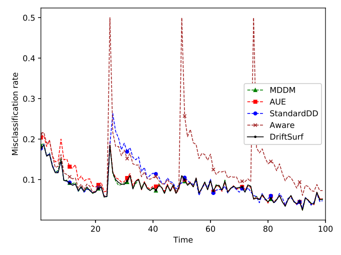
DATASET	Aware	DriftSurf	StandardDD	MDDM-G	MDDM-A	MDDM-E	AUE	PL	1PASS-SGD	OBL
SEA0	0.137	0.086	0.097	**0.086**	0.090	0.087	0.093	**0.085**	0.131	0.110
SEA10	0.197	**0.160**	0.168	0.180	0.166	0.172	0.163	0.161	0.188	0.176
SEA20	0.264	**0.243**	0.249	0.289	0.278	0.289	0.245	**0.243**	0.267	0.254
SEA30	0.350	**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

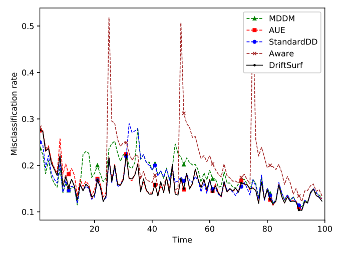
4.8.1 Equal Computational Power for Each Model

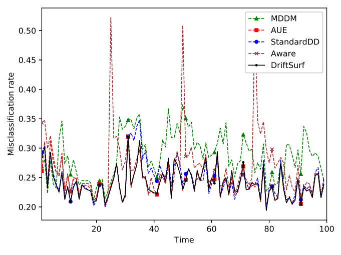
First, we compare each algorithm under the setting where at every time step, each algorithm uses $\rho = 2m$ update steps (gradient computations) to update each of its models. In this case, the total computational power used varies per algorithm. For example, at each time step AUE maintains an ensemble 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 §4.8.2, 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.)

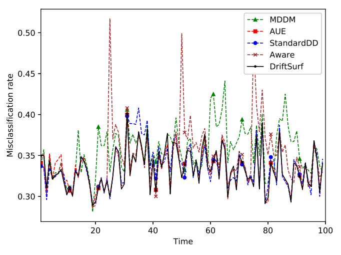
We present the misclassification rates at each time step over the new batch in Figure 4.6, and the average misclassification rate over all time steps is summarized in Table 4.7. (These results are a superset of those presented in Figure 4.2 and Table 4.5 from §4.7). The advantage of DriftSurf over MDDM is most evident on the noisy versions of SEA (also shown in Figure 4.7), 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.

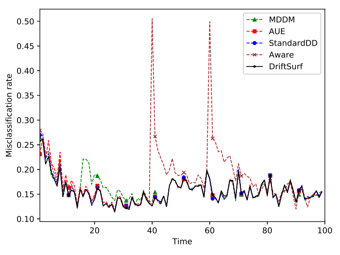
For sharp drifts when immediately switching to a new model is desirable, we observe, most evident on SINE1, that MDDM is the fastest to adapt, followed shortly by DriftSurf, then PL, with AUE lagging behind. CoverType also is a clear example where DriftSurf and StandardDD adapt faster than AUE (but MDDM 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 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

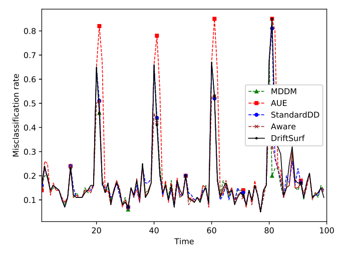

(a) SEA0

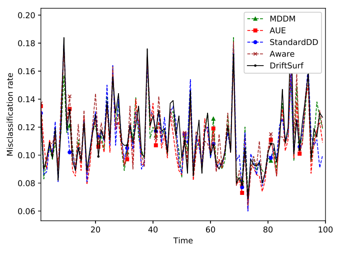

(b) SEA10

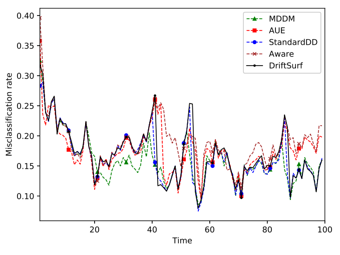

(c) SEA20

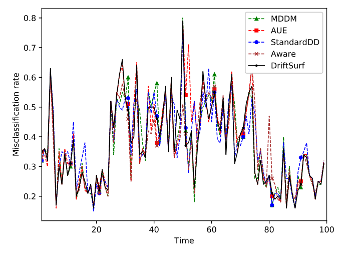

(d) SEA30

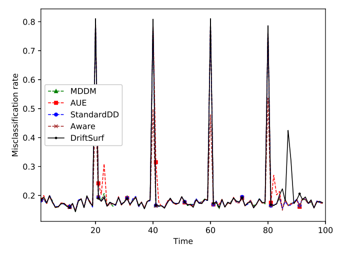

(e) SEA-gradual

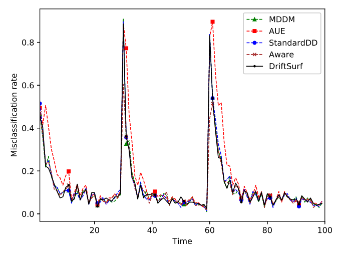

(f) SINE1

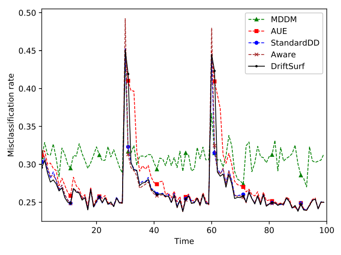

(g) HyperPlane-slow

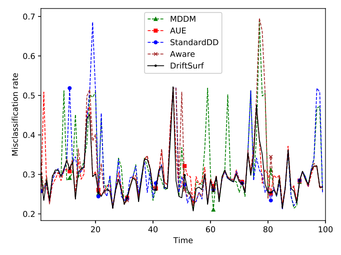

(h) HyperPlane-fast

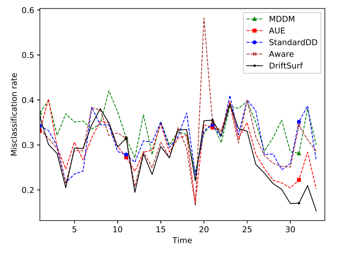

(i) Circles


(j) Mixed


(k) RCV1


(l) CoverType


(m) PowerSupply


(n) Electricity

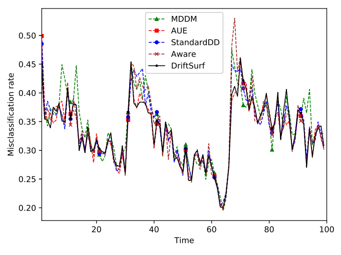

(o) Airline

Figure 4.6: Misclassification rate over time ($\rho = 2m$ for each model)

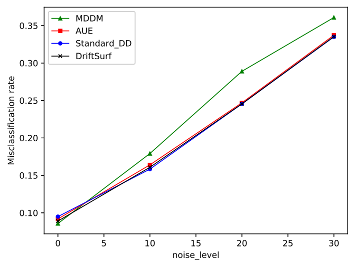


Figure 4.7: Average misclassification rate for SEA dataset with different levels of noise

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.

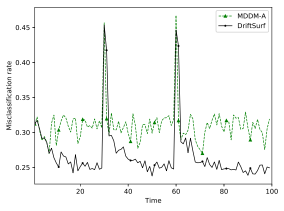
There are two major advantages of DriftSurf and AUE not immediately switching to the latest model: (i) there are drifts for which switching to a new model is not desired because the older model can still provide good accuracy, and (ii) delaying the switch to a new model can be desired if the new model 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 4.7, 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.

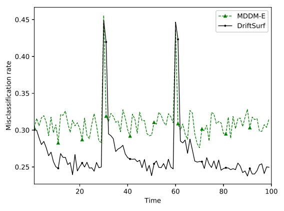
The second advantage previously mentioned, of delaying the switch to the new model, is best exemplified on Airline. Immediately after the two drifts, DriftSurf and AUE are the best performers, followed by StandardDD and MDDM, and then Aware. Immediately after the drift, DriftSurf continues to use the 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.

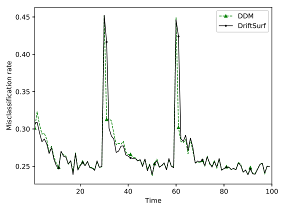
Table 4.7 includes results for MDDM-G (what we use generally for MDDM), as well as two other MDDM variants, MDDM-A and MDDM-E, for a more thorough comparison. The average misclassification rates were similar across each dataset, with no single MDDM variant that consistently outperformed the others. Given the poor performance of MDDM on CoverType, we re-did the experiment on CoverType with two other drift detection methods, DDM [28] and EDDM [3] to investigate further. In Figure 4.8, we observed DDM accurately detected the two drifts, but EDDM also suffered with continual false positives.



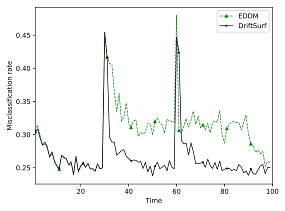
(a) MDDM-A



(b) MDDM-E



(c) DDM



(d) EDDM

Figure 4.8: CoverType dataset, comparing different drift detectors ($\rho = 2m$ for each model)

Table 4.7 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.

4.8.2 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 4.9 for the comparison of DriftSurf, Aware, MDDM, and AUE and in Figure 4.10 for the additional algorithmic comparisons against two ensemble methods, AUE ($k = 2$) and Condor. The average

Table 4.8: Average 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

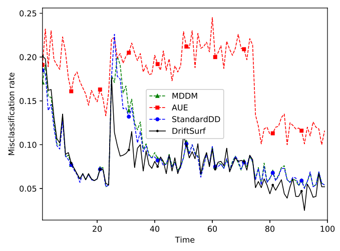
Table 4.9: Average misclassification rate ($\rho = 2m$ divided among all models of each algorithm)

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

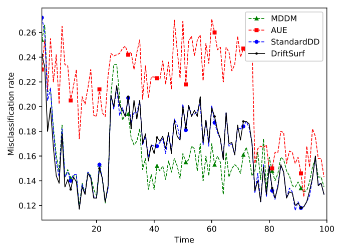
over time is in Table 4.8. For the case $\rho = 2m$, the misclassification rate at each time is shown in Figure 4.11, and the average over time is in Table 4.9.

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 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.

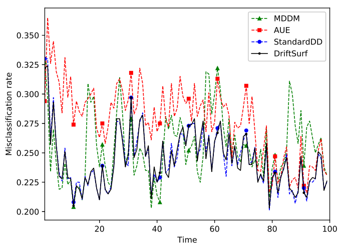
We observe DriftSurf compares favorably to MDDM and StandardDD on the same datasets as it did in the undivided ρ case. However, MDDM's and StandardDD's advantages are magnified on SINE1 and RCV1, the datasets with sharp drifts that were clear to detect, and when immediate switching to the new model was desired. On PowerSupply, we observe that the false positives are



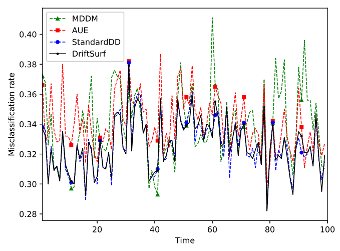
(a) SEA0



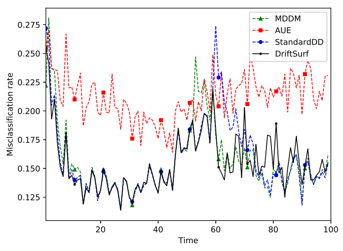
(b) SEA10



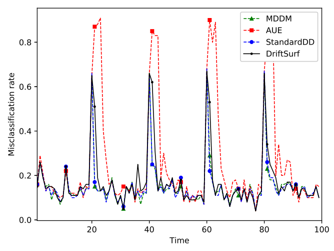
(c) SEA20



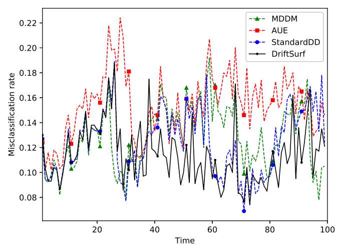
(d) SEA30



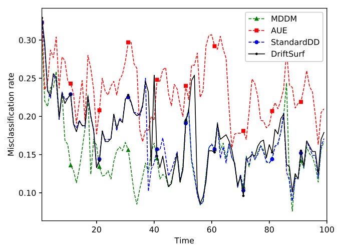
(e) SEA-gradual



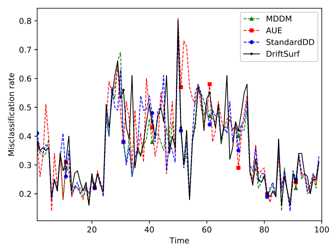
(f) SINE1



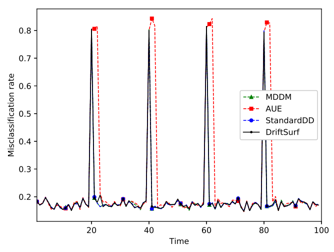
(g) HyperPlane-slow



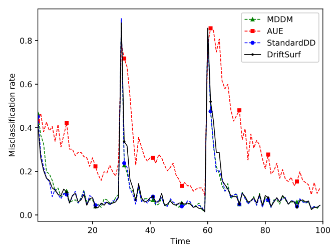
(h) HyperPlane-fast



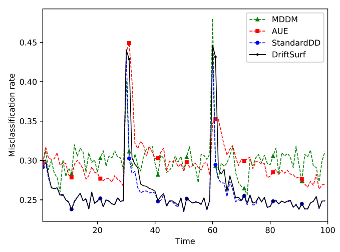
(i) Circles



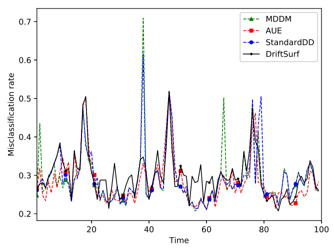
(j) Mixed



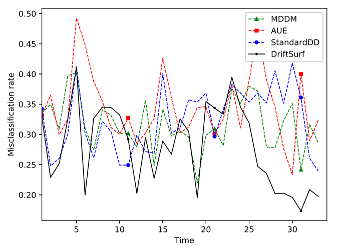
(k) RCV1



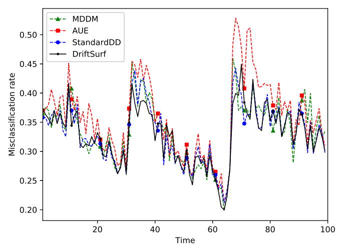
(l) CoverType



(m) PowerSupply

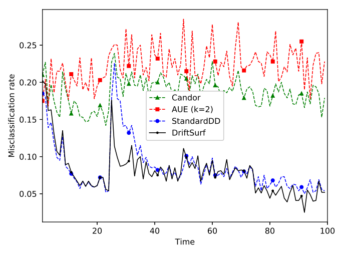


(n) Electricity

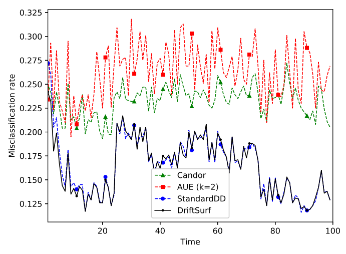


(o) Airline

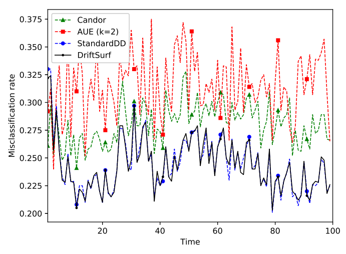
Figure 4.9: Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm) comparing Aware, DriftSurf, AUE, and MDDM



