

2017

Performance Envelopes of Adaptive Ensemble Data Stream Classifiers

Stefan Joe-Yen

Nova Southeastern University, sjoeyen@gmail.com

This document is a product of extensive research conducted at the Nova Southeastern University [College of Engineering and Computing](#). For more information on research and degree programs at the NSU College of Engineering and Computing, please click [here](#).

Follow this and additional works at: https://nsuworks.nova.edu/gscis_etd

 Part of the [Computer Sciences Commons](#)

Share Feedback About This Item

NSUWorks Citation

Stefan Joe-Yen. 2017. *Performance Envelopes of Adaptive Ensemble Data Stream Classifiers*. Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (1014)
https://nsuworks.nova.edu/gscis_etd/1014.

This Dissertation is brought to you by the College of Engineering and Computing at NSUWorks. It has been accepted for inclusion in CEC Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

Performance Envelopes of Adaptive Ensemble Data Stream Classifiers

by

Stefan Joe-Yen

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy


in

Computer Information Systems


College of Engineering and Computing
Nova Southeastern University

2017


We hereby certify that this dissertation, submitted by Stefan Joe-Yen, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.


Sumitra Mukherjee, Ph.D.
Chairperson of Dissertation Committee

Nov 9, 2017
Date

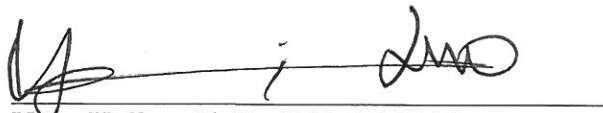

Michael J. Laszlo, Ph.D.
Dissertation Committee Member

Nov 9, 2017
Date


Francisco J. Mitropoulos, Ph.D.
Dissertation Committee Member

Nov 9, 2017
Date

Approved:


Yong X. Tao, Ph.D., P.E., FASME
Dean, College of Engineering and Computing

Nov. 9, 2017
Date

College of Engineering and Computing
Nova Southeastern University

2017

An abstract of a Dissertation submitted to Nova Southeastern University in Partial
Fulfillment of the Requirements for the degree of Doctor of Philosophy
Performance Envelopes of Adaptive Ensemble Data Stream Classifiers

by
Stefan Joe-Yen
Summer 2017

Abstract

This dissertation documents a study of the performance characteristics of algorithms designed to mitigate the effects of concept drift on online machine learning. Several supervised binary classifiers were evaluated on their performance when applied to an input data stream with a non-stationary class distribution. The selected classifiers included ensembles that combine the contributions of their member algorithms to improve overall performance. These ensembles adapt to changing class definitions, known as “concept drift,” often present in real-world situations, by adjusting the relative contributions of their members.

Three stream classification algorithms and three adaptive ensemble algorithms were compared to determine the capabilities of each in terms of accuracy and throughput. For each run of the experiment, the percentage of correct classifications was measured using prequential analysis, a well-established methodology in the evaluation of streaming classifiers. Throughput was measured in classifications performed per second as timed by the CPU clock. Two main experimental variables were manipulated to investigate and compare the range of accuracy and throughput exhibited by each algorithm under various conditions. The number of attributes in the instances to be classified and the speed at which the definitions of labeled data drifted were varied across six total combinations of drift-speed and dimensionality. The implications of results are used to recommend improved methods for working with stream-based data sources.

The typical approach to counteract concept drift is to update the classification models with new data. In the stream paradigm, classifiers are continuously exposed to new data that may serve as representative examples of the current situation. However, updating the ensemble classifier in order to maintain or improve accuracy can be computationally costly and will negatively impact throughput. In a real-time system, this could lead to an unacceptable slow-down.

The results of this research showed that, among several algorithms for reducing the effect of concept drift, adaptive decision trees maintained the highest accuracy without slowing down with respect to the no-drift condition. Adaptive ensemble techniques were also able to maintain reasonable accuracy in the presence of drift without much change in the throughput. However, the overall throughput of the adaptive methods is low and may be unacceptable for extremely time-sensitive applications. The performance visualization methodology utilized in this study gives a clear and intuitive visual summary that allows system designers to evaluate candidate algorithms with respect to their performance needs.

Acknowledgements

A project of this scope, even when it is primarily the result of a single author's vision and effort, involves the time and support of many people. I would like to thank the following individuals who have made a significant contribution to my work and life in Melbourne, FL during the course of this study over the past few years.

First, my gratitude to Sumitra Mukherjee, Ph.D., at Nova Southeastern University College of Engineering and Computing for supervising my work and providing advice and motivation throughout the process. Also, many thanks to the rest of my committee: Francisco Mitropoulos, Ph.D., and Michael Lazlo, Ph.D. for their additional guidance and support.

Next, I would like to express my gratitude to Northrop Grumman Corporation, and my managers including Gary Ranshous, for providing a flexible place to work and partial funding for my graduate studies. I would like to give special recognition to the research advisors and colleagues there who have influenced my work in advanced analytics and data science including Monte Hancock, John Day, Michael Black, Chadwick Sessions, Charles Ethan Hayes, Guy Hoenig (who introduced me to NSU), Adrian Peter, Ph.D., Donald Steiner, Ph.D., Brock Bose, Ph.D., Matthew Welborn, Ph.D. and Veronica Bloom, Ph.D.

Over the past few years, I suffered from chronic back pain due to degeneration and injury to multiple spinal discs. Thus, I give many thanks to the folks at Suntree Internal Medicine, Deuk Spine Institute, and Florida Sports and Spinal Rehab especially Bharat Patel, M.D., Edwin Chan, M.D., Tanya Schrumpf, D.C., Kristina Fowler, D.P.T., Daniel Ferrier, C.C.A., and Sarah Steele, L.M.T. who helped me get through the health related challenges as I completed this work.

Finally, I give my deepest thanks to my family - my wife, Cameron, for patiently supporting me, and my parents, Eugene and Kathleen, for initiating and supporting my constant quest for knowledge.

Table of Contents

Abstract

List Of Tables

List Of Figures

Chapters

- 1. Introduction** 1
 - Background 1
 - Problem Statement 6
 - Dissertation Goal 9
 - Concept Drift and Drift Mitigation 10
 - Research Questions 12
 - Relevance and Significance 13
 - Barriers and Issues 14
 - Assumptions, Limitations and Delimitations 15
 - Definition of Terms 16
 - Summary 17
- 2. Review of the Literature** 18
 - Real-time Pattern Analysis 18
 - Ensemble Classifiers 19
 - Selected Supervised Classifiers 20
 - Adaptive Update Windows 25
 - Prequential Accuracy Analysis 26
 - Concept Drift Mitigation Strategies 26
 - Summary 30
- 3. Methodology** 31
 - Overview 31
 - Experiment Framework 32
 - Experiments and Result Formats 39
 - Experiment Resources 42
- 4. Results** 44
 - Overview 44
 - Performance of Selected Algorithms on Concept-Drifting Data Streams 45
 - Time-series Characteristics of Classifier Error-rates 57

5. Conclusions, Implications, Recommendations, and Summary	65
Conclusions	65
Implications	67
Recommendations	68
Summary	69
A. Algorithm Notation Guide	70
B. Source Code	71
C. Additional Results, Figures, and Tables	83
Time-series Characteristics of Classifier Error-rates	83
References	99