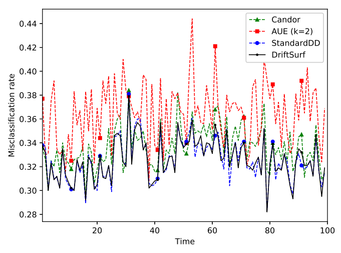
(a) SEA0



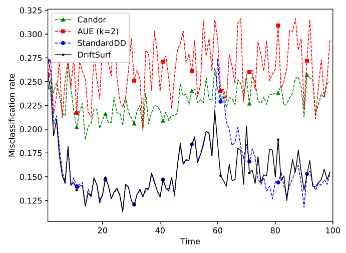
(b) SEA10



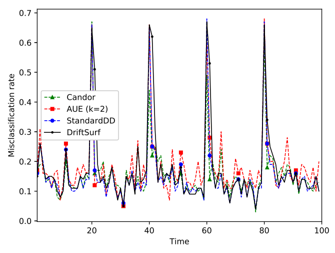
(c) SEA20



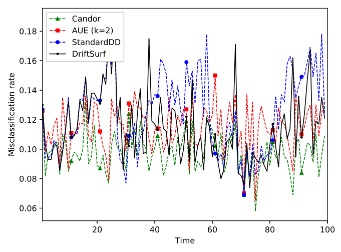
(d) SEA30



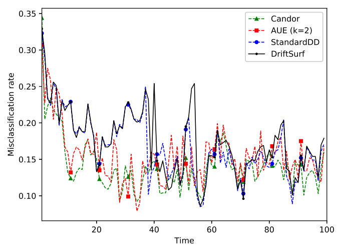
(e) SEA-gradual



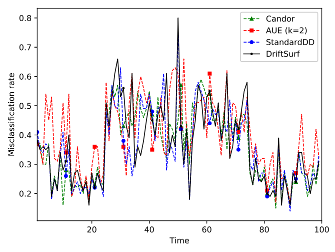
(f) SINE1



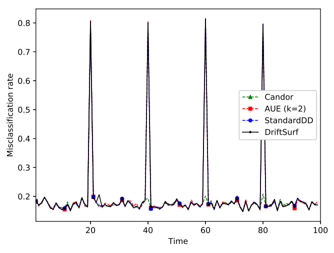
(g) HyperPlane-slow



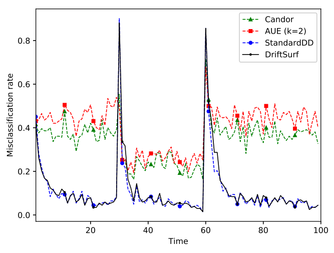
(h) HyperPlane-fast



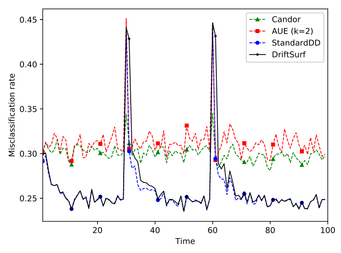
(i) Circles



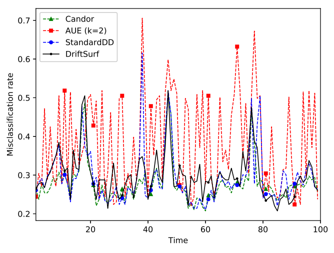
(j) Mixed



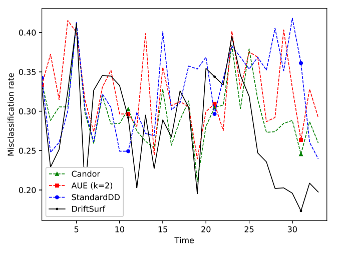
(k) RCV1



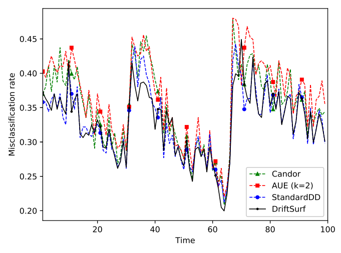
(l) CoverType



(m) PowerSupply

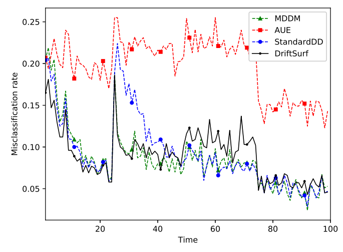


(n) Electricity

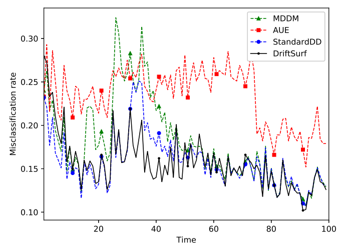


(o) Airline

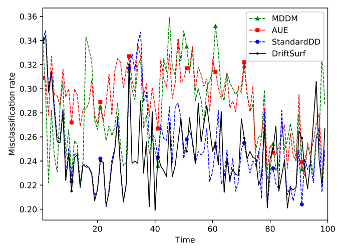
Figure 4.10: Misclassification rate over time ($\rho = 4m$ divided among all models of each algorithm) comparing Aware, DriftSurf, AUE ($k = 2$), and Condor



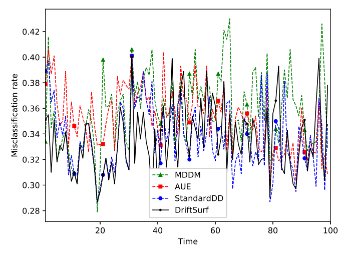
(a) SEA0



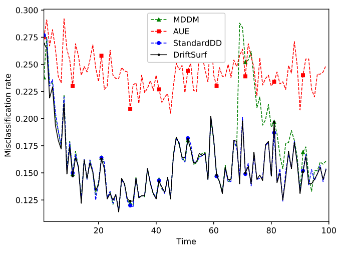
(b) SEA10



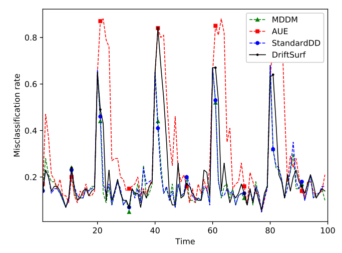
(c) SEA20



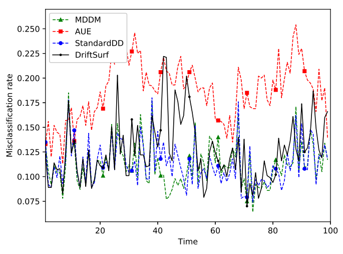
(d) SEA30



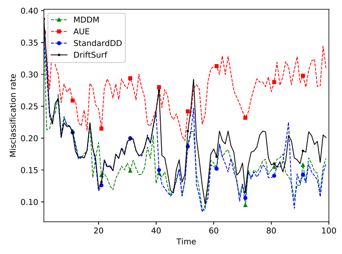
(e) SEA-gradual



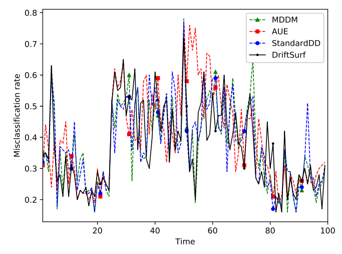
(f) SINE1



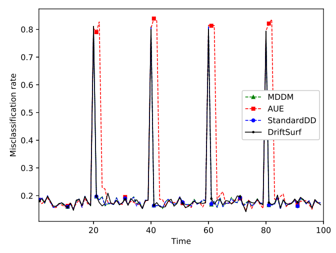
(g) HyperPlane-slow



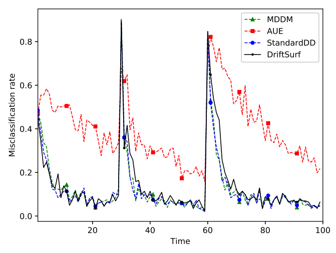
(h) HyperPlane-fast



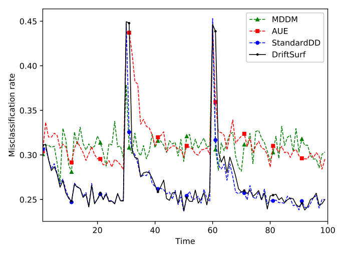
(i) Circles



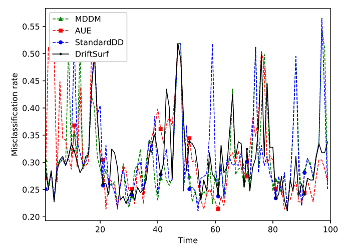
(j) Mixed



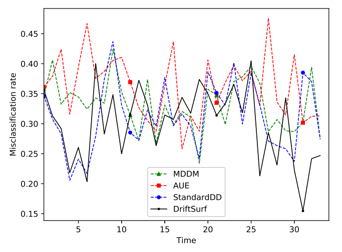
(k) RCV1



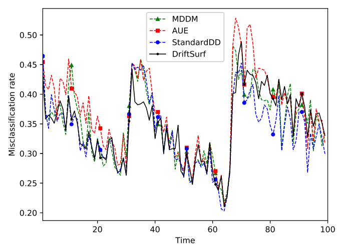
(l) CoverType



(m) PowerSupply



(n) Electricity



(o) Airline

Figure 4.11: Misclassification rate over time ($\rho = 2m$ divided among all models of each algorithm)

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 4.8 and 4.9, 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 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 4.8 and 4.9, 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.

4.8.3 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 4.1 to decide whether to reset the model) under a range of settings of δ and plotted the misclassification rate for each dataset in Figure 4.12.

As we observed in §4.7 from the results averaged over the datasets in Figure 4.4, 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.

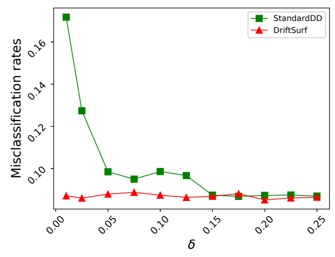
In some other datasets with large, abrupt drifts (SINE1, Mixed, CoverType), StandardDD outperforms DriftSurf, especially for larger values for δ. In such cases, the right thing to do is to reset the model, but DriftSurf suffers from a delayed reaction to such drifts it has to enter reactive state and 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.

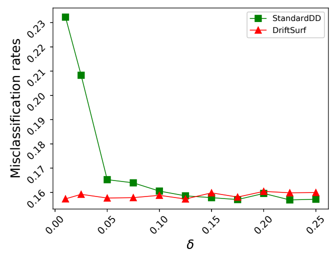
4.8.4 Evaluation of Greedy Reactive State

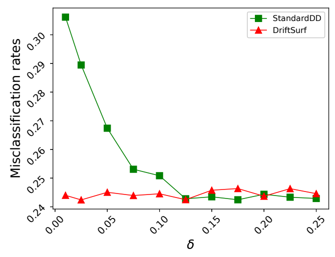
This §includes results for the comparison of DriftSurf to DriftSurf (no-greedy). Recall that DriftSurf uses greedy prediction in the reactive state, meaning that the predictive model used at one time step is the model with better performance from the previous time step, while DriftSurf (no-greedy) only 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 4.10 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.

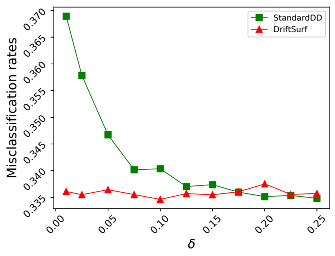
Table 4.10: Average misclassification rate - DriftSurf vs DriftSurf (no-greedy) ($\rho = 2m$ for each model)

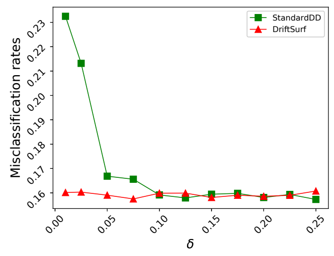
DATASET	DriftSurf	DriftSurf (NO-GREEDY)
SEA0	0.087	**0.085**
SEA10	0.161	**0.158**
SEA20	0.247	**0.246**
SEA30	**0.335**	0.336
SEA-GRADUAL	**0.158**	0.159
HYPER-SLOW	**0.117**	0.118
HYPER-FAST	**0.173**	0.177
SINE1	**0.191**	0.220
MIXED	**0.204**	0.238
CIRCLES	**0.371**	0.376
RCV1	**0.134**	0.158
COVERTYPE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

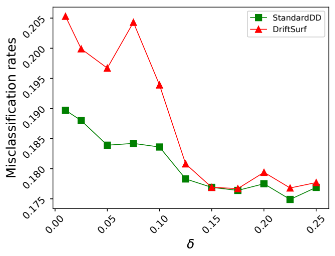

(a) SEA0

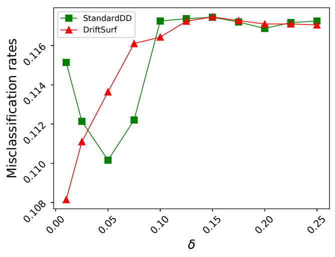

(b) SEA10

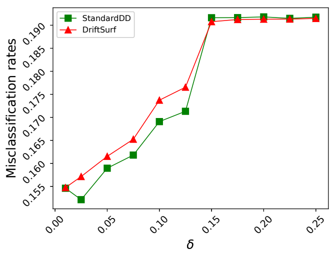

(c) SEA20

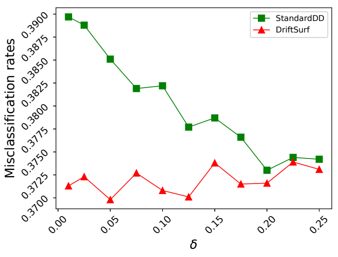

(d) SEA30

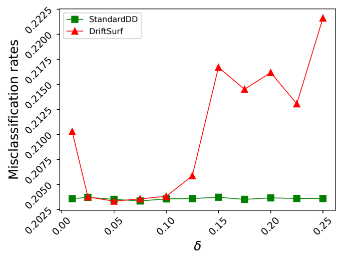

(e) SEA-gradual

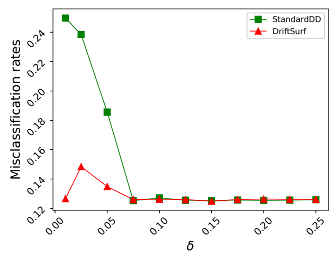

(f) SINE1

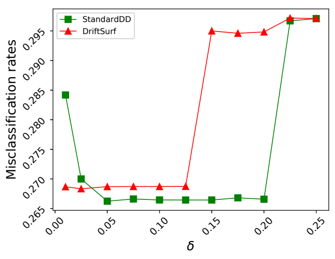

(g) HyperPlane-slow

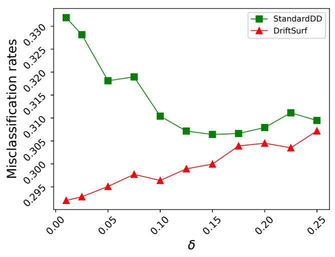

(h) HyperPlane-fast

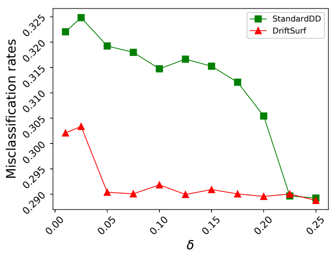

(i) Circles


(j) Mixed


(k) RCV1


(l) CoverType


(m) PowerSupply


(n) Electricity

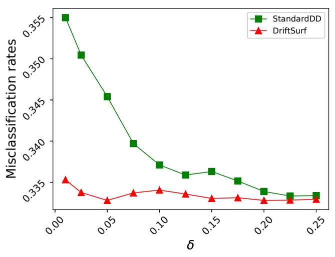
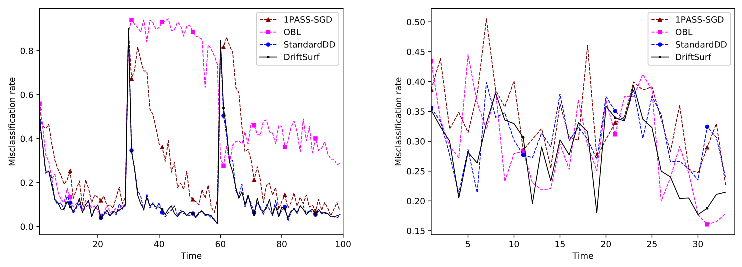

(o) Airline

Figure 4.12: δ-sensitivity ($\rho = 2m$ for each model)

4.8.5 Comparison to 1PASS-SGD and Oblivious

Figure 4.13 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 4.7. In the case of the large, abrupt drift in RCV1, we observe that 1PASS-SGD and especially oblivious have poor performance after drift. The oblivious algorithm continues to re-sample the data from the older distributions, and leads to a model with random, or worse than random, accuracy on the current distribution. Even for 1PASS-SGD, which only trains over data from the most recent time step, we observe its convergence rate is slow after a drift, where its previous training on the old data still hinders it. On the Electricity data with a more subtle drift, we observe that oblivious is actually the best performing algorithm, as discussed earlier, because data from all over time can be trained and fit by a single model. However, 1PASS-SGD still has lower accuracy because, as a single pass method, it uses only m update steps at each time even when $\rho = 2m$ are available to the other algorithms, and also because SGD has a slower convergence rate than the variance-reduced method STRSAGA.



(a) RCV1 (b) Electricity

Figure 4.13: Misclassification rate over time ($\rho = 2m$ for each model)

4.8.6 Using SGD as the Update Process

We primarily use the variance-reduced STRSAGA as the update process because it achieves fast convergence in the stationary setting, as we showed both through theory and experiments in §3. 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 4.11 shows the average misclassification rate for the case where $\rho = 2m$ update steps are used for each model.

As the results presented in Table 4.11 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 rate of drift detection. AUE, on the other hand,

Table 4.11: Average 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**

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 4.12 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 4.7 and Table 4.11 confirms the advantage of using STRSAGA over SGD as the update process.

4.8.7 Using Hoeffding Trees and Naive Bayes as Base Learners

For the experiments in §4.8.1–§4.8.6, 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 Tree (HAT) [7]. 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 [68].

The results when using HT are shown in Table 4.13 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

Table 4.12: Average 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**

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 4.14. 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.

4.8.8 Recovery Time Analysis

In §4.2, 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 here is the 95%-recovery-time metric proposed by [80]. 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 4.15 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

Table 4.13: Average 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**

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 4.14: Average 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

Table 4.15: 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

Chapter 5

Learning from non-IID data over time and distributed in space with FedDrift

Federated learning (FL) [55, 66] is a popular machine learning paradigm that enables collaborative training without sharing raw training data. FL is crucial in the era of pervasive computing, where massive IoT and mobile phones continuously generate relevant data that cannot be easily shared due to privacy and communication constraints. FL also enables different organizations such as hospitals [74] and retail stores [89] to jointly obtain valuable insights while preserving data privacy. FL has become an important technology in the real world with massive deployments (500+ million installations on Android devices) as well as a growing market with many solution providers [64].

Concept drift in FL poses new fundamental challenges where data is heterogeneous in two dimensions, both over *time* and across different *clients*. When different clients experience the data drift *at different times*, no single global model can perform well for all clients. Similarly, when *multiple concepts exist simultaneously*, no centralized training decision works well for all clients. Several recent works have recognized the problem of concept drift in FL and proposed solutions that adapt learning rates or add regularization terms [17, 18, 34, 62]. However, they consider only restricted cases, shown in Figure 5.1 (repeated from §1 for convenience). Under more general settings, such as the distributed drift patterns depicted in Figures 5.2 and 5.3 (repeated from

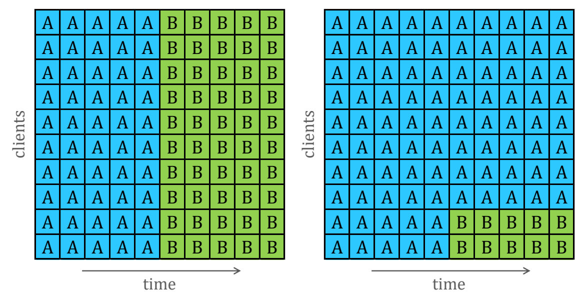


Figure 5.1: Simplistic drifts studied in prior work. (left) Simultaneous timing. (right) One majority concept.

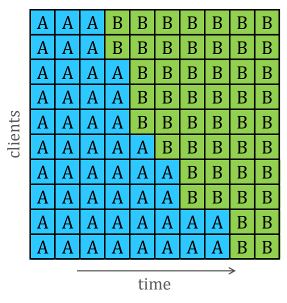


Figure 5.2: Distributed drift pattern (2 concepts).

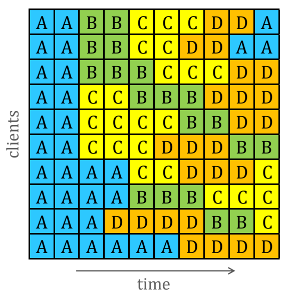


Figure 5.3: Distributed drift pattern (4 concepts).

§1), these existing solutions suffer due to their use of a single global model, and hence fail to generally address the aforementioned challenges. Similarly, applying any prior drift adaptation centrally at the server also suffers due to the use of a single global model (and at best a single global drift detection test), particularly with poor accuracy during the transition period (time steps 4–8 in Figure 5.2, see Figure 5.6(left) in §5.6). Meanwhile, centralized ensemble methods that use multiple models for adapting to drift fare no better—in response to a localized data drift, a newly created global model is trained over a mixture of concepts. The models in an ensemble are distinguished solely over time, and do not account for heterogeneity across clients.

This chapter introduces the first FL solution that employs multiple models to address FL under distributed concept drift. Our solution aims to create one model for each new concept so that all clients under the same concept can train that model collaboratively, similar to what is done for *personalized* or *clustered* FL [25, 32, 33, 63, 77]. We introduce two new algorithms for model creation and client clustering so that our solution addresses all the challenges of distributed concept drift. Our first algorithm, FedDrift-Eager, is a specialized algorithm that creates models based on drift detection. FedDrift-Eager is effective if new concepts are introduced one at a time. Our second algorithm, FedDrift, is a general algorithm that leverages hierarchical clustering to adaptively determine the appropriate number of models. FedDrift isolates drifted clients and conservatively merges clients via hierarchical clustering, so that FedDrift can effectively handle general cases where an unknown number of new concepts emerge simultaneously.

We empirically evaluate our solution using four popular concept drift datasets, and we compare our solution against state-of-the-art centralized concept drift solutions (KUE [14] and DriftSurf (§4) and a recent FL solution that adapts to concept drifts (Adaptive-FedAvg [15]). Our results show that (i) FedDrift-Eager and FedDrift consistently achieve much higher and more stable model accuracy than existing baselines (average accuracy 93% vs. 88% for the best baseline, across six dataset/drift combinations); (ii) FedDrift performs much better than FedDrift-Eager when multiple new concepts are introduced at the same time; and (iii) our solution achieves a similar model accuracy as Oracle (94% accuracy), an idealized algorithm that knows the timing and distribution of concept drifts. On the real-world drift in the FMoW dataset [52], FedDrift achieves 64% accuracy vs. 58% accuracy for the best baselines.

5.1 Related Work

Drift in FL has so far seen only preliminary study. One line of work considers the setting where there is one concept in the system to be learned (either like the example in Figure 5.1(right) when a minority of clients drift, or when clients observe the main concept under random noise), and seek to speed up the convergence of a model for that one concept by suppressing clients with heterogeneous data via regularization [18, 34] or drift detection [62]. When it comes to adapting to a new concept over time, we are only aware of two works, and both only consider drifts with uniform timing (Figure 5.1(left)). First, Casado et al. [17] consider only the *virtual drift* setting (where the labeling $\mathcal{P}(y|x)$ is fixed and only $\mathcal{P}(x)$ changes) and uses drift detection to partition data from distinct concepts, in order to train a single model accurately in the course of revisiting each partition (i.e., rehearsal). Second, Canonaco et al. [15] propose Adaptive-FedAvg, in which the server tunes the learning rate used by all clients based on the variability across updates, with the goal of reacting fast when drift occurs while also achieving stable performance in the absence of drift. We compare against Adaptive-FedAvg in our evaluation.

Our solution to drift in FL (§5.3, §5.4) relies on learning multiple models, which has been studied in prior work on *personalized FL* and *clustered FL*. Clients with similar data can be grouped into clusters, where each cluster trains its global model [12, 25, 32, 33, 63, 77]. As we extend the problem of data heterogeneity in FL with the dimension of time, we train multiple models with the algorithm in §5.3, which is inspired by the prior clustering algorithms IFCA [33] and HypCluster [63]. This serves as the starting point of our solution, where our main contribution is the creation of new clusters as new concepts arrive over time. Our solution in §5.4 to handle an unknown number of concepts relies on hierarchical clustering, which has been studied in static FL by Briggs et al. [12]. In this prior work, it is unclear how to set the distance threshold at which to stop merging clusters. In contrast, our approach has the advantage that the stop merging criterion is identical to the drift detection threshold, which has an intuitive interpretation of performance loss.

5.2 Problem Setup

We consider a FL setting with P clients, assumed to be stateful and participating at each round, and a central server that coordinates training across the clients. Training data are decentralized and arriving over time. The data at each client $c = 1, \ldots, P$ and each time $t = 1, 2, \ldots$ are sampled from a distribution (concept) $\mathcal{P}_c^{(t)}(x, y)$. We consider that data may be non-IID in two dimensions, varying across clients and across time. We say that there is a concept drift at time t and at client c if $\mathcal{P}_c^{(t)} \neq \mathcal{P}_c^{(t-1)}$ (the standard definition of drift with respect to a single node [29]).

One option is to learn a single global model h (which is a function of time but is notationally suppressed) that is used for inference at all clients. In this case, the objective is to minimize over all time t, $\sum_{c=1}^{P} \mathbb{E}_{(x,y)\sim\mathcal{P}_c^{(t)}}[\ell(h(x), y)]$, where ℓ is the loss function. However, the optimal single model may not be well-suited in the presence of concept drifts. By decomposing the joint distribution $\mathcal{P}(x, y) = \mathcal{P}(x)\mathcal{P}(y|x)$, we distinguish between drifts where only $\mathcal{P}(x)$ changes (called virtual drift or covariate drift [47, 82, 85]) versus drifts where the feature-to-label mapping $\mathcal{P}(y|x)$ changes (called real drift or concept drift [85]). While the optimal single model can perform well under the former case (although achieving fast convergence still requires a specialized strategy; e.g., FedProx [60]), lower loss can often be obtained under the latter case by using specialized models for different concepts.

The multiple-model option is to learn a set of global models $\{h_m\}$, and a time-varying clustering of clients. For notation, we denote the cluster identities by one-hot vectors $w_c^{(t)}$, where $w_{c,m}^{(t)} = 1$ when the client c at time t uses model h_m for inference; we denote $h_{w_c^{(t)}}$ to represent the unique model h_m where $w_{c,m}^{(t)} = 1$. The objective is to minimize over all time t, $\sum_{c=1}^{P} \mathbb{E}_{(x,y)\sim\mathcal{P}_c^{(t)}}[\ell(h_{w_c^{(t)}}(x), y)]$.

Table 5.1: Commonly used symbols pertaining to FedDrift

τ	current time (prior time indexed by t)
P	# clients (indexed by c)
M	# global models (indexed by m)
R	# communication rounds (indexed by i)
K	# local steps per model per round (by j)
$S_c^{(t)}$	new data arriving at client c at time t
$N_c^{(t)}$	$= \lvert S_c^{(t)} \rvert$
B	minibatch size
η	step size
h_m	global model m
$h_{c,m}$	local update of h_m by client c
$w_{c,m}^{(t)}$	is $S_c^{(t)}$ used to update h_m?

5.3 Multiple-Model Training in FL

As discussed above, distributed concept drift often means that multiple concepts are present simultaneously. Hence, our proposed solution learns multiple global models, where each model is trained by a cluster of clients for each distinct concept. In this section, we present Algorithm 4 for multiple-model training in FL for a given input clustering, which may vary over time as drifts occur. Then in §5.4, we will show how to learn the necessary input clustering, and how new clusters can be created to adapt to newly appearing concepts.

We define a time step as the granularity at which new data may arrive at a client. A time step may consist of multiple communication rounds. The set of data arriving at client c and time t is denoted by $S_c^{(t)}$. The global models being trained are denoted by h_m for $m \in [M]$, where M is the total number of models at a given time. Each model is trained by a cluster of clients, where the clustering may vary over time as concept drifts occur. The cluster identities $w_{c,m}^{(t)}$ (§5.2) indicate whether the data $S_c^{(t)}$ that arrived at client c at time t are sampled when computing a local update to the global model h_m. Further, the cluster identity of a client at a given time indicates which model is used for inference.

Within each time, the training of the global models in Algorithm 4 is equivalent to Federated Averaging [66], since the aggregation weight of each client within each cluster is fixed at time τ. So the convergence of Algorithm 4 can be guaranteed by directly using previous analyses for Federated Averaging, such as [60, 86]. The difference here is that the objective function that clients are minimizing at time τ is replaced by the following:

$$\tilde{F}_m^{(\tau)}(h_m) = \sum_{c=1}^{P} \tilde{w}_{c,m}^{\tau} F_c^{(\tau)}(h_m) \tag{5.1}$$

where $F_c^{(\tau)}$ denotes the local objective function on client c, and the normalized weight is defined as $\tilde{w}_{c,m}^{\tau} = \sum_{t=1}^{\tau} w_{c,m}^{(t)} |S_c^{(t)}| / \sum_{c=1}^{P} \sum_{t=1}^{\tau} w_{c,m}^{(t)} |S_c^{(t)}|$.

Algorithm 4 Multiple-model training at time τ

```
Input: Cluster identities $w_{c,m}^{(t)}$
  for each round $i = 1, 2, \ldots, R$ do
    for each client $c = 1, 2, \ldots, P$
    and each model $m = 1, 2, \ldots, M$ in parallel do
      $h_{c,m} \leftarrow \textsc{LocalUpdate}(c, h_m, \{w_{c,m}^{(t)}\}_{t=1}^{\tau})$
    end for
    for each model $m = 1, 2, \ldots, M$ do
      $h_m \leftarrow \frac{\sum_{c=1}^{P} h_{c,m} \sum_{t=1}^{\tau} w_{c,m}^{(t)} N_c^{(t)}}{\sum_{c=1}^{P} \sum_{t=1}^{\tau} w_{c,m}^{(t)} N_c^{(t)}}$
    end for
  end for

  $\textsc{LocalUpdate}(c, h_m, \{w_{c,m}^{(t)}\}_{t=1}^{\tau})$:
  for each local step $j = 1, 2, \ldots, K$ do
    $b \leftarrow$ random minibatch of size $B$ from $\cup_{t: w_{c,m}^{(t)}=1} S_c^{(t)}$
    $h_m \leftarrow h_m - \eta \nabla \ell(h_m; b)$
  end for
  return $h_m$
```

Algorithm 5 Clustering to the lowest loss

```
  $\ell_{c,m}^{(\tau)} \leftarrow$ loss of $h_m$ on client data $S_c^{(\tau)}$
  $w_{c,m}^{(\tau)} \leftarrow \mathbf{1}\{m = \arg\min_{m'} \ell_{c,m'}^{(\tau)}\}$
  Run Algorithm 4
```

In the ideal case where each cluster maps to one concept in the system, each h_m is specialized for each concept that is sampled from a unique data distribution ($\mathcal{P}(x, y)$), and these h_m form a strong solution to our overall objective in §5.2. This ideal solution is the Oracle algorithm in our evaluation in §5.6, and we empirically demonstrate that our proposed solutions achieve comparable accuracy.

Note that, as stated, each client c in Algorithm 4 retains its complete history of both the cluster indicators $w_{c,m}^{(t)}$ and the local data arrivals $S_c^{(t)}$. To reduce this overhead, each client could instead maintain just a sliding window of the most recent time steps, as long as the window suffices for the minibatch sampling in LocalUpdate.

Thus, we have separated the problem of concept drift in FL into two components: (i) determining the time-varying clustering of clients in response to concept drifts, which is then used as input for (ii) the multiple-model training in Algorithm 4. Suppose, hypothetically, that there is a global model already initialized for each concept up to some moderate accuracy. In this restrictive setting, Algorithm 5 can be used to determine the cluster identities for each new time step. Each client tests the global models from the previous time step over its newly arrived data and chooses to identify with the model with the best loss (breaking ties randomly).[1] The setting considered encompasses time steps involving drifts that occur between concepts known to the system; e.g., the later stages of a staggered drift from concept A to concept B after some clients have already observed concept B (Figure 5.2). However, Algorithm 5 does not have any mechanism to spawn new clusters or determine the number of clusters. In §5.4, we will show how to determine the input for Algorithm 4 with clustering algorithms that can spawn clusters over time to react to drifts to *new* concepts.

5.4 Clustering Algorithms

Under concept drift in FL, data are heterogeneous both over time and across clients. The concept at each time and client is the ground-truth clustering that we seek to learn. Ideally, the models trained by each cluster correspond 1-to-1 to the concepts present in the system. Specifically, we want to avoid two miss-clustering problems: (**P1**) spawning *multiple clusters* that correspond to a *single concept*, because then each model would be trained over only a subset of the relevant data, not taking full advantage of collaborative training, and (**P2**) merging clients corresponding to *multiple concepts* into a *single cluster* (model poisoning).

We present two clustering algorithms for adapting to concept drift. First, in §5.4.1 we handle the case where only one new concept emerges at a time, which includes the example drift pattern in Figure 5.2, by incorporating a straightforward drift detection algorithm. Second, in §5.4.2 we give a general algorithm that handles the general case where multiple new concepts may emerge simultaneously, which includes the example drift pattern in Figure 5.3, by incorporating a bottom-up technique that *isolates clients* that detect drift (addressing **P2**) and *iteratively merges* clusters corresponding to the same concept (addressing **P1**).

In the rest of this section, we assume that the first time step starts with one concept and one model, and that our clustering is run for each time step $\tau > 1$ as new data arrive.