List of Figures

- 1.1. Example of Performance Envelope Visualization 13
- 3.1. Experiment Framework Architecture 33
- 4.1. Trial B Accuracy on 10-Dimensional Instances 45
- 4.2. Trial B Accuracy on 50-Dimensional Instances 46
- 4.3. Trial B Relative Accuracy on 10-Dimensional Instances 47
- 4.4. Trial B Relative Accuracy on 50-Dimensional Instances 47
- 4.5. Trial B Throughput on 10-Dimensional Instances 48
- 4.6. Trial B Throughput on 50-Dimensional Instances 48
- 4.7. Trial B Stream Instance Generation Rate 49
- 4.8. Trial B Scaled Throughput on 10-Dimensional Instances 49
- 4.9. Trial B Scaled Throughput on 50-Dimensional Instances 50
- 4.10. Coefficient of Variance (CV) of Accuracy across Experiment Trials 51
- 4.11. Mean Accuracy on 10-Dimensional Instances 52
- 4.12. Mean Accuracy on 50-Dimensional Instances 52
- 4.13. Mean Throughput as a Fraction of Real-Time Processing (10D) 53
- 4.14. Mean Throughput as a Fraction of Real-Time Processing (50D) 53
- 4.15. Mean Throughput of slower algorithms (10D) 54
- 4.16. Mean Throughput of slower algorithms (50D) 54
- 4.17. Accuracy vs. Throughput (10D) 55
- 4.18. Accuracy vs. Throughput (50D) 57
- 4.19. Pegasos SVM Time Series (10D) 58
- 4.20. Pegasos SVM Time Series (50D) 59
- 4.21. Naive Bayes Time Series (10D) 59
- 4.22. Naive Bayes Time Series (50D) 60
- 4.23. Hoeffding Adaptive Tree Time Series (10D) 60
- 4.24. Hoeffding Adaptive Tree Time Series (50D) 61
- 4.25. Dynamic Weighted Majority Time Series (10D) 61
- 4.26. Dynamic Weighted Majority Time Series (50D) 62
- 4.27. Accuracy Weighted Ensemble Time Series (10D) 62
- 4.28. Accuracy Weighted Ensemble Time Series (50D) 63
- 4.29. Accuracy Updated Ensemble Time Series (10D) 63
- 4.30. Accuracy Updated Ensemble Time Series (50D) 64
- 5.1. Welch PSD for DWM, 10 Dimensional input stream 66
- 5.2. Welch PSD for HAT, 50 Dimensional input stream 66

5.3.	Correlations across classifiers, 50 Dimensional input stream at drift level (0.001)	67
C.1.	Accuracy vs. Throughput Trade-off (10D)	85
C.2.	Accuracy vs. Throughput Trade-off (50D)	86
C.3.	Pegasos SVM Time Series (10D)	87
C.4.	Pegasos SVM Time Series (50D)	88
C.5.	Naive Bayes Time Series (10D)	89
C.6.	Naive Bayes Time Series (50D)	90
C.7.	Hoeffding Adaptive Tree Time Series (10D)	91
C.8.	Hoeffding Adaptive Tree Time Series (50D)	92
C.9.	Dynamic Weighted Majority Time Series (10D)	93
C.10.	Dynamic Weighted Majority Time Series (50D)	94
C.11.	Accuracy Weighted Ensemble Time Series (10D)	95
C.12.	Accuracy Weighted Ensemble Time Series (50D)	96
C.13.	Accuracy Updated Ensemble Time Series (10D)	97
C.14.	Accuracy Updated Ensemble Time Series (50D)	98

Listings

- B.1. TDEF script 71
- B.2. StreamBenchmark script 77
- B.3. StreamExample script 80

Chapter 1

Introduction

Background

This dissertation presents experimental research that evaluates algorithmic alternatives for a prevalent approach to analytics on streaming data. In particular, it addresses the problem of maintaining acceptable classification accuracy and throughput in a variety of supervised binary classifiers including ensemble approaches that continuously process input data as it arrives. The goal is to provide designers of decision support systems with theoretically grounded, and empirically supported basis for selecting among numerous alternatives available for this class of algorithm.

The range of acceptable performance of an analytic system is defined by the nature of the intended application. Expected performance is subject to the constraints of specific characteristics of the data being processed and the available hardware. These constraints are often beyond the control of the system designer and end-user. However, due to the significant successes of a prolific research community, the machine learning approach has produced an embarrassment of riches when it comes to algorithms available for use. Only recently have any significant meta-analyses been performed to determine what the specific strengths and weaknesses of the various algorithms are in a systematic way.

In “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?” (Fernández-Delgado, Cernadas, Barro, & Amorim, 2014), a meta-analysis of 179 classifiers from 17 families of algorithms was conducted. This showed that decision-tree based learners

consistently performed exceptionally well. The study focused on evaluating non-streaming classifiers. With the continued rise of “Big Data,” The performance of data-stream classifiers is of growing importance. Thus we need a similar principled approach to comprehensively evaluating their performance characteristics. This study is a beginning toward that end.

This study focuses on a specific paradigm of machine learning algorithms and examines the behavior a small set of published algorithmic alternatives for classifying non-stationary streaming data. The results are intended to support principled guidelines for system architects to produce high-performance systems.

One of the main challenges for machine learning on streams is the need for constant adaptation and re-training. Since the complete data set is not available in a static form, evidence for sorting inputs into target classes changes over time. Thus data used to train supervised classification algorithms may become obsolete, especially if the definition of the target classes is subsequently modified. Re-defining the classes, or other factors which change the underlying data distribution, adversely affects the performance of the classifier ensemble. If the models learned by the component classifiers no longer reflect the current situation, the accuracy of ensemble classification will suffer. Improving results depends on detecting when the classifiers are no longer performing acceptably and then using new data to update the model. However, the adaptive update or retraining of the classifiers incurs a computational cost. Thus, model updates should only occur when it would significantly benefit the accuracy of the results.

In the current paradigm of large-scale stream data mining, real-time analysis presents new challenges to machine learning techniques that were developed in the context of mining of static databases, also called “data-at-rest.” Stream based pattern recognition software operates in-line as the data arrives. The data is usually not stored and thus not available for re-processing by the algorithm. This type of data is commonly referred to as streaming data, also called “data-in-motion.” The measurable indicators of underlying patterns in the data may change over time. For large-scale computing, also known as “Big Data,” this poses new

challenges in data mining algorithm design and implementation. The rate of data production in information systems has already exceeded the available resources for storing such data (Bifet et al., 2010). Also, the relevance of the data collected in many domains expires rapidly. End users in domains such as defense and finance look to automated information processing systems to produce timely and accurate analyses of rapidly changing data sources to support their decisions. Therefore, solutions that adapt to changing circumstances to maintain the accuracy of their results while continuing to process data rapidly are highly desirable. The demand for real-time systems is only expected to increase moving forward.

This research examines open questions related to maintaining high data processing throughput in supervised stream classification while preserving accuracy. The following sections and chapters describe the motivation for the questions that were explored in this study. The context in which classifier performance was examined is described briefly below. Detailed definitions of the experimental variables are presented in the “Problem Statement” section. An experiment framework to examine system performance is described in Chapter 3. Details of the hardware used are provided in the “Resources” section of that chapter. Source code and additional information is available in the appendices.

The specific aim of this research is to determine the performance bounds of adaptive stream classifiers. The experiments described here measured the impact on classification throughput of algorithms used to counteract concept-drift errors in single and ensemble-based supervised binary classifiers that operate on indefinitely large streams of inputs. The limits of performance are examined empirically and analytically to suggest a principled, quantitative approach for evaluating and extending such systems. The measured performance is discussed in a theoretical context of algorithmic complexity and scalability to separate the fundamental limitations of the selected algorithms from hardware-dependent concerns.

For the purposes of this study, the term “performance” refers to two measurable system characteristics – accuracy and throughput. For binary classification problems, acceptable levels of false-positives and false-negatives are problem dependent. In this study, the error

type was not relevant. Thus, the accuracy measured did not distinguish between false positive (Type I) and false negative (Type II) error rates. In the stream paradigm, measuring the continued accuracy of the system requires comparison of the outputs of the classifier with ground-truth that is recent enough to be relevant (Bifet, Read, Zliobaite, Pfahringer, & Holmes, 2013). These have been determined for each experiment by using ground-truth labeled data over a sliding window of classification results. The experimental framework utilizes an evaluation method known as prequential analysis to determine the evaluation window (Bifet et al., 2010). This is described further in Chapter 2 and Chapter 3

The experiment system collects counts of correct and incorrect classifications over the evaluation window in a contingency table also known as a confusion matrix (Chapter 3, Table 2). Thus, the accuracy of each approach can be compared on an overall scale as well as by individual error type. There are numerous statistics that can be computed from counts of Type I and Type II errors (Sokolova & Lapalme, 2009). Summary statistics that take both error types into account have been used when a single number representing accuracy is needed. The balanced error rate (BER) is defined as the arithmetic mean of each type of error rate. The F1-measure is a similar statistic that uses the harmonic mean of precision and recall which are ratios derived from Type I and Type II error rates. Certain problems and domains of interest are less forgiving of one type of error than the other. The F1-measure is a special case of the more general weighted F-measure that has equally weight on each error type. Additional derived accuracy measures provided by the MOA framework are κ statistics described further in Chapter 3. This study is interested in general results so neither error type is assumed to be more harmful. Thus, the simple prequential accuracy, p_0 was used as a summary of accuracy since it is readily available in the standard MOA prequential evaluation output. The computation of κ , as well as a brief discussion of using κ vs. p_0 for algorithms that measure accuracy in streams, is developed further in Chapters 3 and 5.