5.4.1 Special Case: One New Concept at a Time

When a new concept emerges, the clients that observe the drift should be split off to a new cluster to start training a new model. Drift detection has been well-studied in the centralized, non-FL, setting [3, 6, 28, 35, 70, 72]. For staggered drifts in FL, trying to apply a drift detection test *globally* at the server over the aggregate error results in poor performance during the transition period. Instead, in Algorithm 6, we apply drift detection *locally* at each client.

There are many drift detection tests in the literature, but the particular test is not our focus and for simplicity we consider a test of the following form. A drift is signaled at client c at time τ with respect to a model h_m if the loss of the model over the newly arrived data, denoted as $\ell_{c,m}^{(\tau)}$, degrades by a threshold δ relative to the loss measured at time $\tau - 1$:

$$\ell_{c,m}^{(\tau)} > \ell_{c,m}^{(\tau-1)} + \delta. \tag{5.2}$$

[1]If there are no new data at a particular client, then we say its cluster identity is carried over from the previous time step so the model used for inference is well-defined.

Algorithm 6 FedDrift-Eager at time τ

```
$\ell_{c,m}^{(\tau)} \leftarrow$ loss of model $h_m$ on client data $S_c^{(\tau)}$
$w_{c,m}^{(\tau)} \leftarrow \mathbf{1}\{m = \arg\min_{m'} \ell_{c,m'}^{(\tau)}\}$
**if** $\min_m \ell_{c,m}^{(\tau)} > \min_m \ell_{c,m}^{(\tau-1)} + \delta$ at any client $c$ **then**
    // create one model for all drifted clients
    $M \leftarrow M + 1$
    Initialize a new global model $h_M$
    $w_{c,*}^{(\tau)} \leftarrow \mathbf{0}$; $w_{c,M}^{(\tau)} \leftarrow 1$
**end if**
Run Algorithm 4
```

This test checks for any drift that incurs performance degradation with respect to a given model. However, the desired condition for creating a *new model* should check only for concept drifts that correspond to a concept *previously unobserved* and *ill-suited* for all existing models. For other drifts, such as the later stage of the staggered drift from concept A to concept B in Figure 5.2 (after concept B has already been detected and an appropriate model created), a client should join an existing cluster (in this case, the cluster for B). Hence, in Algorithm 6, the drift detection test for model creation compares against the *best performing* model:

$$\min_m \ell_{c,m}^{(\tau)} > \min_m \ell_{c,m}^{(\tau-1)} + \delta. \tag{5.3}$$

We note that detection tests that compare across multiple models have been previously studied in centralized learning in the context of adapting to recurring drifts [48]. The clustering in Algorithm 6 (FedDrift-Eager) applies this multiple-model drift detection test at each client, and creates a new cluster for all the clients that detect a new concept; otherwise, each client identifies with the cluster with the best-performing model. This algorithm relies on the assumption that only one new concept occurs at a time by assigning the drifted clients to a single cluster. Despite this limitation, Algorithm 6 still merits interest as it experimentally performs well on the non-trivial case of the staggered drift in Figure 5.2 that has not been addressed by the prior work, as shown in §5.6. However, for the drift in Figure 5.3 in which concepts B and C emerge simultaneously at different clients, this algorithm creates only one cluster and sub-optimally tries to train a single model for both new concepts (problem **P2** above). Next, we extend this algorithm to address the general case where an unknown number of new concepts can occur at a time.

5.4.2 General Case

When drifts to new concepts are detected at multiple clients, in general we do not know whether the drifts all correspond to one concept or multiple concepts (or even zero concepts in the event of false positives in detection). We designed Algorithm 7 (FedDrift) for clustering in the face of this uncertainty. For each client that detects drift to a new concept, Algorithm 7 conservatively isolates the clients to individual clusters, and then merges clusters corresponding to the same concept slowly and safely over time by iteratively applying classical hierarchical agglomerative clustering [46, 81].

The generic hierarchical clustering procedure is specified by a distance function over the set of elements to be clustered and a stopping criterion, and at each step until the stopping criterion is met, merges the two closest clusters, where the distance between clusters of multiple elements is commonly defined to be the maximum distance between their constituents (known as a max-linkage clustering). In Algorithm 7, the MERGE subroutine combines two clusters i and j by averaging their models with weight proportional to the size of each model's training dataset (over all clients) and unifying the cluster identities.

To specify a distance function for hierarchical clustering, Algorithm 7 first aggregates at the server the loss estimates L_{ij} of the model h_i evaluated over a subsample of the data associated with the cluster for model h_j.[2] Then the distances between each cluster are initialized as $D(i, j) \leftarrow \max(L_{ij} - L_{ii}, L_{ji} - L_{jj}, 0)$.[3] $L_{ij} - L_{ii}$ measures the loss degradation of model h_i when evaluated over the data associated with h_j, relative to the loss over its own data. We informally interpret this difference as the magnitude of drift between the concept associated with h_i to the concept associated with h_j, analogous to the drift detection condition in Eq (5.2) (although not identical due to the bias of L_{ii} measuring a model's accuracy over its own training data). The term $D(i, j)$ is defined to be symmetric by also accounting for the magnitude of the drift $L_{ji} - L_{jj}$ in the reverse direction from concept j to concept i.

In addition to defining the cluster distances $D(i, j)$, employing hierarchical clustering also requires setting a stopping criterion. Typically, that corresponds to specifying either the desired number of clusters (which in our case is unknown), or an upper limit on the distance between clusters to stop merging. By our identification of the cluster distance as a magnitude of drift, we naturally re-use the drift detection threshold δ to also represent the tolerance level up to which clusters can be merged, eliminating one hyperparameter.

In Algorithm 7, both creating new clusters and merging existing clusters are based on the observed difference of the models' accuracy across two samples of data. For the clustering to accurately distinguish concepts, we assume that relevant changes in the concepts are manifested in the degradation of a model's predictive accuracy, and that the local sample size is sufficient for statistical significance—the same assumptions necessary for prior drift detection tests [35, 70, 72].

One subtlety to Algorithm 7 is that the hierarchical clustering is iteratively run at every time step, because the cluster distances vary with time. A simpler alternative would be to only try merging newly created clusters of local models after one time step of training. However, at that one time step, even models corresponding to the same concept may fail to merge given the limited sample size and limited number of training iterations. In other words, while the models are still warming-up, they may still be separated by a distance exceeding δ. As the models converge over time, the distance may drop below δ, which Algorithm 7 accounts for by iteratively attempting to merge.

The hierarchical clustering strategy of Algorithm 7 allows it to adaptively determine the appropriate number of clusters even when an unknown number of new concepts emerge at a time, but it also incurs additional computational resources relative to Algorithm 6. Algorithm 7

[2]More precisely, at client c, the data clustered to h_j are subsampled proportionate to the size of the local dataset relative to the global dataset for h_j, $\sum_t w_{c,j}^{(t)} N_c^{(t)} / \sum_{c'} \sum_t w_{c',j}^{(t)} N_{c'}^{(t)}$.

[3]We note that $D(i, j)$ is not necessarily a true distance function as there is no guarantee that it satisfies the triangle inequality.

Algorithm 7 FedDrift at time τ

```
$\ell_{c,m}^{(\tau)} \leftarrow$ loss of model $h_m$ on client data $S_c^{(\tau)}$
**for** each client $c = 1, 2, \dots, P$ in parallel **do**
    **if** $\min_m \ell_{c,m}^{(\tau)} > \min_m \ell_{c,m}^{(\tau-1)} + \delta$ **then**
        Initialize a local model at client $c$ to be added to the set of global models at $\tau + 1$, and
        assign client $c$ to its own cluster
    **else**
        $w_{c,m}^{(\tau)} \leftarrow \mathbf{1}\{m = \arg\min_{m'} \ell_{c,m'}^{(\tau)}\}$
    **end if**
**end for**
**for** each $i, j$ from $1, 2, \dots, M$ in parallel **do**
    $L_{ij} \leftarrow$ loss of model $h_i$ on sample of $\cup_{c,t:w_{c,j}^{(t)}=1} S_c^{(t)}$
**end for**
Cluster distances $D(i,j) \leftarrow \max(L_{ij} - L_{ii}, L_{ji} - L_{jj}, 0)$
**while** $\min_{i \neq j} D(i,j) < \delta$ **do**
    MERGE$(i, j, D)$
**end while**
Run Algorithm 4

MERGE$(i, j, D)$:
Add a new model $h_k \leftarrow \frac{h_i \sum_{c,t} w_{c,i}^{(t)} N_c^{(t)} + h_j \sum_{c,t} w_{c,j}^{(t)} N_c^{(t)}}{\sum_{c,t} w_{c,i}^{(t)} N_c^{(t)} + \sum_{c,t} w_{c,j}^{(t)} N_c^{(t)}}$
$w_{c,k}^{(t)} \leftarrow w_{c,i}^{(t)} + w_{c,j}^{(t)}$ for all $c, t$
$D(k,l) = \max(D(i,l), D(j,l))$ for all $l$
Delete models $h_i, h_j$
```

creates more global models M, adding to the communication cost of sending $O(MP)$ models. Additionally, the hierarchical clustering adds an $O(M^2 \log M)$ time complexity at the server at every time step (using a heap data structure for finding the minimum pairwise distance). In §5.7, we discuss how we can restrict Algorithm 7 to create fewer overall models for higher efficiency. Also, similar to Algorithm 4, each client c could maintain $w_{c,m}^{(t)}$ and $S_c^{(t)}$ for just a sliding window of the most recent time steps, as long as the window suffices for Algorithm 7's subsampling step.

5.5 Experimental Setup

Prior work on FL under drifts is limited to simple cases such as in Figure 5.1, as noted in §5. Our evaluation covers the synthetic drifts in Figures 5.2 and 5.3, which represent more complex scenarios where drifts (i) occur across clients with staggered timing, (ii) correspond to different concept changes across different clients, and (iii) involve recurring concepts (e.g., the sequence A–B–C–D–A). We also evaluate on the real-world drift in the FMoW dataset, which shows gradual concept changes staggered across clients (shown in Figure 5.4, repeated from §1).

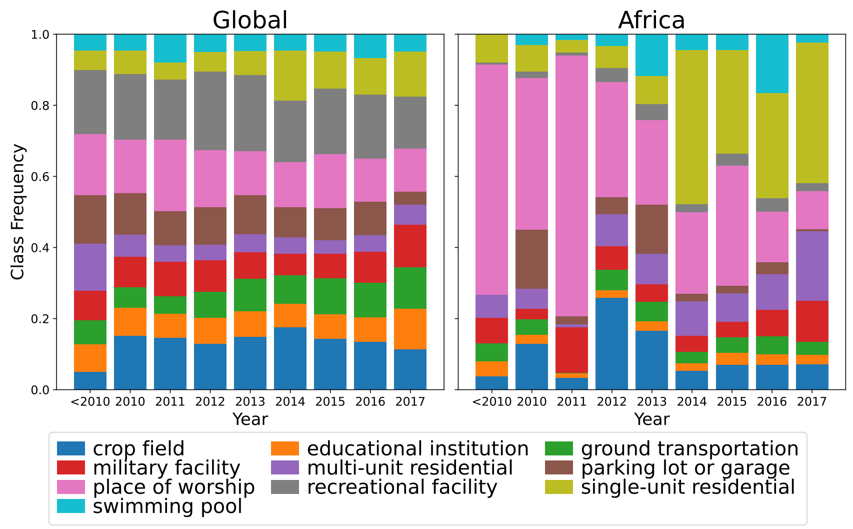


Figure 5.4: Class distribution over time in FMoW. The drift viewed globally (left) is small relative to the localized drift for Africa (right).

The synthetic drift patterns are studied with respect to the following datasets: SINE [71], CIRCLE [71], SEA [8], and MNIST [57]. SINE and CIRCLE each have 2 defined concepts, and we generate partitions of the data under the 2-concept staggered drift of Figure 5.2, while SEA and MNIST have more defined concepts, and we generate partitions under both the 2-concept and 4-concept drift patterns of Figures 5.2 and 5.3 for 10 clients and 10 time steps. For the real drift in FMoW, we evaluate on a subset of the data including the 10 most common classes, and identify each of the 5 major regions as one client and each new year as one time step. §5.5.1 has further dataset details.

We compare our algorithms FedDrift-Eager and FedDrift against the following baselines. First, the Oblivious algorithm learns a single model with FedAvg and has no mechanism for drift adaptation. Second, we consider traditional (non-FL) drift adaptation algorithms applied centrally at the server on top of FedAvg. Drift adaptation is typically classified into three categories, and we compare against algorithms representative of each: the drift detection method DriftSurf (§4), two ensemble methods KUE [14] and AUE [13], and a Window method that forgets data older than one time step (more are reported in §5.7). Third, Adaptive-FedAvg [15] is an FL algorithm that learns a single model and adapts to drifts by centrally tuning the learning rate used by all clients as a function of the variability across updates. Fourth, we compare to static FL clustering algorithms IFCA [33] and CFL [77], which we extend to the time-varying setting by adding a window method (more variations reported in §5.7). Fifth, Oracle is an idealized algorithm that has oracle access to the concept ID at training time and runs the multiple-model training of Algorithm 4 with the ground-truth clustering.

We run our experiments using the FedML framework [38]. At each time step, each client observes a new batch of training data. For all the experiments on synthetic datasets, the models trained under each algorithm are fully connected neural networks with a single hidden layer of size $2d$ where d is the number of features. On the FMoW dataset, each algorithm trains ResNet18 models pretrained on ImageNet [39]. After training for each time step, we test each algorithm over the batch of data arriving at the following time step, for all time steps. Each experiment is run for 5 trials, and we report the mean and the standard deviation. Additional algorithm details are in §5.5.2.

5.5.1 Datasets

We consider synthetic distributed drifts with respect to the following datasets previously used in the concept drift and personalized FL literature [12, 13, 15, 62]: SINE and CIRCLE [71] which each have 2 defined concepts, and SEA [8] and MNIST [57], which have up to 4 concepts. In SINE, the first concept is a decision boundary of the sine curve $x_2 < \sin(x_1)$ for data points sampled from the unit square, and the second concept reverses the direction (swapping the labels). In CIRCLE, the two concepts are each decision boundaries of two different circles in the unit square, representing a smaller concept change than SINE. The first circle is centered at (0.2, 0.5) with radius 0.15 and the second circle is centered at (0.6, 0.5) with radius 0.25. In SEA, each concept corresponds to a shifted hyperplane. Each point in SEA has three attributes in [0, 10], where the label is determined by $x_1 + x_2 \leq \theta_j$ where j corresponds to 4 concepts, $\theta_A = 9, \theta_B = 8, \theta_C = 7, \theta_D = 9.5$. (The third attribute x_3 is not correlated with the label.) In SEA, at every concept there is noise in the observed labels, where the label is swapped with 10% chance for each data point independently. In MNIST, concept A corresponds to the original labeling of the hand-drawn digits, and under each other concept, the labels of two of the digits are swapped (B swaps digits 1 and 2, C swaps digits 3 and 4, and D swaps digits 5 and 6).

For each of the synthetic drift datasets in our experiments, the training data are distributed across 10 clients and arrive over 10 time steps. The partition of the data at each client and time is a constant 500 number of samples from the concept corresponding to the concept drift patterns in Figures 5.2 and 5.3 in §5. In our experimental results, after training at each time τ we report the test accuracy over the data at $\tau + 1$. For clarification, in reporting the accuracy at the last time step 10, we test over an 11th sample of data at each client that is from the same concept observed during training at time 10.

We also evaluate on the real-world drift in the Functional Map of the World (FMoW) dataset included in the WILDS benchmark [20, 52]. The learning task is to classify the land use or building type from satellite images, which has significant practical relevance, "aiding policy and humanitarian efforts in applications such as deforestation tracking, population density mapping, crop yield prediction, and other economic tracking applications" [52]. Each image is RGB and square with a width of 224 pixels. The WILDS benchmark is not explicitly posed as a drift *adaptation* problem that we study in this thesis, but instead as a drift *robustness* problem, and so they originally partitioned the data into train/validation/test splits. For our evaluation, we re-partition the dataset, distributing training data across 5 clients arriving over 9 time steps, using the metadata annotation of each image by region (Africas, Americas, Asia, Europe, Oceania) and year. The first 8 years from 2002–2009 have much fewer images collected, which we group into one time step, and then we treat each year from 2010–2017 as one time step each. The partition of the data at each client and time step is a subsample of up to 1000 images at the 10 classes that are the most common (counting across all regions and years). The test data evaluated for the last time step are a disjoint subsample also from the same year 2017 as the training data. Figure 5.4 in §5.6 depicts how the data drifts gradually over time, where the development of new infrastructure is a result of social, political, economic, and environmental factors. Viewed globally, the drift is small. Koh et al. [52] write: “intriguingly, a large subpopulation shift across regions only occurs with a combination of time and region shift.” Further, they call for solutions that “can leverage the structure across both space and time” and also hypothesize a benefit to “potentially transfer knowledge of other regions with similar economies and infrastructure” which we empirically confirm where FedDrift clusters Africa and Oceania together for years 2014–2015.