For evaluation purposes, the stream of inputs provided to the system was synthesized and assigned ground-truth categories (labels) by a concept-drift simulating data generator.

The adaptive classifiers in this study use various strategies that compare the classification outputs against ground-truth over a moving window of recent results to determine whether the current accuracy of the system is acceptable or if an update to one or more of the classifier models is needed.

Although platform differences affect the rate of data processing, the CPU and network resources of the experiment system were held constant during each major trial of the comparative evaluation. The ensemble throughput was measured for only a single input/output (I/O) stream with a different pseudorandom seed in each major trial. Possible speed gains from parallelizing the classification process are beyond the scope of this study but may be explored in future research. Thus, throughput is defined as the number of inputs processed per unit of time, nominally reported as “records per second.” For realistic problem sizes, the expected number of records per second will be on the order of tens of thousands or greater for real-time performance. Results are scaled accordingly or presented in scientific notation in graphs and charts to keep the numbers from becoming unwieldy. The absolute baseline of throughput is dependent on the I/O rate of the experiment framework. This was determined for each platform prior to running the experiment and used as a normalizing factor for the reporting of throughput.

A set of three concept drift adaptation algorithms for ensemble classifiers, and three basic data stream classification algorithms, described in a later section of this report, were deployed on a continuous stream of data generated by the experiment framework. The experiment library contains a drift data generator that changes ground-truth label assignments over time to produce shifts in the class boundaries at a user-specified rate. The adaptive classifier algorithms contain drift-detection features that update the model to prevent performance degradation over time. The ability of each classifier to react to the concept-drift was investigated at different drift rates.

To produce the performance envelopes central to this study, the average throughput of each adaptive algorithm was plotted along with its associated average accuracy as a

point in 2D space. The time course of drift adaptation was also rendered to gain insights into the performance of each algorithm. The details of how each algorithm varied as the rate of concept drift increased serves to characterize the effectiveness of its method of drift correction. Similarities between the algorithms were investigated by treating the time course of prequential accuracy as a response to an input signal.

The details of the modifications to well-established binary classification algorithms that have been developed to counteract the concept-drift effect mentioned above are discussed further in Chapter 2 and Chapter 3. This study investigates the effectiveness of such adaptive approaches to concept-drift mitigation in terms of how the speed of processing is affected by the various methodologies. The effective frontier of speed versus accuracy, available to each mitigation strategy, provides a basis for recommending their use in real-time systems. A key feature of rendering the results in a multidimensional plot is that both accuracy and throughput can be jointly assessed in a visually intuitive manner as an evaluation of overall system performance. This provides guidelines for selecting update thresholds for a particular algorithm as well as comparing the overall capabilities of different algorithms.

While this study focuses on binary classification, the ensemble approach is extensible to multi-class problems. The results of this research provides a quantitative approach and initial basis to allow system architects to make principled decisions when designing a pattern recognition system to analyze streaming data based on the performance requirements of their particular problem.

Problem Statement

The ongoing increase in the creation, collection, and transmission of data over networked resources has resulted in a situation where more data is being produced and disseminated than can be stored and examined in a practical amount of time. Therefore there is a rising need for analysis of data streams also called “data-in-motion.” A data stream consists of continuously arriving input information from data sources such as sensors, e-mails, or data

about network traffic and routing (Gaber, Zaslavsky, & Krishnaswamy, 2005).

In the stream paradigm, it is possible for the definition of patterns-of-interest to change over time. Thus, initial examples used to build the classifier model do not continue to represent the intended interpretation of the underlying patterns in the data. This leads to increased misclassification by the system. However, the members of the classifier ensemble are continually exposed to new data that can serve as examples for training. Thus, in order to continue to produce correct results, each classifier can be updated to take new examples into account while discounting the obsolete examples. Even with algorithms designed to continually adapt to changing definitions, updating the classifiers in the ensemble in order to maintain or improve accuracy is computationally costly and negatively impacts throughput. Thus, the aim of this research is the investigation of the computational cost of several adaptive methods used to maintain the accuracy of classification. Insights on maintaining the balance between accuracy and throughput adds to the body of knowledge in the supervised classification of data streams.

The transient, time sensitive nature of data-in-motion makes it desirable for processing to occur in real-time. This means that the data cannot always be assumed to be stored for later analysis. Any processing of the data needs to be completed before a user requests the results.

The data stream paradigm is particularly challenging to supervised pattern recognition algorithms since examples of all categories in the data may not be present to the system at the outset. Also in a multi-class situation new examples may not be consistent with any previously observed category requiring the system to incorporate methods for discovering new categories (Masud, Gao, Khan, Han, & Thuraisingham, 2010).

In such a supervised or semi-supervised machine learning system, the definition of underlying categories to be learned from the training data may not be stationary in time. This leads to a phenomenon known as “concept drift” that can adversely affect the accuracy of an automated system when it is applied to prediction or recognition problems. Concept

drift changes existing class boundaries. It can also introduce new categories over time. Controlling sources of variation is integral to the experiments discussed in this report. The intrinsic degradation of the classifier ensemble needs to be isolated from other sources of variation. Thus, this study is not concerned with novel class detection but limit the drifts to changing the definition of each class. The experiment framework modifies the output of the labeling function over time as described below. The categories are set up to form a two-class (binary) classification problem. However, the assignment of ground-truth labels varies with time. This study uses the freely available Massive Online Analytics (MOA) framework (Bifet et al., 2010) to generate a concept drifting stream of data that simulates a real-world condition in which sensor outputs may be subject to persistent concept drifts.

This study evaluates and summarizes the performance of concept drift correction approaches to classification problems on data streams. Six stream classifiers are applied to an input stream that has a configurable drift-rate parameter. The responses of stream-enabled variants of Naïve Bayesian (NB), Pegasos Support Vector Machine (PEG), and a Hoeffding Adaptive Tree (HAT) classifier are compared against the same stream segment comprising 10^8 instances to be classified. These classifiers were also used as components to three ensemble classifiers described further in Chapters 2 and 3. The specific drift correction and ensemble methodologies are also detailed in Chapters 2 and 3. The outputs of the system were examined over time to detect changes in accuracy. Adaptive updates to the component or combined classifiers were applied during the course of normal operation in all of the ensemble methods in an attempt to counteract concept drift.

The classifiers, ensembles, concept-drift stream generator, and prequential evaluation engine were implemented using the open-source, Massive Online Analysis (MOA) framework (Bifet et al., 2010) as well as custom software in Python, Java, and Matlab written for this effort.

Dissertation Goal

The motivation for this research was to evaluate the impact of adaptive concept-drift mitigations on stream processing rates. As described above, for the streaming domain, throughput is a significant factor in evaluating system performance. Ideally, processing speed must be maintained at line-rates to be considered successful for real-time applications. To counteract concept drift, it is important to efficiently determine the window of relevant examples for supervised and unsupervised classification of streams. One method uses probabilistic sampling to operate within space and time constraints (Aggarwal, 2006). This study focuses on evaluating various classifiers' responses to concept-drift. Classifier diversity has been identified as a positive contributor to the robustness of the combined answer in such ensemble classifiers (Minku, White, & Yao, 2010). The study makes use of ensembles of a variety of established supervised classifiers such as SVM that have been modified to handle streaming data and new classifiers specifically developed for streams (Bifet & Kirkby, 2009). Many stream-based ensemble algorithms have built-in adaptive strategies for retraining these classifiers when changes in the underlying class definitions occur. This study describes the effects of these mitigations on throughput and accuracy to determine if any of these approaches can maintain or improve throughput without sacrificing accuracy.

Demand for solutions to the data-in-motion problem is expected to increase in the coming years since the number of available data sources and the need for rapid analysis of data are only expected to increase in the future (Bifet & Kirkby, 2009). Future systems should benefit from exploring new methods to efficiently adapt to evolving data streams.

Concept Drift and Drift Mitigation

Formal Definition of Concept Drift

In the simulation provided by the experiment framework, time is measured in discrete steps, t . At each time step, input x , is assigned a true class label, $L_t(x)$, and an estimated class label produced by the classifier ensemble $C_t(x)$. The classifier ensemble combines estimated labels produced by each of its component classifiers, $C_{it}(x), \{1 \leq i \leq n\}$, where n is the number of classifiers, to produce a single result. Differences between the true class label and the estimate are considered errors. The labeling function produces a two-class distribution of examples. Type I, and Type II error rates are measured over a sliding window and used to compute the prequential accuracy that is reported by the evaluator at regular intervals.

The main object of this study is optimizing performance of streaming classifiers under concept drift. The set of “concepts” was operationally defined as a set of two class labels, one of which is applied to each instance in a stream of exemplars in the form of real-valued attribute vectors, x_t , where t is an integer indicating the discrete time-step of the experiment. The target classification labels were applied by a labeling function, using a concept-space model provided by the MOA framework that randomly selects examples of each concept from Gaussian-distributed hyperspheres of varying density. The MOA framework has methods to create the initial concept space by randomly generating one or more ground-truth centroids, g_ℓ for each class label, ℓ . Each centroid was also assigned a standard deviation at random. The labeling function then assigned labels to randomly selected data points depending on their position relative to a centroid.

For each experiment run, the centroids were labeled with one of two classes, $\ell = \mathbf{class1}$ or $\ell = \mathbf{class2}$. New examples are generated by randomly selecting a centroid and then generating a displacement vector d_t that is randomly drawn from a Gaussian distribution with the centroid’s associated standard deviation. The example attribute vector at time t is then defined as , $x_t = g_\ell + d_t$. The ground truth label for this vector is defined as $L_t(x_t) = \ell$.

The output of this processing step is a pair $\langle x_t, L_t(x_t) \rangle$ representing a data point and

its target class. Concept drifts are then produced by varying the location of the centroids. At the beginning of each experiment, each centroid receives a random drift direction. The centroid then moves at a specified speed over the course of the experiment. In order to ensure a balanced training set, the labels were assigned so that approximately 50% of the centroids belong to the class, $\ell = \mathbf{class1}$.

The simulation begins at initial time, t_0 . At each time interval, $t = t_0 + n\Delta t; \{n = 0, 1, 2, \dots\}$, class centroids are displaced from their initial positions at a user-specified rate. Over time, the examples labeled as **class1** or **class2** will come from different regions of the attribute space. This produces the concept drifts central to the study.

Adaptive Mitigation Strategies

The adaptive behavior of the system depends on a measurement of the error rate of the classifier ensemble. The mitigation strategies evaluated in this study monitor both the individual classifier error rate at time t , $(E_i(t))$, and the ensemble classifier error rate, $(E(t))$ and use them to update the classifiers in order to maintain accuracy. The following different approaches to error mitigation on the throughput of the system were investigated.

1. Dynamic Weighted Majority (DWM) (Kolter & Maloof, 2007) described further in Chapters 2 and 3.
2. The Accuracy Updated Ensemble Algorithm (AUE) (Brzezinski & Stefanowski, 2014) described further in Chapters 2 and 3.
3. The Accuracy Weighted Ensemble Algorithm (AWE) (H. Wang, Fan, Yu, & Han, 2003) described further in Chapters 2 and 3.

To maintain throughput, retraining should be kept to a minimum. When available, algorithms that can update their classification functions continuously are preferable to those that need a large set of examples for explicit retraining. In general, an adaptive window (ADWIN) approach was applied to detect concept drift and determine how often classifiers need to be updated with new ground truth labels from the recent window. When the system

starts operation, the window size is set to an initial value. If a drift to an unacceptable error rate is detected often, the interval between classifier updates was decreased. Conversely, if the performance of the ensemble is determined to be stable, the window size was increased and the classifiers were not updated as frequently thus conserving resources.

Research Questions

Experiment Variables

The parameters that were varied experimentally were the rate of concept drift and the dimensionality of the input instances. The concept-drift rate was specified at one of three values with associated drift rate parameters described further in Chapter 3: none (0.00), low (0.001), or high (0.010). The input instances were generated at two complexity levels: 10 attributes-per-instance or 50 attributes-per-instance.

Experiment Metrics

The research goals were explored by collecting data to investigate the following research questions.

- What techniques for adaptively adjusting both the individual classifiers and the ensemble results cause the least reduction in the main variables: accuracy and throughput? Alternatives include selective retraining with recent labels, utilizing classifiers with on-line learning features that adapt continuously, and adjusting the weighted contribution of low performing classifiers to the ensemble result. Evaluation of the mitigation strategies characterizes the trade-off between throughput in records per second and prequential accuracy, for each mitigation strategy.
- What is the joint quantitative relationship between throughput and accuracy for each of the approaches? This can be summarized and visualized by considering each the measured throughput and accuracy of each approach as coordinates in a 2D performance space. A notional example of this is shown in Figure 1.1. In this example, four

hypothetical drift mitigation methods are shown with accuracy thresholds ranging from 60% to 95% and the associated throughputs as a percentage of maximal throughput (real-time). These represent the eponymous performance envelopes measured in this study.

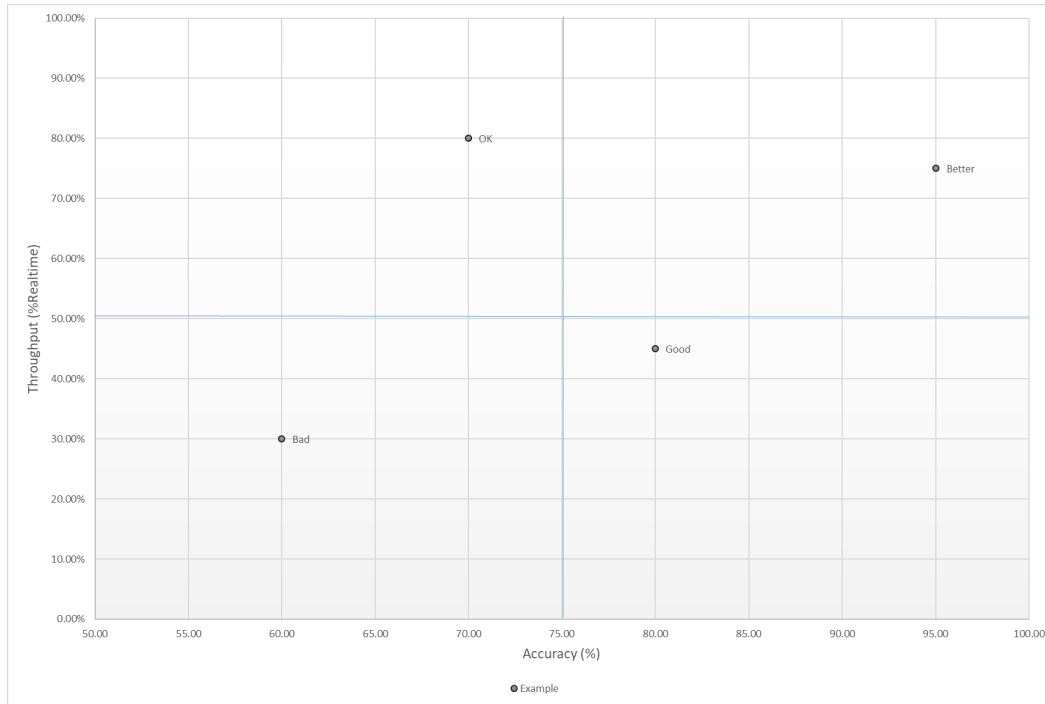


Figure 1.1: Example of Performance Envelope Visualization

Relevance and Significance

In the past few years, data mining on streaming inputs has risen to prominence in the pattern recognition community (Gaber et al., 2005)(Aggarwal, 2006). Specific toolkits for applying and evaluating machine-learning methods to the stream paradigm have been put forth (Bifet & Kirkby, 2009). While, techniques to evaluate stream classifiers that react to concept drift have been proposed (Abdualrhman & Padma, 2015), the precise nature of the effect of drift mitigation on the throughput of an ensemble of streaming classifiers has yet to be systematically investigated. This study explores the performance characteristics of adaptive updating in an ensemble of supervised classifiers to provide quantitative advice

on architectural considerations for applying this approach in the streaming domain. The resulting performance envelopes allows system designers to make principled decisions on the suitability of various algorithms for real-time classification problems.

This study has implications beyond the direct domain of data stream mining. Improvements in real-time data processing of massive amounts of data would inform improvements in other areas of Pattern Recognition, Artificial Intelligence and Machine Learning. These results are also relevant to the fields of Knowledge Discovery in Databases (KDD) and Decision Support Systems (DSS). It is hoped that these findings will lead to improvements in classifiers for real-time and online learning problems.