5.5.2 Experimental Parameters

Across all algorithms we evaluate, the algorithms that learn a single model use FedAvg for training, and the clustering algorithms that learn multiple models use Algorithm 4 in §5.3 for training (which reduces to FedAvg when there is one cluster). The training parameters used in our experiments are shown in Table 5.2. For efficiency of the larger FMoW experiments, we reduce to 10 rounds and batch size 32—we observe that this suffices by convergence of the training accuracy.

Regarding the learning rate selection, first we discuss all algorithms excluding Adaptive-FedAvg. We searched for learning rates of the form 10^{-a} for $a = 1, 2, 3, 4$, for each dataset, and found that $\eta = 10^{-2}$ was the best for SINE-2, CIRCLE-2, SEA-2, and SEA-4, that $\eta = 10^{-3}$ was best for MNIST-2 and MNIST-4, and that $\eta = 10^{-4}$ was best for FMoW. (This held for both of the two extremes among our baselines, Oblivious and Oracle, and we apply the same learning rate across all the algorithms. For FMoW, there is no known Oracle, so we searched only using the Oblivious baseline.) Also note that for computing the LOCALUPDATE at each client, we use the implementation of Adam in PyTorch with the options weight decay = 10^{-3} and amsgrad = True. We treat Adaptive-FedAvg separately, because it uses SGD with its own internal learning rate scheduler as its mechanism to react to drifts. We found that the initial learning rate of 10^{-2} was the best for each dataset with the exception of SINE-2, instead using 10^{-1}. (This higher learning rate explains the high standard deviation in the reported accuracy of Adaptive-FedAvg on SINE-2.)

Next, we report the selection of the drift detection threshold δ in the algorithms DriftSurf, FedDrift-Eager, and FedDrift. While the optimal δ is expected to vary across datasets, even for a fixed dataset, different algorithms can peak in performance at varying δ. The performance of each of these three algorithms for each dataset across δ in the range $0.02, 0.04, \ldots, 0.20$ is shown in Figure 5.5. To not bias towards any one algorithm, the experimental results are reported for each algorithm and dataset using its best δ. (The δ used for the FedDrift-C variant discussed in §5.7

Table 5.2: Training parameters

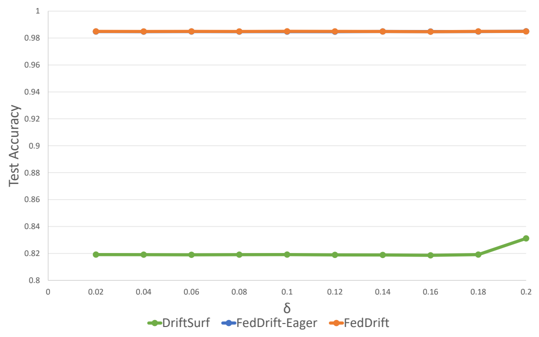
Parameter	Description	Experimental setting (all synthetic drifts)	Experimental setting (FMoW)
R	# communication rounds	100	10
K	# local steps per model per round	50	50
B	minibatch size	50	32
η	step size	varies	varies

is identical to that used for FedDrift.) However, using a fixed $\delta = 0.04$ for FedDrift-Eager and FedDrift makes at most a 1 pp difference in the results reported in Table 5.3 (on one trial).

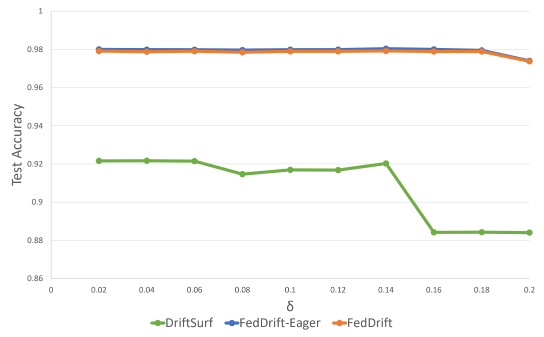
For all other hyperparameters of the algorithms we evaluate, we follow the parameter choices stated by the authors, with the following exceptions: for DriftSurf we use $r = 3$ (which performed equal or better than the $r = 4$ used in experiments in §4 for a longer time horizon of 100 time steps); for CFL we use $\gamma = 0.1$ (for which there is no default, but is shown to be a good setting from Theorem 1 and Figure 3 of their paper [77] given that the number of distinct concepts at a time is at most 5 across all evaluated datasets); and for AUE we use $K = 5$ as the total ensemble size (compared to the $K = 10$ in their paper they consider over a significantly longer time horizon). In reporting FMoW results, for training efficiency, we further restrict to a total ensemble size of 4 for AUE and KUE.

Furthermore, for the FMoW dataset, which has more than one distinct data distribution at the initial time step unlike the remaining datasets, we use a different initialization of IFCA variants and FedDrift. For IFCA variants, clients initially self-select among 5 cluster centers instead of being all assigned to a single cluster. For FedDrift, clients are initialized to a local model each, which can be merged starting at the next time step. (If we instead initialize all clients to a single model that can later be split, we observed the average test accuracy of FedDrift is 64.46%, or 0.45 pp worse.)

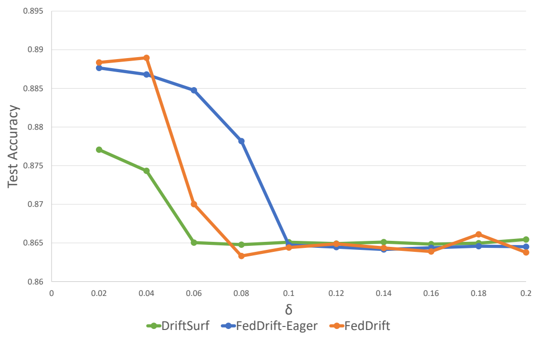
Finally, regarding the model training in Algorithm 4 at time τ, we apply one optimization for efficiency to only train models that are currently clustered to. (Although note that any such models are still retained by FedDrift-Eager and FedDrift in order to react to recurring drifts even if they are not actively being trained.)



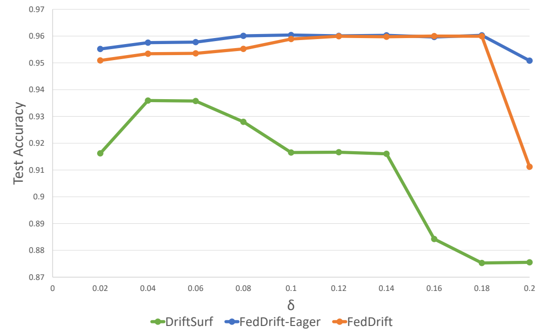
(a) SINE-2



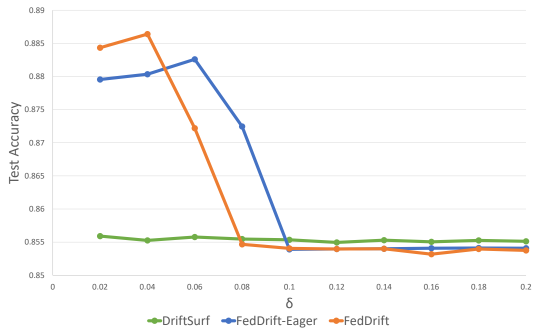
(b) CIRCLE-2



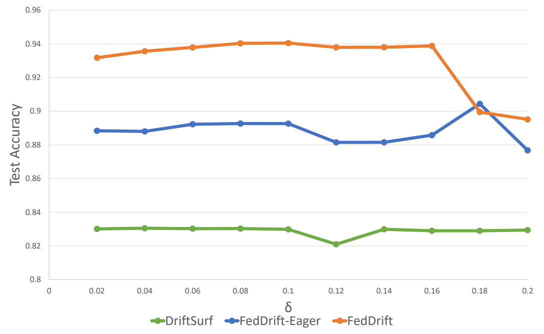
(c) SEA-2



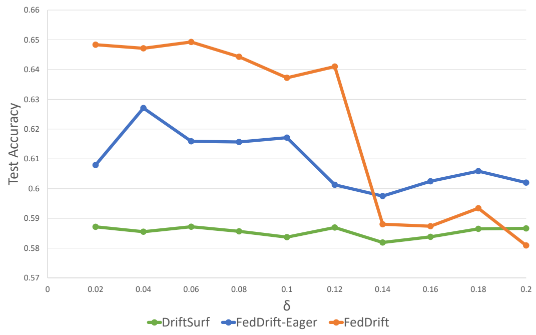
(d) MNIST-2



(e) SEA-4



(f) MNIST-4



(g) FMoW

Figure 5.5: Average accuracy of each drift detection-based algorithm under varying thresholds δ.

5.6 Main Experimental Results

We empirically demonstrate that FedDrift-Eager and FedDrift are more effective than prior centralized drift adaptation and achieve high accuracy that is comparable to an oracle algorithm in the presence of distributed concept drifts.

In Table 5.3, we report the test accuracy averaged across all clients and all time steps except for the times of drifts (for synthetic datasets). We omit the times of drift because there is no chance for a client to adapt to the drift yet, and we eliminate the noise from beneficial clustering mistakes if by chance a client were clustered to the model appropriate for the test data after the drift. (For completeness, §5.7 shows results averaged over all time steps including drifts.)

Across all the 2-concept datasets under the staggered drift, we observe that the multiple-model algorithms FedDrift-Eager and FedDrift outperform the prior centralized solutions. In Figure 5.6, the accuracy is broken down per time step on CIRCLE-2, where we observe that centralized algorithms particularly suffer during the transition period. The fundamental issue is that when both concepts simultaneously exist, no single model can accurately fit for all clients. Even the ensemble algorithm (KUE) has poor performance because any new model added is updated by each client, and during the transition period, there is no model trained solely over data from the second concept. FedDrift-Eager and FedDrift learn models specialized for the second concept immediately after it emerges, and learn to apply the appropriate model at each client during the transition, matching the performance of Oracle.[4]

Another challenge that the 2-concept staggered drift poses for DriftSurf, KUE, AUE, and Adaptive-FedAvg is that their adaptation strategies are a function of estimators that, from the central server's perspective, are aggregated over some clients that are drifting and others that are not. It is muddy whether drift is truly occurring, and even the unsophisticated window-based algorithm performs slightly better.

The clustering algorithms IFCA and CFL with a window perform relatively well on the 2-concept staggered drifts because they can flexibly employ a model specialized for the second concept during the transition period, but are overall behind FedDrift and FedDrift-Eager. We observe IFCA's success in adapting to drift is dependent on its random parameter initialization for its clusters, which works well for the sharp drift on SINE-2, but less reliably for the smaller drifts on other datasets. For CFL, we observe that its iterative cluster splitting reacts quickly to drift, but creates excessive models for a concept over time without unifying clients under staggered drift. §5.7 has more details.

Regarding the 4-concept drift, Table 5.3 shows that all baselines are ill-suited, while FedDrift performs close to Oracle, and that FedDrift-Eager has intermediate performance (due to its false unification of simultaneously emerging concepts). To understand the performance of FedDrift, see Figure 5.7. In the ideal case (Oracle), there would be exactly one model for each concept. For FedDrift, at time 3 one new model is created for 5 of the 6 clients that drifted, and one false negative where a drifted client stays on the original model. With hierarchical clustering applied at the beginning of time 4, the 3 clusters corresponding to the green concept are correctly merged, while all clients on the yellow concept cluster to model 4 which had the lowest test loss over

[4]The accuracy of FedDrift-Eager and FedDrift are higher than Oracle in a few cases but within the standard deviation, which we attribute to randomness in the model initialization and training.

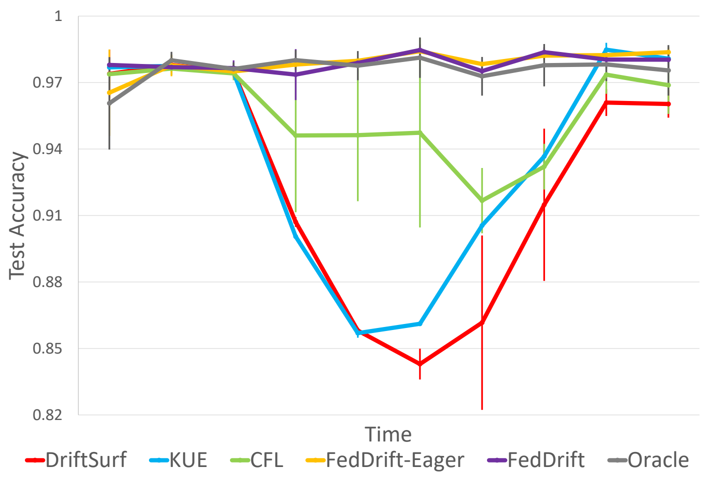


Figure 5.6: Accuracy at each time (averaged across clients) on CIRCLE-2.

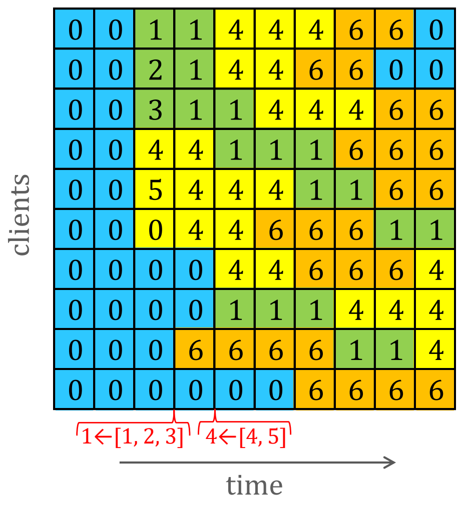


Figure 5.7: The clustering learned by FedDrift on MNIST-4. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept.

Table 5.3: Average test accuracy (%) across all clients and time, omitting drifts (5 trials)

	SINE-2	CIRCLE-2	SEA-2	MNIST-2	SEA-4	MNIST-4	FMoW
Oblivious	50.44 ± 1.52	88.36 ± 0.27	86.37 ± 0.34	87.25 ± 0.14	85.38 ± 0.28	82.97 ± 0.04	58.46 ± 0.08
DriftSurf	83.90 ± 1.01	92.54 ± 0.67	87.27 ± 0.34	91.71 ± 1.60	85.48 ± 0.28	82.99 ± 0.05	58.42 ± 0.16
KUE	87.05 ± 0.12	93.83 ± 0.04	87.62 ± 0.42	89.74 ± 0.07	85.53 ± 0.12	79.78 ± 0.16	36.65 ± 7.64
AUE	86.06 ± 0.60	92.74 ± 0.51	87.46 ± 0.12	92.19 ± 0.07	85.55 ± 0.08	81.29 ± 0.19	54.22 ± 0.14
Window	86.42 ± 0.74	93.67 ± 0.15	88.08 ± 0.10	92.15 ± 0.34	85.76 ± 0.16	81.16 ± 0.46	58.79 ± 0.14
Adaptive-FedAvg	78.02 ± 10.73	86.26 ± 0.00	86.69 ± 0.39	92.16 ± 0.04	85.32 ± 0.25	81.62 ± 0.07	52.76 ± 0.23
IFCA+Window	**98.49 ± 0.13**	94.31 ± 1.62	88.04 ± 0.17	91.76 ± 0.50	86.17 ± 1.00	81.27 ± 0.43	49.40 ± 0.76
CFL+Window	95.15 ± 0.32	95.62 ± 1.14	87.66 ± 0.36	90.53 ± 0.81	85.67 ± 0.21	79.99 ± 0.58	58.70 ± 0.13
FedDrift-Eager	98.46 ± 0.03	97.86 ± 0.20	88.35 ± 0.37	**95.99 ± 0.06**	88.08 ± 0.24	89.21 ± 2.02	61.62 ± 0.45
FedDrift	98.48 ± 0.01	**97.88 ± 0.17**	**88.65 ± 0.43**	95.93 ± 0.01	**88.41 ± 0.29**	**94.09 ± 0.08**	**64.91 ± 0.31**
Oracle	98.46 ± 0.01	97.57 ± 0.59	88.53 ± 0.23	96.00 ± 0.02	88.75 ± 0.20	94.60 ± 0.04	-

the new data. Also at time 4, model 6 is created for the new orange concept. Then at time 5, hierarchical clustering merges models 4 and 5 (due its iterative application in FedDrift, as the distance decreases after model 4 is further trained). After time 5, FedDrift has a distinct model for each concept, and no excess models.