Barriers and Issues

As discussed earlier, the problem of analyzing data-in-motion is inherently difficult due to several factors. The primary factor is the sheer volume of data being produced by information systems worldwide. This data is being captured because it is presumed that it will be useful. However, if it cannot be analyzed and acted on in a timely manner, then acquiring the massive quantity of measurements becomes a wasted effort. The advent of powerful portable devices has enabled ubiquitous computing. However, despite advances in storage technology, portable devices do not have the capacity to store and process extensive databases. In addition to technical barriers, there are social and political aspects of analyzing data-in-motion because of security and privacy concerns.

Even though online learning has been investigated in the machine learning community for some time, the problem of processing large-scale, continuous real-world data has come to the attention of a broader community in the past three to five years. Stream mining techniques have been an active topic of investigation but no standard solution to efficient stream classification, especially on non-stationary data, has been arrived as yet.

As data production and collection continue to be applied to more segments of the population, it is critical for practical data mining techniques to keep up with a rate of data

collection that continues to out-pace the capacity to store it. The goal of this research is to investigate analytic methods that will scale along with this increase.

Assumptions, Limitations and Delimitations

These experiments operate under the assumption that the input data consists of a series of real-valued inputs collected at discrete intervals of time. For simplicity, these intervals are uniformly spaced and an input value will always be available at time t .

An essential feature of this study requires that the distribution of the underlying data changes over time. An assumption is that the sampling experiment is conducted over enough time samples for the distribution change to have a noticeable effect on the classification model. The adaptive mitigations to these concept drifts described in this report are limited to systematic shifts. The underlying distribution of the data must retain enough structure that a model can be formed. This model is subsequently invalidated when the distribution changes again. Corrections are applied to produce a new model when the system determines that the error rate of the current classification model has reached an unacceptable level.

Throughput is a primary metric examined in this study. Since the heterogeneous ensemble used by DWM, described further in Chapters 2 and 3, depends on multiple classifier types, each with different intrinsic classification rates, the throughput of the ensemble is limited by the throughput of the individual classifiers. In the simplest case, the maximum rate of the ensemble is no greater than the maximum rate of its slowest component classifier. This is controlled for by assuming the processing rate to be averaged over a sufficiently large time interval that the result accurately reflects the long term performance of the system.

Also, the rate at which inputs are presented to the classifier represents an upper-bound to the maximum throughput. This is controlled for by measuring the rate of a null-processor (i.e. the stream input instances are generated but not processed). This gives the “unburdened” baseline rate of the stream processing to which each classification strategy can be compared.

Definition of Terms

Adaptive mitigation – changes made by the system to improve performance of the classifier when the error rate is determined to be too high.

Binary classification – automated categorization of input data into two categories. Each item of input is assigned to either Class 1 or Class 2.

Classifier – a function that assigns a category to an input vector. Classifiers are developed by applying a training algorithm that examines example input data and builds a model to assign categories to previously unseen data. If the examples are provided with correct labels already assigned, the training algorithm is known as supervised learning. If no labels are provided, the examples are partitioned into categories based on intrinsic features of the data. This training method is called unsupervised learning.

Concept drift – a change in the class definitions over time. For this study time is measured in discrete intervals. Thus, an input that belonged to one category at time t now belongs to a different category at time $t + n$, where $n \in \mathbb{Z}$.

Ensemble classifier – a classifier function that uses the results of several independent classification models to decide the category label of each input. Various strategies can be employed by an ensemble classifier to adjudicate the final answer. Most often it involves some form of weighted average of the outputs of the component models.

Major Trial – a complete set of experiment data comprising results from all six candidate classifiers in all six (3 drift rates \times 2 dimensionality) experimental conditions.

Prequential Evaluation – a streaming classifier evaluation paradigm where instances from the stream are used first to test and then to train the streaming classifier. Accuracy is determined either in a sliding window or in some cases from the beginning of the stream output.

Streaming data – data that are produced, transmitted or recorded sequentially. The data stream is formatted as a time series where time can be interpreted as continuous (as in streaming video) or discrete as in periodic sensor measurements. A data stream can be denoted formally as a series of ordered pairs (x, t) where x represents an n -dimensional data vector ($x \in \mathbb{R}^n$) and t represents its associated time-stamp ($t \in \mathbb{Z}$ for discrete cases; $t \in \mathbb{R}$ for continuous scenarios).

Throughput – a measure of the amount data processed per unit of time. The granularity of elements processed and time units are defined in the context of the task.

Summary

In order to gain insights on efficient means for categorizing massive amounts of data, various algorithms for supervised binary classification including ensembles with drift detection were applied to a sequence of data with an underlying distribution that evolves in time. The classification models were compared in terms of accuracy and throughput. The detailed time course of shifts in accuracy was also examined. The insights gained in this study can be used to recommend improved methods for working with massive stream-based data sources.

Chapter 2

Review of the Literature

Real-time Pattern Analysis

One of the most challenging aspects of data stream mining are the time and space constraints of processing massive amounts of data in real-time. The data typically arrive at a rate that is too rapid to store and process offline. The relevance of the data may also have a short time window. An example of this situation is the analysis of real-time sensor data. Also, in some problem domains, the features of the data being analyzed may be expected to change in time. In these cases, analyzing historical data may do more harm than good (Dulhare & Premchand, 2010).

In many real time systems, data is not stored once it is processed (Bifet & Kirkby, 2009). This constraint means that machine learning evaluation techniques such as n-fold cross-validation and classical, iterative machine learning paradigms such as Self-Organizing Maps (Kohonen, 1982) which require several passes over the data examples are not applicable. New techniques must be feasible in the single-pass case. The single pass paradigm of stream mining also includes the goal of processing incoming data at or faster than the rate of arrival (Gaber et al., 2005).

Trainable pattern recognition systems are categorized as *supervised* or *unsupervised* depending on whether the classifier function is built by incorporating examples that are explicitly labeled with the target class (Bishop, 2007). In the supervised case, the system is architected to minimize the error between its output in response to an input set and a set

of known target outputs for the same input set.

For stream processing, observations that provide discriminative features of the categories to be learned may only become available over an extended time course. Also, the categories of interest commonly evolve in time. This leads to the phenomenon of concept drift (Becker & Arias, 2007). Unsupervised methods, which do not depend on ground truth, can also be used in an attempt to discover patterns in the input values. These require the definition of a measure of similarity between inputs. The inputs are transformed into a vector space model that facilitates distance measurements. Items close to each other in this space are considered members of the same class. This is known as *clustering*. Applying appropriately defined distance measurements is central to the success of these methods and new metrics are being researched and refined (Aggarwal, 2003). Stream-friendly adaptations to clustering include incremental vs. hierarchical and iterative methods (e.g., k-means) commonly used on data-at-rest (Ruiz, Menasalvas, & Spiliopoulou, 2009).

The non-stationary nature of data streams also requires adaptations in unsupervised clustering techniques where the cluster centers must be continuously updated in light of new data (Gaber et al., 2005). Clustering can also be considered multi-class classification. In that sense, real-time incremental feedback may also play a role in improving clustering.

Ensemble Classifiers

As with static data mining solutions, the approach of combining the outputs of multiple classifiers has been used to increase accuracy on data stream analytics. In an influential study (D. H. Wolpert & Macready, 1997), it was shown that no single model used for classification can provide optimal accuracy on all data that could be used as model inputs. The essential argument became known as the “no free lunch” theorem.

The solution to this issue is to devise methods of combining multiple classification models and has become a well-established method to improve correct classification of input. The optimal combination is accomplished by applying several models to the same input set and

adjudicating the responses to create systems called ensemble classifiers. Ensemble classifiers have since been shown to consistently outperform single classification models (Bauer & Kohavi, 1999) (Breiman, 2001).

The ensemble approach has been used in stream based classification problems to detect and manage concept drift in the streaming data by deferring decisions on the input data until several examples have been aggregated by the ensemble (Masud et al., 2010). The loss of discriminating features is a possible adverse consequence of aggregation and averaging.

The diversity of the classifiers in the ensemble has also been shown to be a significant factor in creating robust machine learning systems for data streams since they have been found to maintain a lower error rate after the onset of a concept drift event (Minku et al., 2010). This study uses a variety of supervised classification models that have proved successful in a variety of problem domains. The strengths, weaknesses, and adaptive modifications of various types of supervised classifiers is discussed below. The stream-friendly versions of these algorithms are either present in the MOA toolkit or readily implemented using MOA's extension API.