One drawback of FedDrift is that it can create more models compared to FedDrift-Eager, adding to the communication cost. §5.7 shows that restricting FedDrift to just one new global model per time step (additional local models are still permitted) decreases its accuracy by only 0.87% on the MNIST-4 dataset, while saving communication.

Finally, we discuss the drift in the real-world FMoW dataset where we observe FedDrift has superior performance. The authors of the WILDS benchmark primarily make note of the performance loss of a globally trained model on data from Africa over time [52]. We observe FedDrift successfully adapts to the local drift, switching the model applied at Africa at year 2014, the time with a significant increase in single-unit residential buildings in Figure 5.4. Instead of creating a new model for 2014, we find FedDrift joins the cluster for Oceania where a local model was previously created, and stays at that cluster for 2014 and 2015, before then splitting into a new individual cluster for 2016 and 2017. We also observe that FedDrift detects a drift at 2015 for both Europe and the Americas, creating two more local models that contribute to higher accuracy.

Meanwhile, FedDrift-Eager similarly adapts to the change in Africa yielding a performance benefit, but it does not adapt well to the simultaneous drift for Europe and the Americas. Both FedDrift and FedDrift-Eager outperform the centralized adaptation baselines which fail to adapt to the drift when viewed globally (c.f. Figure 5.4). Finally, the low accuracy of IFCA is explained by its random initialization of model parameters for its clusters, in lieu of the pretrained ImageNet initialization under the rest of the algorithms, and the low accuracy of KUE is explained by its ineffective random subspace projections of the data for this task.

5.7 Additional Experimental Results

We present additional experimental results on more baseline algorithms and on variants of our algorithms restricted to limited memory or communication.

Additional Baseline Algorithms. The additional algorithms presented in this section are:

- **Four traditional drift adaptation algorithms.** AUE-PC is a variation of the ensemble method AUE with the ensemble weights set *per-client*. Window-2 is a window method like Window, except that it forgets data older than two time steps instead of one. Weighted-Linear and Weighted-Exp also forget older data like window methods, but do so more gradually by down-weighting older data with either linear or exponential decay.
- **The FL clustering algorithm CFL [77].** In extending the original static algorithm to our time-varying setting, we also consider a variant CFL-W, in which during training, each client samples only from the window of the newest data arriving at each time.
- **Three variations of the IFCA clustering algorithm [33]** that we considered for extending the original algorithm to the time-varying setting. First, IFCA(T) is exactly Algorithm 5 in §5.3, which defines cluster identities for each client and each time, in order to associate the data within a client that are heterogeneous over time across multiple clusters. IFCA(T) chooses the cluster identity once per *time step* (where time steps consist of multiple communication rounds)—this differs from the original algorithm described by Ghosh et al. [33], which recomputes the cluster identity once per *round*. Second, IFCA does the per-round clustering; more precisely, for each time step τ, the cluster identity $w_{c,m}^{(\tau)}$ is recomputed at every round under the same equation used at the beginning of the time step in Algorithm 5. Third, IFCA-W is a variant of IFCA that trains only over the most recent data arrivals at each time, and the cluster identities of data from previous time steps are forgotten. In general, the IFCA-based algorithms require the number of clusters as input, which we provide as oracle knowledge—either 2 or 4 depending on the total number of concepts over time in each dataset. This gives IFCA-based algorithms an advantage over all other algorithms we evaluate, which do not know the number of clusters a priori. For the initialization of all three variations, at time 1 and round 1, all clients are assigned to a single cluster, matching the assumption we made for FedDrift and FedDrift-Eager in §5.4. The exception to this initialization strategy is on FMoW, where the total number of concepts is not known, and the concept at time 1 across clients is not identical; for this dataset, we instead initialize all IFCA-based algorithms with a total of 5 clusters (matching the number of regions), and where each client identifies with the best-performing randomly initialized model (same as the original paper).
- **A more communication-efficient variant of FedDrift.** FedDrift-C is the algorithm referred to in the last paragraph of §5.4 that is restricted to introducing one new global model per time step. More details on this algorithm are described later in this section.
- **Sliding window variants of FedDrift-Eager and FedDrift.** FedDrift-Eager-W and FedDrift-W are restricted to using only the most recent time step of data $S_c^{(t)}$ and cluster identities $w_{c,m}^{(t)}$.
- **A baseline sliding window variant** Oracle-W, which has oracle access to the ground-truth clustering but only uses the most recent time step of data in training.

In general, we use the -W suffix in the name of an algorithm to indicate a limited memory of a window of one time step. This memory restriction reduces the number of samples used for training at a time and might reduce the accuracy achievable under ground-truth clustering

Table 5.4: Average test accuracy (%) across clients and time, omitting drifts (5 trials), extended results

	SINE-2	CIRCLE-2	SEA-2	MNIST-2	SEA-4	MNIST-4	FMoW
Oblivious	50.44 ± 1.52	88.36 ± 0.27	86.37 ± 0.34	87.25 ± 0.14	85.38 ± 0.28	82.97 ± 0.04	58.46 ± 0.08
DriftSurf	83.90 ± 1.01	92.54 ± 0.67	87.27 ± 0.34	91.71 ± 1.60	85.48 ± 0.28	82.99 ± 0.05	58.42 ± 0.16
KUE	87.05 ± 0.12	93.83 ± 0.04	87.62 ± 0.42	89.74 ± 0.07	85.53 ± 0.12	79.78 ± 0.16	37.46 ± 7.95
AUE	86.06 ± 0.60	92.74 ± 0.51	87.46 ± 0.12	92.19 ± 0.07	85.55 ± 0.08	81.29 ± 0.19	54.22 ± 0.14
AUE-PC	87.67 ± 1.70	93.05 ± 0.19	87.61 ± 0.08	92.22 ± 0.09	85.60 ± 0.05	81.43 ± 0.22	54.15 ± 0.10
Window	86.42 ± 0.74	93.67 ± 0.15	88.08 ± 0.10	92.15 ± 0.34	85.76 ± 0.16	81.16 ± 0.46	58.79 ± 0.14
Window-2	85.21 ± 1.67	93.03 ± 0.46	87.71 ± 0.33	92.54 ± 0.37	85.67 ± 0.16	82.16 ± 0.32	59.44 ± 0.23
Weighted-Linear	72.78 ± 1.23	89.91 ± 0.65	87.00 ± 0.01	89.70 ± 0.12	85.49 ± 0.17	82.79 ± 0.05	58.05 ± 0.17
Weighted-Exp	82.77 ± 0.64	92.69 ± 0.25	87.59 ± 0.15	92.19 ± 0.17	85.59 ± 0.09	82.55 ± 0.06	58.49 ± 0.09
Adaptive-FedAvg	78.02 ± 10.73	86.26 ± 0.00	86.69 ± 0.39	92.16 ± 0.04	85.32 ± 0.25	81.62 ± 0.07	52.76 ± 0.23
CFL	60.27 ± 4.82	88.39 ± 0.40	86.36 ± 0.28	86.97 ± 0.40	85.33 ± 0.26	81.95 ± 0.55	57.92 ± 0.32
CFL-W	95.15 ± 0.32	95.62 ± 1.14	87.66 ± 0.36	90.53 ± 0.81	85.67 ± 0.21	79.99 ± 0.58	58.70 ± 0.13
IFCA(T)	98.45 ± 0.03	91.72 ± 5.19	86.46 ± 0.23	87.33 ± 0.15	85.44 ± 0.14	82.90 ± 0.05	47.76 ± 1.98
IFCA	98.46 ± 0.02	92.20 ± 5.32	86.45 ± 0.25	87.55 ± 0.25	85.35 ± 0.09	82.89 ± 0.04	48.17 ± 1.30
IFCA-W	98.49 ± 0.13	94.31 ± 1.62	88.04 ± 0.17	91.76 ± 0.50	86.17 ± 1.00	81.27 ± 0.43	49.40 ± 0.76
FedDrift-Eager	98.46 ± 0.03	97.86 ± 0.20	88.35 ± 0.37	**95.99 ± 0.06**	88.08 ± 0.24	89.21 ± 2.02	61.62 ± 0.45
FedDrift	98.48 ± 0.01	**97.88 ± 0.17**	**88.65 ± 0.43**	95.93 ± 0.01	**88.41 ± 0.29**	**94.09 ± 0.08**	**64.91 ± 0.31**
FedDrift-C	98.51 ± 0.11	97.42 ± 0.57	88.30 ± 0.53	95.85 ± 0.05	87.46 ± 0.42	93.22 ± 0.44	61.86 ± 0.30
FedDrift-Eager-W	98.51 ± 0.12	97.34 ± 0.76	88.43 ± 0.23	94.05 ± 0.02	87.90 ± 0.25	89.31 ± 0.38	61.94 ± 0.38
FedDrift-W	**98.58 ± 0.17**	97.68 ± 0.09	88.43 ± 0.22	93.95 ± 0.02	88.17 ± 0.39	91.47 ± 0.07	64.22 ± 0.60
Oracle	98.46 ± 0.01	97.57 ± 0.59	88.53 ± 0.23	96.00 ± 0.02	88.75 ± 0.20	94.60 ± 0.04	-
Oracle-W	98.47 ± 0.03	97.84 ± 0.11	88.70 ± 0.17	94.04 ± 0.02	88.74 ± 0.13	91.89 ± 0.05	-

(Oracle-W vs. Oracle). Yet, the window is not strictly a drawback: (i) forgetting the older data builds in a passive adaptation to drift and (ii) in our setting it also guarantees that each client's training data at a step are all drawn from the same distribution—this is why we also investigate -W variants when extending the prior static clustering algorithms CFL and IFCA to our setting when data arrive over time.

Test Accuracy Results. Table 5.4 (extending Table 5.3 in §5.6) shows the test accuracy of all algorithms, averaged across all clients and time steps, but omitting the times of drifts. As noted in §5.6, we omit the times of drift when all algorithms suffer from the performance loss. For completeness, the test accuracy averaged over all time steps including drifts is shown in Table 5.5. In this latter table, note that Oracle and Oracle-W suffer a performance loss too at the time of drift. Under the test-then-train evaluation, Oracle has access to the concept ID of the data at training time but not at test time, where at each client, the model used for inference corresponds to the observed concept in the most recently arrived training data. Note that for the real-world gradual drifts in FMoW, the ground-truth is unknown, so we omit results for Oracle. Furthermore, because drifts occur gradually and there is no oracle knowledge of their timing, we report identical test accuracy results on FMoW in Tables 5.4 and 5.5, averaging across all clients and time steps.

Based on these tables, we make the following observations on the additional algorithms. The AUE-PC variant of AUE extends the model weights in the ensemble method to be individualized per-client, based on the performance of each model over each client's local data (as opposed to weights chosen based on the aggregate performance at the server). This additional flexibility leads to only a marginal accuracy improvement over AUE across all datasets. While it is generally valuable for clients at different stages of a staggered drift to use different models for inference, the

Table 5.5: Average test accuracy (%) across clients and time, including drifts (5 trials), extended results

	SINE-2	CIRCLE-2	SEA-2	MNIST-2	SEA-4	MNIST-4	FMoW
Oblivious	45.77 ± 1.52	87.12 ± 0.26	86.12 ± 0.35	86.28 ± 0.12	85.11 ± 0.24	81.60 ± 0.03	58.46 ± 0.08
DriftSurf	79.19 ± 0.88	91.16 ± 0.68	87.00 ± 0.35	90.55 ± 1.68	85.13 ± 0.19	81.62 ± 0.04	58.42 ± 0.16
KUE	87.05 ± 0.12	93.83 ± 0.04	87.62 ± 0.42	89.74 ± 0.07	85.53 ± 0.12	79.78 ± 0.16	37.46 ± 7.95
AUE	81.28 ± 0.81	91.50 ± 0.46	87.21 ± 0.11	91.07 ± 0.07	85.15 ± 0.07	79.65 ± 0.25	54.22 ± 0.14
AUE-PC	82.18 ± 2.01	91.75 ± 0.17	87.34 ± 0.08	91.07 ± 0.09	85.16 ± 0.04	79.70 ± 0.24	54.15 ± 0.10
Window	81.92 ± 0.88	92.40 ± 0.11	87.86 ± 0.08	91.35 ± 0.43	85.33 ± 0.10	78.88 ± 0.62	58.79 ± 0.14
Window-2	80.35 ± 2.02	91.73 ± 0.49	87.45 ± 0.34	91.47 ± 0.47	85.24 ± 0.15	80.06 ± 0.61	59.44 ± 0.23
Weighted-Linear	67.20 ± 1.43	88.67 ± 0.64	86.77 ± 0.02	88.56 ± 0.12	85.16 ± 0.11	81.38 ± 0.04	58.05 ± 0.17
Weighted-Exp	76.80 ± 0.88	91.30 ± 0.26	87.34 ± 0.16	91.05 ± 0.18	85.19 ± 0.06	80.96 ± 0.07	58.49 ± 0.09
Adaptive-FedAvg	73.82 ± 10.75	85.60 ± 0.00	86.55 ± 0.35	91.31 ± 0.05	85.01 ± 0.21	79.45 ± 0.06	52.76 ± 0.23
CFL	54.41 ± 4.33	87.08 ± 0.31	86.10 ± 0.30	86.00 ± 0.38	85.00 ± 0.25	80.45 ± 0.64	57.92 ± 0.32
CFL-W	86.83 ± 0.55	93.72 ± 0.93	87.36 ± 0.42	89.47 ± 0.74	85.25 ± 0.17	77.35 ± 0.81	58.70 ± 0.13
IFCA(T)	88.77 ± 0.02	90.06 ± 4.62	86.22 ± 0.22	86.36 ± 0.14	85.12 ± 0.09	81.53 ± 0.05	47.76 ± 1.98
IFCA	88.78 ± 0.02	90.49 ± 4.73	86.21 ± 0.28	86.56 ± 0.21	85.06 ± 0.04	81.51 ± 0.03	48.17 ± 1.30
IFCA-W	88.80 ± 0.12	92.84 ± 1.19	87.84 ± 0.14	90.81 ± 0.67	85.52 ± 0.50	79.17 ± 0.39	49.40 ± 0.76
FedDrift-Eager	88.76 ± 0.01	95.51 ± 0.18	87.86 ± 0.33	**94.09 ± 0.05**	86.64 ± 0.18	83.58 ± 0.79	61.62 ± 0.45
FedDrift	88.77 ± 0.02	**95.54 ± 0.15**	**88.13 ± 0.39**	94.03 ± 0.02	**86.68 ± 0.20**	**85.72 ± 0.07**	**64.91 ± 0.31**
FedDrift-C	88.82 ± 0.09	95.12 ± 0.50	87.78 ± 0.44	93.97 ± 0.05	86.21 ± 0.40	85.62 ± 0.47	61.86 ± 0.30
FedDrift-Eager-W	88.82 ± 0.12	95.05 ± 0.67	87.87 ± 0.23	92.04 ± 0.03	86.44 ± 0.20	82.15 ± 0.32	61.94 ± 0.38
FedDrift-W	**88.88 ± 0.15**	95.35 ± 0.08	87.95 ± 0.15	91.93 ± 0.03	86.46 ± 0.31	83.29 ± 0.06	64.22 ± 0.60
Oracle	88.77 ± 0.01	95.25 ± 0.52	87.99 ± 0.20	94.11 ± 0.02	86.89 ± 0.17	86.10 ± 0.03	-
Oracle-W	88.77 ± 0.03	95.51 ± 0.10	88.15 ± 0.14	92.03 ± 0.01	86.83 ± 0.06	83.58 ± 0.03	-

more fundamental obstacle is that each global model trained by AUE-PC is updated by all clients. In the course of the 2-concept staggered drift, all of the models in the ensemble are trained either over a mixture of data from both concepts or solely from the first concept, and there is no accurate model available that is a good fit for the second concept.

The Window-2 algorithm and the weighted sampling algorithms Weighted-Linear and Weighted-Exp are techniques for forgetting older data, but less abruptly compared to Window-1, and in general they all perform similarly. On the sharp drift of SINE-2, the fastest forgetting algorithm Window performs the best of these. On the other hand, on the 4-concept drift of MNIST-4 in which the time axis does not well separate different concepts, the slowest forgetting algorithm Weighted-Linear performs best. Meanwhile, the performance of all four algorithms are close on the SEA datasets, which have greater overlap between the concepts.

The clustering algorithms CFL and CFL-W start with each client in one cluster, and recursively split clusters over rounds and over time based on the intra-cluster similarity of their local updates. We observe that the CFL-W variant is the better-performing of the two on each dataset except MNIST-4 (which is also the only dataset where Oblivious outperforms Window), and is a consequence of the passive drift adaptation of its sliding window which forgets older data. The performance of CFL-W is relatively high on SINE-2 and CIRCLE-2. As an example, the clustering learned on SINE-2 is shown in Figure 5.8. We observe that, for the first 6 time steps, it correctly distinguishes the two concepts by using distinct models. The disadvantage of the clustering of CFL-W is that it creates excess models for the same concept and does not take full advantage of collaborative training. At time 5, it is limited to splitting its cluster for model 0 when the green concept occurs, but cannot merge the drifted clients to the existing cluster created for the green concept at the previous time step. This limitation of only being able to subdivide existing clusters,

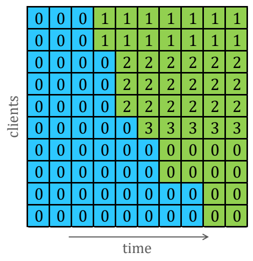


Figure 5.8: The clustering learned by CFL-W on SINE-2. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept.

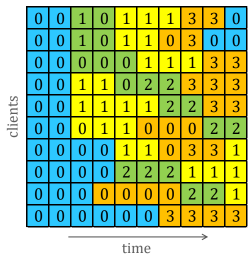


Figure 5.9: The clustering learned by FedDrift-Eager on MNIST-4. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept.

but not merge clusters or re-assign clients to existing clusters results in poor performance on more complex drifts.

For IFCA, IFCA-W, and IFCA(T), the clustering is pre-initialized with a random model for each concept that can occur over time for each dataset. In general, we observe that this is not a reliable method for reacting to drift. All the IFCA variants perform well under the sharp label-swap drift of SINE-2. When the new concept occurs, the drifted clients cluster to the second model, and the learned clustering matches the ground-truth. On CIRCLE-2, we found that IFCA and IFCA(T) learned the correct clustering in 2 out of 5 trials, and otherwise used only a single model in the other 3 trials. IFCA-W learned the correct clustering in 1 out of 5 trials. (Note the high standard deviation in Table 5.4.) Across the SEA and MNIST datasets, none of the three algorithms ever used more than a single model (with one exception—on SEA-4, in 1 out of 5 trials, IFCA-W used a distinct model for the yellow concept). For the SEA and MNIST datasets, we observe that the IFCA and IFCA(T) degrade to the Oblivious algorithm, and that IFCA-W degrades to the Window algorithm. On the FMoW dataset, we observe again that random initialization can sometimes address drift, but unreliably: in 1 out of 5 trials each for all IFCA variants, a separate model is used for the Africa region at later time steps. (However, the IFCA variants are among the worst performing in our evaluation because their random initialization precludes the pre-trained ImageNet initialization we use for other algorithms.) The authors of the original paper on IFCA note that the accuracy of the clustering is sensitive to the initialization of the models, and propose random restarts to address this issue, but restarts do not translate well to the time-varying setting we study. In our work, FedDrift-Eager and FedDrift address the initialization problem by using drift detection to deal with new concepts as they occur and to cultivate new clusters.

For FedDrift-Eager-W and FedDrift-W, restricting to a window has minimal impact on the accuracy for the SEA dataset. There is a significant loss of accuracy for the MNIST dataset relative to the non-windowed versions, but note that the same significant loss occurs when going from Oracle to Oracle-W, so this loss is a result of windowing, not specific to our algorithm. Indeed, the accuracy of FedDrift-W is quite close to Oracle-W.

The communication-efficient FedDrift-C. As noted in §5.4, one of the drawbacks of FedDrift is that it can create more models M compared to FedDrift-Eager, adding to the communication cost of sending $O(MP)$ models. The goal is to only use a number of global models close or equal to the number of distinct concepts, and while FedDrift can hierarchically merge created models of the same concept, FedDrift can observe temporary spikes in the number of global models. To mitigate this cost, we evaluate FedDrift-C, which differs from FedDrift in that, at each time after drift occurs, only one random client that drifted contributes its local model as a global model. In the case that multiple new concepts occur at a time, only one of the new concepts will be learned immediately, but clients that are still at an unlearned concept are eligible to detect drift again at the following time step and get another chance to contribute its local model. Meanwhile, while a concept goes unlearned globally, drifted clients do not contribute to any of the global models.

For the 4 concepts in MNIST-4, we observed that FedDrift learned a total of 7 global models (later merged down to 4) as shown in Figure 5.7 in §5.6. FedDrift-C more efficiently maintained a maximum of 4 global models across all time, at a penalty of 0.87% accuracy due to the delayed learning of one of the two simultaneously arising concepts. Meanwhile, FedDrift-Eager suffers a larger 4.88% penalty after it incorrectly merged the two simultaneous concepts, as shown in Figure 5.9—model 1 is initially trained over the green and yellow concepts, and while the clients at the green concept later abandon model 1 and eventually learn a separate model 2, the green concept training data still poison both model 0 and model 1.

We quantify this accuracy-communication trade-off in Figure 5.10 where we show the average test accuracy and total number of models sent by FedDrift-Eager, FedDrift, and FedDrift-C under various selections of the drift detection threshold δ. Increasing the value of δ restricts cluster splitting (increases false negative detections) and promotes cluster merging, which reduces the number of models and concepts learned (at $\delta = 1$, each algorithm is identical to Oblivious). Empirically, we confirm that choosing larger settings of δ can trade-off accuracy for efficiency. (Choosing δ too small for FedDrift can also negatively affect accuracy due to increased false positive detections, but to a lesser degree because the hierarchical clustering of FedDrift can correct some false positives—see below on Impact of False Positives.) We observe that, generally, using FedDrift-C over FedDrift preserves most of the accuracy improvement over Oblivious while saving communication—with one exception at the largest $\delta = 0.20$ where both algorithms are susceptible to false merging, but FedDrift has more total models added to make the mistake of merging two concepts that FedDrift-C avoids. We also observe that the Pareto front is mostly configurations of FedDrift and FedDrift-C over FedDrift-Eager. Finally, we observe that all variants of FedDrift are more efficient than ensemble algorithms—relative to Oblivious, FedDrift variants send 2–3x models compared to AUE which sends 5x—because for ensembles, clients contribute to every model at each communication round, compared to FedDrift where clients contribute only to the clusters they belong to (the broadcast of all models for clustering in FedDrift is only once per time step).

Random Drift Patterns. Throughout this chapter, we have considered the 4-concept drift pattern in Figure 5.3 in §5 as a specific concrete example in order to depict the challenges in distributed concept drift, motivate the design of FedDrift, and discuss the experimental performance by comparing the learned clustering matrix to the ground-truth. To examine the performance more generally, we consider a family of datasets MNIST-R with random concept changes. Using the same four concepts as in MNIST-4, MNIST-R is generated with all clients at the first concept to

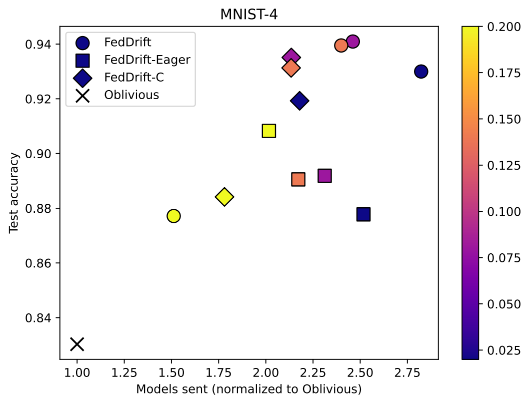
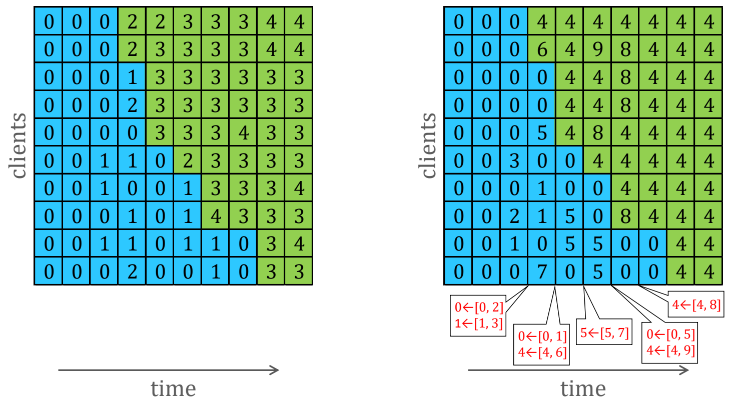


Figure 5.10: The accuracy-communication trade-off on MNIST-4 for FedDrift-Eager, FedDrift, and FedDrift-C. Each algorithm is evaluated under various selections of the splitting/merging threshold δ between 0.02 and 0.20, indicated by color. The vertical axis is the average test accuracy across clients and time, omitting drifts. (1 trial)

start, and then each client independently randomly observes one of the four concepts every two time steps (as opposed to every time step which is not possible to adapt to). Across 5 random seeds, the average accuracy is shown in Table 5.6 (and in Table 5.7 for all time including drifts). We generally observe the same relative performances of each algorithm as on the previously specified MNIST-4 drift. The performance of FedDrift is close to that of Oracle, FedDrift-C is close behind, FedDrift-Eager is lower given that it is likely to have multiple new concepts occurring simultaneously in MNIST-R, and then all prior baselines follow.



(a) FedDrift-Eager (b) FedDrift

Figure 5.11: The clustering learned on SINE-2 when $\delta = 0.01$. Each cell indicates the model ID at each client and time step, and the background color indicates the ground-truth concept.

Impact of False Positives. To demonstrate the application of the hierarchical clustering in FedDrift, in §5.6 we discussed the example of the learned clustering for MNIST-4 in Figure 5.7. Here in Figure 5.11 we present another example on SINE-2 at a small $\delta = 0.01$ (corresponding

Table 5.6: Average test accuracy (%) under random drift patterns, omitting drifts (5 trials)

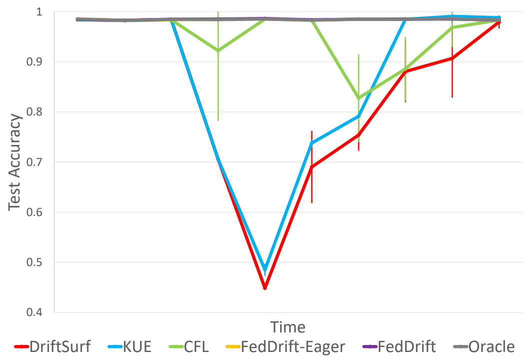
	MNIST-R
Oblivious	85.12 ± 1.37
DriftSurf	85.03 ± 1.36
KUE	81.56 ± 1.90
AUE	83.87 ± 1.64
AUE-PC	83.67 ± 1.66
Window	82.37 ± 1.94
Window-2	83.65 ± 1.83
Weighted-Linear	84.87 ± 1.34
Weighted-Exp	84.60 ± 1.44
Adaptive-FedAvg	83.17 ± 1.51
CFL	84.20 ± 1.54
CFL-W	82.24 ± 1.77
IFCA(T)	84.50 ± 1.21
IFCA	84.39 ± 1.45
IFCA-W	85.93 ± 3.35
FedDrift-Eager	89.85 ± 1.49
FedDrift	**94.06 ± 0.38**
FedDrift-C	92.76 ± 0.56
FedDrift-Eager-W	86.60 ± 2.27
FedDrift-W	90.83 ± 0.17
Oracle	95.03 ± 0.15
Oracle-W	91.66 ± 0.31

Table 5.7: Average test accuracy (%) under random drift patterns, including drifts (5 trials)

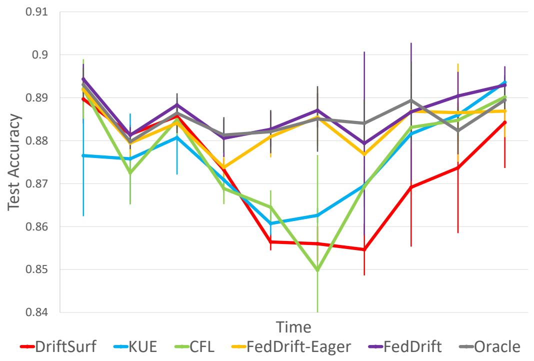
	MNIST-R
Oblivious	83.92 ± 1.23
DriftSurf	83.83 ± 1.21
KUE	79.77 ± 2.03
AUE	81.96 ± 1.03
AUE-PC	81.52 ± 1.43
Window	80.11 ± 1.45
Window-2	81.30 ± 1.58
Weighted-Linear	83.64 ± 1.21
Weighted-Exp	83.39 ± 1.29
Adaptive-FedAvg	81.41 ± 1.24
CFL	83.05 ± 1.37
CFL-W	80.58 ± 1.94
IFCA(T)	83.31 ± 1.11
IFCA	83.29 ± 1.29
IFCA-W	81.65 ± 0.67
FedDrift-Eager	85.26 ± 0.81
FedDrift	**86.77 ± 0.76**
FedDrift-C	86.65 ± 0.94
FedDrift-Eager-W	81.74 ± 1.60
FedDrift-W	83.70 ± 0.80
Oracle	87.32 ± 0.86
Oracle-W	84.29 ± 0.89

to more aggressive detection) to demonstrate an example of how hierarchical clustering can be beneficial even in the case of a 2-concept drift in mitigating false positives. At time 3, in both FedDrift-Eager and FedDrift there are three false positives, where in FedDrift-Eager, the new model 1 is retained but its underlying data forgotten, while in FedDrift, although initially 3 redundant models are created, they are all merged back with model 0 within 2 time steps, averaging their parameters and reincorporating their clustered data. The advantage of hierarchical clustering is also evident at time 4 when 2 false positives and 2 true positives occur together. In FedDrift-Eager, one new model is created for all the clients, but this new model is "poisoned" by contributions from the blue concept and does not work well at time 5, resulting in another drift detection to create model 3 (and forgetting about the data associated with model 2). FedDrift, on the other hand, creates models solely trained over either the blue and green concepts, and eventually merges all models of an identical concept, recovering all of the data. While the false positive mitigation demonstrated in this example is not a significant contributor to the observed higher accuracy of FedDrift in our evaluation because we use higher δ values as noted in §5.5.2, it is relevant when there is greater uncertainty in selecting the threshold hyperparameter.

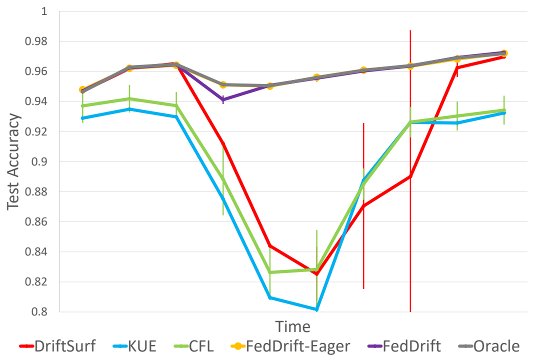
Test Accuracy Over Time. Finally, in Figure 5.12, we include remaining plots for the accuracy over time for FedDrift-Eager, FedDrift, and selected baselines representing drift detection, ensembles, and clustered FL, supplementing Figure 5.6 in §5.6. (Note the varying scales of the y-axes.) Similarly, here we observe the same general trends: (i) the centralized drift adaptation algorithms suffer in performance, particularly during the transition period when no one model works well across all clients; (ii) CFL can react to the drift early on SINE-2 as with CIRCLE-2 before, but its performance degrades with excessive further splits; (iii) for the 4-concept drift in SEA-4 and MNIST-4 centralized baselines and CFL never recover in performance with multiple concepts present; and (iv) on SEA-4 and MNIST-4, FedDrift is close to Oracle except for a gap at time 3 when it uses local models prior to merging, while FedDrift-Eager lags behind FedDrift when it creates a single model for the 2 simultaneously arising concepts but can slowly recover with further detections.