Selected Supervised Classifiers

This section is divided into subsections that briefly review the characteristics of the classification algorithms used in this study. These algorithms are relevant to the overall architecture and listed in no particular order. Further algorithmic details specific to the experiment framework configuration are provided in Chapter 3. The basic operating principles, strengths, and weaknesses are briefly summarized.

Naïve Bayes (NB)

Basic Principle – The Naïve Bayes classifier rule estimates classes of the given input based on probability distribution of examples provided in training.

Operational Method – The class conditional probabilities for inputs, (x) and a set of k

target classes C_k is denoted $\Pr(x|C_k)$. These are computed from the target class labels associated with the example feature values in the training set. Then the posterior class probabilities are determined using Bayes' theorem in the form shown in Equation 2.1 (Bishop, 2007).

$$\Pr(C_k|x) = \frac{\Pr(x|C_k) \Pr(C_k)}{\sum_k \Pr(x|C_k) \Pr(C_k)} \quad (2.1)$$

Strengths – The model is efficient and can obtain the probabilities directly from the data without extensive preprocessing. Because the model is inherently probabilistic, a quantitative measure of confidence in the decisions is available.

Weaknesses – The term “naïve” in the name refers to the assumption of conditional independence. This assumption may not hold in the actual data leading to misclassification.

Adaptive Update Mechanisms – For Bayesian classifiers, online bagging and boosting (Oza & Russell, 2001) methods have been described that only require one pass through the training data as opposed to the multiple sampling from a static batch that these processes usually require. Online boosting in particular can be applied to this classification algorithm to update the current model based directly on the previous misclassification. The window of relevant results was adjusted dynamically as described below and in Chapter 3.

Support vector machines

Basic Principle – Support vector machines (SVM) are a classification method that searches for a decision surface that maximally separates the exemplars from each of two classes. The examples from each class that determine the margin are known as “support vectors.”

Operational Method – For the basic SVM classifier, the function that describes the de-

cision model is linear of the form shown in Equation 2.2 (Bishop, 2007).

$$y(x) = w^T \phi(x) + b \quad (2.2)$$

Training examples are given as a set of n input vectors $\{x_1, \dots, x_n\}$ and associated target values $\{t_1, \dots, t_n\}$ taken from the binary set of class labels $\{-1, 1\}$. The $\phi(x)$ are fixed transformation functions (kernels) that may be applied to convert the feature vectors to a form that aids optimization of the model. The weight (w) and bias (b) parameters that maximize the margin are then derived as a quadratic optimization problem (Chang & Lin, 2011). The trained classifiers have a target output given by:

$$y(x_n) = \begin{cases} < 0 & \text{if } t_n < -1 \\ > 0 & \text{if } t_n \geq 1 \end{cases} \quad (2.3)$$

Thus the binary classification is performed on each unknown input x by examining the value of $\text{sgn}(y(x))$ and assigning one class label to inputs that map to positive values and another class label to inputs that map to negative values.

Strengths – SVM training selects the model that maximally separates the classes. This should lead to lower misclassification rates. The properties of the SVM model and training procedures have been theoretically analyzed (Burges, 1998) and shown to have good generalization properties.

Weaknesses – SVM is an inherently binary classifier and multiple instances must be deployed serially or in parallel to perform multi-class classifications. The basic SVM model is linear and assumes that the classes are linearly separable. Non-linear class boundaries cause misclassification errors. Modifications to linear classifiers to handle such cases can be made by using non-linear basis functions.

Adaptive Update Mechanisms – SVM usually operates in batch mode. However, se-

quential versions of SVM are also available (W. Wang, Men, & Lu, 2008). The sequential approach starts with an initial set of data points and finds an optimal kernel transformation to apply to the SVM model given the current examples. Each new data point from the stream is checked for consistency with the current model utilizing a set of vectors orthogonal to the support vectors transformed by the current kernel. If the new point falls outside the bounds of the optimal model the optimization process is repeated otherwise the model remains unchanged. In a direct comparative study, in addition to learning in real-time, the online SVM approach showed a lower error rate than batch SVM (W. Wang et al., 2008). The MOA stream version of the SVM classifier, based on an optimized gradient solver (Shalev-Shwartz, Singer, & Srebro, 2007) known as Pegasos (PEG), was used for the experiments in this study.

Hoeffding trees

Basic Principle – Hoeffding Trees (HT), invented by Domingos and Hulten (2000) are a variety of decision tree classifiers and thus share the basic operating principles of decision trees. The classical decision tree learns a set of classification rules by analyzing example feature vectors and inducing rules based on the feature values that lead to correct decisions. The HT algorithm is named after the Hoeffding bound which is utilized to decide whether or not to split on a feature. This theoretical bound, used in HT, states that for n observations with a given range of values R , the difference between the sample mean and the true mean is no greater than ϵ where:

$$\epsilon = \sqrt{\frac{R^2 \ln\left(\frac{1}{\delta}\right)}{2n}} \quad (2.4)$$

The condition is guaranteed with probability: $(1 - \delta)$ (Domingos & Hulten, 2000).

Operational Method – During training of a classical decision tree, features that separate the classes are identified and used in the rules. The goal is to minimize the number

of decision nodes in the tree by using the most informative attributes to split on in terms of information gain. Information gain is defined by measuring the entropy of the classes discriminated by the splitting rule (Bifet & Kirkby, 2009). The entropy of a set of n partitions of the class labels is shown in Equation 2.5. The partition is given as a distribution of fractional values $(p_i, \{1 \leq i \leq n\})$ that sum to 1.

$$\sum_{i=1}^n -p_i \log_2(p_i) \tag{2.5}$$

Hoeffding Trees allow the decision tree to be constructed on the fly from examples examined one at a time. The inventors of the HT show that decision tree rules can be constructed using sufficient statistics instead of storing the examples themselves. For the HT algorithm the sufficient statistics are counts of the class label that co-occur with each attribute value. Thus, this classifier works most efficiently on attributes with a limited range of discrete values. The most informative feature is selected to split on among the top two candidates by choosing the one whose average information gain is greater within the ϵ bound given above.

Strengths – As with classical decision trees, an advantage of the HT is that the decision rules are explicit and provide the human user with a transparent explanation of the classification process. The conclusion justification can be vetted by the user to see if the decision process seems to be sensible or accidental. It can also expose features of the problem that may not have been readily apparent to the user. Another strength is that they operate efficiently. The average case complexity on n examples with m attributes is $O(mn \log(n))$ (Bifet & Kirkby, 2009).

Weaknesses – If deployed naively, the HT would try to reevaluate the splitting rules at every time step which incurs a computational cost. Additional methods are needed to decide when to incorporate new examples to adjust the rules. Exploring ways to effectively adjust the training interval is one of the prime goals of this study.

Adaptive Update Mechanisms – The incremental learning provided by Hoeffding trees has been extended as Hoeffding Adaptive Trees (HAT) to incorporate active detection of and response to concept drifts (Bifet & Gavalda, 2009). Adaptation depends on change detection which is provided by adding sentinels to each node in the Hoeffding Tree. The most straightforward sentinel is based on the ADWIN algorithm (Bifet & Gavalda, 2007). ADWIN monitors a window of recent examples and tests the hypothesis that sub windows do not significantly differ. If this null hypothesis is rejected then the older portion of the window is dropped. The adaptive trees show a lower memory footprint than comparable methods however there is a definite slowdown incurred by the adaptation (Bifet & Gavalda, 2009).

Adaptive Windows for Retraining Examples

An established approach to dealing with non-stationary concept spaces has been to apply techniques from statistical time series analysis (Gaber et al., 2005). This study uses an ensemble of such adaptive classifiers designed for concept-drifting data.

Adaptive correction begins with detection of the drift away from accuracy. Once the drift is detected, it is important to determine a set of recent examples that provide a representative sample to update the classification model. A common technique is to update the classifiers based on data collected within a window of time whose length depends on the current accuracy (Bifet et al., 2010). Another approach to time uses hierarchical structuring of time periods of different resolutions. This structure provides a mechanism to facilitate the efficient detection of drifts. The adaptive approach used to determining the update window of relevant examples is described next in Chapter 3.

Prequential Accuracy Analysis

Prequential accuracy analysis is the most commonly used method to evaluate streaming classifiers (Bifet et al., 2010). Each instance in the input stream is first used to test the classifier’s performance then as a new training example for the classifier. The windowed version used in this study sampled the current accuracy periodically and reported the percentage of instances correctly categorized by each algorithm over a sliding window of the past 10,000 examples classified.