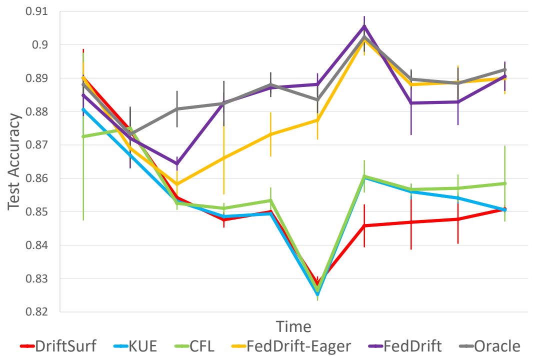
(a) SINE-2



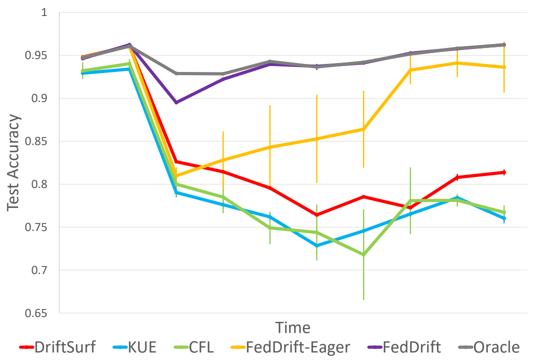
(b) SEA-2



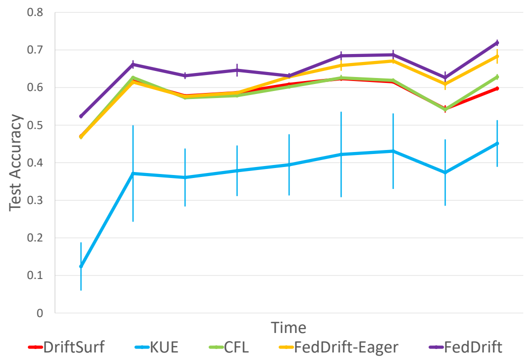
(c) MNIST-2



(d) SEA-4



(e) MNIST-4



(f) FMoW

Figure 5.12: Test accuracy of selected algorithms at each time on SINE-2, SEA-2, MNIST-2, SEA-4, MNIST-4, and FMoW. Vertical lines represent standard deviations.

Chapter 6

Conclusion

This thesis contributes new algorithms for efficient incremental training over continuously arriving data in the pursuit of high prediction accuracy at all times. In increasing level of data heterogeneity:

- STRSAGA learns from IID data arrivals with efficient model updates (Chapter 3).
- DriftSurf adapts to non-IID data over time using drift detection in a more statistically accurate stable-state/reactive-state process (Chapter 4).
- FedDrift adapts to non-IID data over time and distributed across space in the FL setting by learning a time-varying clustering (Chapter 5).

Theoretical and experimental results establish these algorithms as the state-of-the-art. In the following, we identify several directions for future work that complement this thesis.

Transfer learning Following drift detection in DriftSurf or FedDrift, a new model is initialized and trained solely over data from the new distribution. This is an inefficient strategy for large models and may suffer in accuracy in the small data regime after (subtle) drift, without transferring what can be learned from the previous model or data. However, the use of a new model trained from scratch is only for simplicity of presentation. Transfer learning can largely be viewed as a complementary question to drift detection, and where existing techniques can be applied in DriftSurf or FedDrift for the new model; e.g., weight sharing (typically retraining only the last layers of a neural network) [16] or biased regularization [90].

Largely unexplored is how to adjust and apply these transfer learning techniques during training in the streaming data setting, when the ratio of old to new data varies over time. For high accuracy at all times, it may be reasonable to impose a high bias shortly after drift towards the previous distribution, and wean off the bias as new data continue to arrive.

Furthermore, drift adaptation and transfer learning do need to be considered simultaneously in cases where drift can be identified as affecting only a local region of the feature space, and where the model structure can be adapted granularly. For decision trees, there are efficient strategies for replacing only some subtrees under drifts [7, 49]. Efficient adaptation for other model classes, such as mixture-of-experts networks, could be explored.

Hyperparameters DriftSurf and FedDrift adapt to drift by training multiple models, where models are distinguished solely by the underlying training dataset. Our focus is entirely on the viewpoint that newer models trained on more recent data are better in the presence of drift, and older models that are longer-trained using more historical data are better in the absence of drift, which is theoretically validated by better statistical accuracy asymptotically. Yet, the problem space for model training is much broader than identification of the training set, and the interaction between training hyperparameters and drift adaptation is of significant practical interest. One well-studied example is that the learning rate can be adaptive to streaming data to trade-off stability and plasticity, which underpins the prior work on Adaptive-FedAvg that we compared against in §5.6.

Other hyperparameters warrant more study, such as those governing the function class and model complexity warrant more study. For example, while ensemble methods are generally robust under a variety of drifts, their higher computational costs are in some settings prohibitive, but the cost could be mitigated by using cheaper models. In more sophistication, the model complexity could be set heterogeneously across models, depending on their role in the adaptation. Consider that both DriftSurf and FedDrift manage the trade-off between drift-robustness and efficiency by temporarily training excess models after drift is detected, and then drop or merge models after the uncertainty is resolved. Particularly in the small data regime after drift is detected, temporarily training lower complexity models may suffice.

Cross-device FL FedDrift is designed for the stateful FL setting and incurs communication costs proportional to the number of models. FedDrift is practical in the *cross-silo* FL setting over 10–1000 clients, but the "bottom-up" hierarchical clustering over local models (even temporarily created) will not scale to a *cross-device* FL setting with millions of participating clients; the distinction between the cross-silo and cross-device settings is detailed by Kairouz et al. [47]. The identification of distributed drift adaptation as time-varying clustering made in this thesis is still sound, but a more practical solution may necessitate designing a "top-down" clustering. For more communication-efficiency, it is also worth considering hybrid strategies that employ clustering at a higher tolerance of intra-cluster heterogeneity, in combination with other personalized FL techniques (e.g., fine-tuning, model interpolation).

Further motivation to identify other potential clustering strategies is for better privacy. While raw data remain with each client under FedDrift, local models are still shared for hierarchical clustering. Alternative clusterings could be amenable to rigorous privacy guarantees.

Fairness in FL Users from minority or marginalized sub-populations can observe fewer data points and at a slower rate. In FedDrift, local drift detection could take longer to trigger, and cluster merging could fail over lesser-trained local models. While unfairness is probably not made worse by using FedDrift (clients elect the best existing cluster at each time and are not forced to use worse models), it may not be alleviated either. One starting point for exploration is that cluster splitting/merging thresholds in FedDrift could be set differently across clients depending on their sample sizes.

Bibliography

[1] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. Hierarchical change-detection tests. *IEEE Trans. Neural Netw. Learn. Syst*, 28(2):246–258, 2016. 4.1

[2] Stephen H Bach and Marcus A Maloof. Paired learners for concept drift. In *ICDM*, pages 23–32, 2008. 4.1, 4.6.1

[3] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavaldà, and R Morales-Bueno. Early drift detection method. In *StreamKDD*, pages 77–86, 2006. 4.1, 4.3, 4.8.1, 5.4.1

[4] B. Bercu, B. Delyon, and E. Rio. *Concentration inequalities for sums and martingales*. Springer, 2015. 3.3.1

[5] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Nikolaos Karianakis, Yuanchao Shu, Kevin Hsieh, Victor Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. *CoRR*, abs/2012.10557, 2020. 1

[6] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *ICDM*, pages 443–448, 2007. 4.1, 4.3, 5.4.1

[7] Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pages 249–260, 2009. 4.8.7, 6

[8] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive online analysis. *JMLR*, 11:1601–1604, 2010. 4.6.2, 5.5, 5.5.1

[9] Léon Bottou and Yann LeCun. Large scale online learning. In *NIPS*, pages 217–224, 2003. 3

[10] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 9:323–375, 2005. 2

[11] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *NIPS*, pages 161–168, 2007. 2, 2, 2

[12] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *International Joint Conference on Neural Networks (IJCNN)*, 2020. 5.1, 5.5.1

[13] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Trans. Neural Netw. Learn. Syst*, 25(1):81–94, 2013. 4, 4.1, 4.6.1, 4.6.2, 5.5, 5.5.1

[14] Alberto Cano and Bartosz Krawczyk. Kappa updated ensemble for drifting data stream

mining. *Machine Learning*, 109(1):175–218, 2020. 5, 5.5

[15] Giuseppe Canonaco, Alex Bergamasco, Alessio Mongelluzzo, and Manuel Roveri. Adaptive federated learning in presence of concept drift. In *International Joint Conference on Neural Networks*, pages 1–7, 2021. 1, 5, 5.1, 5.5, 5.5.1

[16] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997. 6

[17] Fernando E Casado, Dylan Lema, Marcos F Criado, Roberto Iglesias, Carlos V Regueiro, and Senén Barro. Concept drift detection and adaptation for federated and continual learning. *Multimedia Tools and Applications*, pages 1–23, 2021. 5, 5.1

[18] Yujing Chen, Zheng Chai, Yue Cheng, and Huzefa Rangwala. Asynchronous federated learning for sensor data with concept drift. In *IEEE International Conference on Big Data*, pages 4822–4831, 2021. 1, 5, 5.1

[19] Chun Wai Chiu and Leandro L Minku. Diversity-based pool of models for dealing with recurring concepts. In *IJCNN*, pages 1–8, 2018. 4.1

[20] Gordon Christie, Neil Fendley, James Wilson, and Ryan Mukherjee. Functional map of the world. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6172–6180, 2018. 1, 5.5.1

[21] H. Daneshmand, A. Lucchi, and T. Hofmann. Starting small-learning with adaptive sample sizes. In *ICML*, pages 1463–1471, 2016. 3, 3.1, 3.2, 3.2.1, 1, 3.2.2, 3.2.2, 4, 5, 6, 3.2.2, 4.5.2, 38

[22] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Annh Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019. 4.6.2

[23] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014. 3, 3.1, 3.2

[24] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. 3.4, 4.6.2

[25] Moming Duan, Duo Liu, Xinyuan Ji, Yu Wu, Liang Liang, Xianzhang Chen, Yujuan Tan, and Ao Ren. Flexible clustered federated learning for client-level data distribution shift. *IEEE Transactions on Parallel and Distributed Systems*, 2021. 5, 5.1

[26] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Trans. Neural Netw.*, 22(10):1517–1531, 2011. 4.1

[27] Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass. In *Conference on learning theory*, pages 728–763, 2015. 3, 3.1

[28] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence-SBIA*, pages 286–295, 2004. 4.1, 4.3, 4.8.1, 5.4.1

[29] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia.

A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), 2014. 5.2

[30] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44, 2014. 1, 2

[31] Abhinav Garg, Naman Shukla, Lavanya Marla, and Sriram Somanchi. Distribution shift in airline customer behavior during COVID-19. *CoRR*, abs/2111.14938, 2021. 1

[32] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv:1906.06629*, 2019. 5, 5.1

[33] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *NeurIPS*, pages 19586–19597, 2020. 5, 5.1, 5.5, 5.7

[34] Yongxin Guo, Tao Lin, and Xiaoying Tang. Towards federated learning on time-evolving heterogeneous data. *arXiv preprint arXiv:2112.13246*, 2021. 5, 5.1

[35] Maayan Harel, Koby Crammer, Ran El-Yaniv, and Shie Mannor. Concept drift detection through resampling. In *ICML*, pages 1009–1017, 2014. 4.1, 4.3, 5.4.1, 5.4.2

[36] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM TIIS*, 5 (4):19, 2016. 3.4

[37] Michael Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, University of New South Wales, 1999. 4.6.2

[38] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020. 5.5

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5.5

[40] Brian Hentschel, Peter J Haas, and Yuanyuan Tian. Online model management via temporally biased sampling. *ACM SIGMOD Record*, 48(1):69–76, 2019. 4.1

[41] Fabian Hinder, André Artelt, and Barbara Hammer. Towards non-parametric drift detection via dynamic adapting window independence drift detection (dawidd). In *ICML*, pages 4249–4259, 2020. 4.1

[42] Elena Ikonomovska. Airline dataset, 2009. URL `http://kt.ijs.si/elena_ikonomovska/data.html`. 4.6.2

[43] Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018. 4.5.2

[44] Yibo Jin, Lei Jiao, Zhuzhong Qian, Sheng Zhang, and Sanglu Lu. Budget-aware online control of edge federated learning on streaming data with stochastic inputs. *IEEE Journal on Selected Areas in Communications*, 39(12):3704–3722, 2021. 1, 3.1

[45] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013. 3, 3.1

[46] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. 5.4.2

[47] Peter Kairouz, H Brendan McMahan, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021. 5.2, 6

[48] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010. 5.4.1

[49] Sebastian Kauschke and Johannes Fürnkranz. Batchwise patching of classifiers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 6

[50] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *VLDB*, pages 180–191, 2004. 4.1

[51] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *IDA*, 8(3):281–300, 2004. 4.1

[52] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M. Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A benchmark of in-the-wild distribution shifts. In *ICML*, 2021. 1, 1, 5, 5.5.1, 5.6

[53] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *JMLR*, 8:2755–2790, 2007. 4.1

[54] J. Konecnỳ and P. Richtárik. Semi-stochastic gradient descent methods. *arXiv preprint arXiv:1312.1666*, 2013. 3.1

[55] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016. 5

[56] Ivan Koychev. Gradual forgetting for adaptation to concept drift. In *ECAI Workshop on Current Issues in Spatio-Temporal Reasoning*, 2000. 4.1

[57] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5.5, 5.5.1

[58] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 5(Apr):361–397, 2004. 3.4

[59] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004. 4.6.2

[60] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020. 5.2, 5.3

[61] Yang Lu, Yiu-ming Cheung, and Yuan Yan Tang. Dynamic weighted majority for incremental

learning of imbalanced data streams with concept drift. In *IJCAI*, pages 2393–2399, 2017. 4.1

[62] Dimitrios Michael Manias, Ibrahim Shaer, Li Yang, and Abdallah Shami. Concept drift detection in federated networked systems. *arXiv preprint arXiv:2109.06088*, 2021. 1, 5, 5.1, 5.5.1

[63] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020. 5, 5.1

[64] MarketsAndMarkets. Federated learning solutions market by application (drug discovery, industrial iot), vertical (healthcare and life sciences, bfsi, manufacturing, retail and ecommerce, energy and utilities), and region - global forecast to 2028. `https://www.marketsandmarkets.com/Market-Reports/federated-learning-solutions-market-151896843.html`, 2021. 5

[65] P. Massart. *Concentration inequalities and model selection*. Springer, 2007. 3.3, 3.3.1, 3.3.1, 3.3.1

[66] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017. 5, 5.3

[67] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge university press, 2017. 3.3.1

[68] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *JMLR*, 19(72):1–5, 2018. 4.6.3, 4.8.7

[69] Yaqiong Peng, Zhiyu Hao, and Xiaochun Yun. Lock-free parallelization for variance-reduced stochastic gradient descent on streaming data. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2220–2231, 2020. 3.1

[70] Ali Pesaranghader and Herna L Viktor. Fast hoeffding drift detection method for evolving data streams. In *ECML PKDD*, pages 96–111, 2016. 4.1, 4.3, 5.4.1, 5.4.2

[71] Ali Pesaranghader, Herna L Viktor, and Eric Paquet. A framework for classification in data streams using multi-strategy learning. In *ICDS*, pages 341–355, 2016. 4.6.2, 5.5, 5.5.1

[72] Ali Pesaranghader, Herna L Viktor, and Eric Paquet. McDiarmid drift detection methods for evolving data streams. In *International Joint Conference on Neural Networks (IJCNN)*, 2018. 4.1, 4.3, 4.6.1, 5.4.1, 5.4.2

[73] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *NIPS*, 24, 2011. 3.1

[74] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1), 2020. 5

[75] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of*

Mathematical Statistics, pages 400–407, 1951. 3, 3.1

[76] N. L. Roux, M. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, pages 2663–2671, 2012. 3.1

[77] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 32(8):3710–3722, 2020. 5, 5.1, 5.5, 5.5.2, 5.7

[78] Raquel Sebastião and João Gama. Change detection in learning histograms from data streams. In *PAI*, pages 112–123, 2007. 4.1

[79] V. Shah, M. Asteris, A. Kyrillidis, and S. Sanghavi. Trading-off variance and complexity in stochastic gradient descent. *arXiv preprint arXiv:1603.06861*, 2016. 3.1

[80] Ammar Shaker and Eyke Hüllermeier. Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. *Neurocomputing*, 150:250–264, 2015. 4.8.8

[81] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. 5.4.2

[82] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2), 2000. 5.2

[83] Yu Sun, Ke Tang, Zexuan Zhu, and Xin Yao. Concept drift adaptation by exploiting historical knowledge. *IEEE Trans. Neural Netw. Learn. Syst.*, 29(10):4822–4832, 2018. 4.1

[84] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. ODIN: Automated drift detection and recovery in video analytics. *Proceedings of the VLDB Endowment*, 13(11), 2020. 1

[85] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004. 5.2

[86] Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of local-update sgd algorithms. *Journal of Machine Learning Research*, 22(213): 1–50, 2021. 5.3

[87] Ne Wang, Ruiting Zhou, Lina Su, Guang Fang, and Zongpeng Li. Adaptive clustered federated learning for clients with time-varying interests. In *IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2022. 3.1

[88] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996. 4.1

[89] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), 2019. 5

[90] Peng Zhao, Le-Wen Cai, and Zhi-Hua Zhou. Handling concept drift via model reuse. *Machine Learning*, 109:533–568, 2020. 4.1, 4.6.1, 6