Concept Drift Mitigation Strategies

The following set of concept drift correction algorithms were applied and compared at varying rates of drift to determine the relative efficiency of the approaches. Empirical findings are presented in Chapter 4 and implications and recommendations are discussed in Chapter 5.

Dynamic Weighted Majority (DWM)

This method was introduced by Kolter and Maloof (2007) in one of the first studies to specifically address concept drift in the context of ensembles of stream classifiers. The algorithm adaptively adjusts the weighted contribution of each member classifier referred to as a decision-making unit (DMU). A specification of the algorithm is given in Algorithm 2.0.1. The following is a key to the notation given in the algorithm listing.

$\{ \langle \vec{x}, L \rangle \}_n$ - A set of n training instances (pairs of feature vectors, \vec{x} and associated class labels, L);

$c \in \mathbb{N}$ - The number of classes, $c \geq 2$;

β - The factor for decreasing weights, $0 \leq \beta < 1$;

θ - The threshold for deleting DMUs;

p - The period between DMU removal, creation, and weight update;

$\{ \langle d, w \rangle \}_m$ - A set of m DMUs and their weights;

$\Lambda, \lambda \in 1, \dots, c$ - global and local class predictions;

$\vec{\sigma} \in \mathbb{R}^c$ - The sum of weighted predictions for each class.

Algorithm 2.0.1 Dynamic Weighted Majority (Kolter & Maloof, 2007)

```

1: procedure DWM( $\{\langle \vec{x}, L \rangle\}_n, c, \beta, \theta, p$ )
2:    $m \leftarrow 1$ 
3:    $d_m \leftarrow \text{createNewExpert}()$ 
4:    $w_m \leftarrow 1$ 
5:   for  $i \leftarrow 1, \dots, n$  do ▷ Loop over examples
6:      $\vec{\sigma} \leftarrow 0$ 
7:     for  $j \leftarrow 1, \dots, m$  do ▷ Loop over DMUs
8:        $\lambda \leftarrow \text{classify}(d_j, \vec{x}_i)$ 
9:       if  $\lambda \neq y_i$  and  $i \bmod p = 0$  then
10:         $w_j \leftarrow \beta w_j$ 
11:       end if
12:        $\sigma_\lambda \leftarrow \sigma_\lambda + w_j$ 
13:     end for
14:      $\Lambda \leftarrow \text{argmax}_j \sigma_j$ 
15:     if  $i \bmod p = 0$  then
16:        $w \leftarrow \text{normalizeWeights}(w)$ 
17:        $d, w \leftarrow \text{removeExperts}(d, w, \theta)$ 
18:       if  $\Lambda \neq y_i$  then
19:         $m \leftarrow m + 1$ 
20:         $d_m \leftarrow \text{createNewExpert}()$ 
21:         $w_m \leftarrow 1$ 
22:       end if
23:     end if
24:     for  $j \leftarrow 1, \dots, m$  do
25:        $d_j \leftarrow \text{train}(d_j, \vec{x}_i, y_i)$ 
26:     end for
27:     output  $\Lambda$ 
28:   end for
29: end procedure

```

The Accuracy Updated Ensemble Algorithm (AUE)

This method was introduced by Brzezinski and Stefanowski (2014) and aimed to be robust against various types of concept drift. A specification of the algorithm is given in Algorithm 2.0.2.

Algorithm 2.0.2 Accuracy Updated Ensemble (Brzezinski & Stefanowski, 2014)

Require: : S : data stream of examples partitioned into chunks, k : number of ensemble members, m : memory limit

Ensure: : E : ensemble of k weighted incremental classifiers

```
1:  $E \leftarrow \emptyset$ ;  
2: for all data chunks  $B_i \in S$  do  
3:    $C' \leftarrow$  new component classifier built on  $B_i$  ;  
4:    $w_{C'} \leftarrow \frac{1}{MSE_r + \epsilon}$ ;  
5:   for all classifiers  $C_j \in E$  do  
6:     Apply  $C_j$  on  $B_i$  to derive  $MSE_{ij}$ ;  
7:     Compute weight  $w_{ij}$ ,  $w_{ij} = \frac{1}{MSE_r + MSE_{ij} + \epsilon}$ ;  
8:   end for  
9:   if  $|E| < k$  then  
10:     $E \leftarrow E \cup C'$ ;  
11:   else  
12:    Substitute least accurate classifier in  $E$  with  $C'$ ;  
13:   end if  
14:   for all classifiers  $C_j \in E \setminus C'$  do  
15:    Incrementally train classifier  $C_j$  with  $B_i$  ;  
16:   end for  
17:   if  $\text{memoryUsage}(E) > m$  then  
18:    Prune (decrease size of) component classifiers;  
19:   end if  
20: end for
```

The Accuracy Weighted Ensemble Algorithm (AWE)

This method was introduced by Wang (H. Wang et al., 2003) and is designed to adapt to concept drift by re-weighting the ensemble based on the expected accuracy of its members. Expected accuracy in this algorithm can be looked at from two perspectives. The costs of the errors or the benefits of correct classifications. The costs based approach computes the mean squared error of the classifier in question (MSE_i) against the mean squared error of a classifier that gives random answers (MSE_r). These are computed as:

$$\begin{aligned}MSE_i &= \frac{1}{S_n} \sum_{(x,c)_n} (1 - f_c^i(x))^2 \\MSE_r &= \sum_c p(c)(1 - p(c))^2,\end{aligned}\tag{2.6}$$

where $p(c)$ is the probability of an input x being classified as class c . The benefits approach assumes that you have assigned a numerical benefit for each classification of class c as being class c' , denoted $b_{c,c'}$, and that we know the probability given by classifier C_i that input x belongs to class c , denoted as $f_c^i(x)$. The benefit is then computed as:

$$b_i = \sum_{(x,c)_n} \sum_{c'} b_{c,c'} \cdot f_c^i(x)\tag{2.7}$$

The full specification of the algorithm is given in Algorithm 2.0.3.

Algorithm 2.0.3 Accuracy Weighted Ensemble (H. Wang et al., 2003)

Require: : S : dataset of *ChunkSize* from the incoming stream, K : the number of ensemble members, ξ : number of bins, C : a set of K previously trained classifiers

Ensure: : C : a set of of K incremental classifiers with updated weights, μ, σ : mean and variance for each stage and each bin

- 1: train classifier C' from S
 - 2: compute error rate/benefits of C' via cross-validation on S
 - 3: derive weight w' for C' using $w_i = MSE_r - MSE_i$, as in equation 2.6 or $w_i = b_i - b_r$ as in equation 2.7
 - 4: **for all** $C_i \in C$ **do**
 - 5: apply C_i on S to derive MSE_i or b_i ;
 - 6: Compute weight w_i as above;
 - 7: **end for**
 - 8: $C \leftarrow K$ of the top weighted classifiers in $C \cup C'$;
 - 9: return C
-

Summary

This chapter provided an overview of the methods and algorithms that have been published in the literature for designing ensemble stream classifiers. Several mitigation strategies for concept drifts were described and selected for further evaluation in this study. The next chapter discusses the design and specification of the experimental framework used to comparatively evaluate the performance of these approaches.

Chapter 3

Methodology

Overview

This investigation was conducted in the form of an empirical study of the relationship between throughput and error rates in supervised data stream classifiers including adaptive ensembles that update their classification model in an attempt to maintain accuracy when the underlying patterns in the data change. Experiments were executed using the MOA framework (Bifet et al., 2010) as well as custom scripts for experiment management written in Python. Other software utilities were used in data analysis to derive statistics and generate graphics including Matlab, Microsoft PowerPoint and Microsoft Excel.

The MOA framework provides built-in functions for the deployment and evaluation of stream based classifiers. These include adaptation of bagging and boosting methods on ensembles of Bayesian classifiers (Bauer & Kohavi, 1999) and an adaptive online version of decision trees known as Hoeffding Adaptive Trees (HAT). In this study, 3 classification model types discussed in Chapter 2 – Naïve Bayes (NB), Pegasos Support Vector Machine (PEG), and Hoeffding Adaptive Tree (HAT) were used alone or in conjunction with the 3 adaptive ensemble methods to solve a binary classification problem on streams. The Adaptive Weighted Ensemble (AWE) was configured with five independent HAT members. The Adaptive Update Ensemble also comprised five HAT members. In data plots, the abbreviations AUE5 and AWE5 refer to these configurations.

Leveraging the diversity of individual classifier outputs is a key feature of the ensemble

method (Brown & Kuncheva, 2010) and using a heterogeneous set of classifiers made it possible to investigate independent trends in the evolving accuracy of each model type vs. the accuracy of the combined classification produced when ensembled. The Dynamic Weighted Majority (DWM) ensemble provided by MOA allows for the arbitrary specification of member classifier types. The DWM for this study was set up as a heterogeneous collection consisting of two HAT, two PEG, and one NB. In contrast, the MOA implementation of AWE and AUE only allowed ensembles of classifiers in a format compatible with Hoeffding Trees. Thus, those ensembles were assembled from five HAT classifiers.

There were four major trials executed on three separate machines. The specification of the experiment hardware is detailed below. In each major trial, The same data stream was presented to both the individual classifiers and the ensembles by using the same seed in the pseudorandom generator that creates the stream. As discussed above, the goal is to discover which algorithms produce accurate answers while preserving the maximum throughput. Developing and testing reliable and repeatable methodologies to evaluate and rank data stream classifiers was a major motivation for this research.

Experiment Framework

The availability of real-world test data is limited for stream analyses. The underlying nature of the data-in-motion problem requires that data be delivered to the classifier indefinitely. Thus, the data streams for this study were generated in real-time by a concept drift generation simulation framework. As discussed above, the freely available MOA software was designed to support such experiments. MOA library functions, with systematically varied parameters, were used to provide the stream data to be classified. In particular, the *generators.RandomRBFGeneratorDrift()* function of the MOA framework were used to create a stream of classification examples drawn from a mixed RBF distribution. A fixed number of centroids (using the MOA default of 50) are generated at random. Each centroid is randomly assigned to an initial class label. Examples are then generated by selecting a random direc-

tion and displacement of the example vector from the centroid. Concept drifts are induced in the model by moving the centroids away from their initial positions at a fixed drift rate specified by the *driftRate* parameter listed below.

The overall conceptual architecture of the experiment framework is shown in 3.1. Each component is then explained further below. In the simulation provided by the experiment framework, time is measured in discrete steps and each value of the input, true class label, and estimated class are indexed by the current time step t . The experiment start is denoted $t = 0$ and N is the time step at which the experiment is shut down. Individual features in the n -dimensional data vector, at time t , are denoted: $x_{it}; \{1 \leq i \leq n\}$.

The ground truth label is determined by the RBF Model. The estimated label produced for input x_t by the classifier, at time t , is denoted $C_t(x_t)$ and in the case of ensembles is the adjudicated decision derived from combining individual classifiers: $C_{it}(x_t); \{1 \leq i \leq 5\}$.

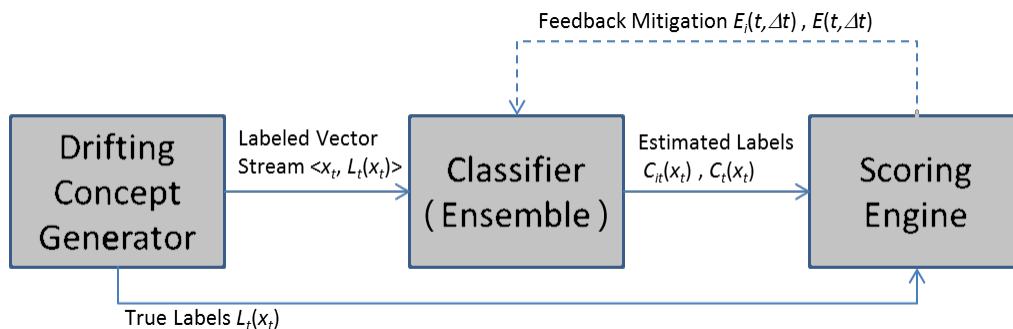


Figure 3.1: Experiment Framework Architecture

Drifting Concept Generator (DCG)

- Inputs – This is the first element in the pipeline and requires no input data.
- Parameters:
 - mSeed – The seed for the MOA random model generator
 - iSeed – The seed for the MOA random instance generator
 - numCentroids – The number of centroids (fixed to the default of 50)

- numAttributes - The dimensionality of the generated instances
- driftRate – The speed of centroid movement. Centroid positions are perturbed in a random direction at a rate specified by this parameter.
- Outputs – The DCG outputs a labeled stream of n -dimensional vectors. At each time step t , the DCG outputs the pair: $\langle x_t, L_t(x_t) \rangle$, where $L_t(x_t)$ is the class label from the set $\{\text{class1}, \text{class2}\}$ and x_t is the n -dimensional feature vector consisting of real-valued components $x_{it}; \{1 \leq i \leq n, 0 \leq t \leq N\}$.

The Drifting Concept Generator (DCG) component produces the labeled stream of data for the classification system to examine and classify. Input data were generated by using the MOA *generators.RandomRBFGeneratorDrift()* function to create a stream of random real-valued data vectors. The vectors at each instance are drawn from from an n -dimensional space clustered into normally distributed densities in hyperspherical regions around randomly positioned centroids. Each centroid g_ℓ has its own associated class label, ℓ , and standard deviation initialized when the experiment begins. The data dimensionality was set to 10 and 50 for different conditions as described below. For this study, the standard deviations of each class distribution were set to the framework’s default value. The random number generator seeds were set to a fixed, user-specified value during each major trial to allow for repeatability of particular experiment runs.

The DCG simulates the existence of categories in data by assigning labels to each vector. The n -dimensional vectors x_t are generated as a random displacement d_t from a randomly selected centroid $g_{\ell t}$, $L_t(g_{\ell t}) = \ell$, where ℓ is the concept label from the set $\{\text{class1}, \text{class2}\}$ that was assigned to the cluster. Thus, the stream instance at time t is given by Equation 3.1 while the label for this instance is the label of its parent cluster given by Equation 3.2.

$$x_t = g_t + d_t \tag{3.1}$$

$$L_t(x_t) = L_t(g_{\ell t}) \quad (3.2)$$

The output of this processing step is a pair $\langle x_t, L_t(x_t) \rangle$ representing a data point and its target class. At each time interval, $t = n\Delta t; \{n = 0, 1, 2, \dots\}$, the centroids change position with a speed set by the drift rate parameter. Three settings of driftRate were used across different trials (0.00, 0.001, and 0.010) to cover no, low, and high drift rate conditions respectively.

Classifier (Ensemble)

- Inputs – At each time step t , the Classifier or Classifier Ensemble ingests the data vector pair, $\langle x_t, L_t(x_t) \rangle$ produced by the DCG. If a classifier is in training mode, the target class label, $L_t(x_t)$, is used in the production of the classifier function. In test mode, the label is discarded and the data vector, x_t , is assigned an estimated class label. The MOA prequential evaluation framework manages interleaved test then train for each example in the stream.
- Parameters – This varies according to classifier type. The MOA experiment framework include the ability to store parameter settings as well as entire trained classifier models for repeatability of experiments. Other parameters include the individual and composite model-update intervals: $\Delta t_i; \{1 \leq i \leq 5\}$ and Δt_C , respectively.
- Outputs – At each time step, t , the ensemble classifier outputs: $C_{it}(x_t)$, the class label assigned to input x_t by classifier i , and $C_t(x_t)$, the class label assigned by the ensemble classifier to input x_t .

During training, at each time step t , each classifier in the ensemble was presented with a copy of the labeled vector as the pair: $\langle x_t, L_t(x_t) \rangle$. All classifiers are tested and trained continuously according to the prequential scheme described in Chapter 2.

Each classifier examined the generated inputs and produced an estimated label according to its specific algorithm as discussed in Chapter 2. The individual classifier functions: $C_{it}(x_t)$, where $\{1 \leq i \leq 5\}$, selected a classification label from the set $\{\text{class1}, \text{class2}\}$ for each input: x_t . In ensemble conditions, individual outputs were combined and adjudicated to create the composite response of the ensemble.

The ensemble classifier adjudicates the weighted average of the individual classifiers to select a classification label $C_t(x_t)$ from the set $\{\text{class1}, \text{class2}\}$ for each input: x_t . This corresponds to the outputs of the labeling function.

For DWM, AWE5, and AUE5, The contributions of the classifiers are adjusted over time according to the updating schemes described in Chapter 2.

Scoring Engine

The function of the scoring engine is included as part of MOA’s prequential analysis function.

- Inputs – At each time step t , the Scoring Engine receives: $C_{it}(x_t)$, the class label assigned to input x_t by classifier i , and $C_t(x_t)$, the class label assigned by the ensemble classifier to input x_t , and $L_t(x_t)$ the true class label for input x_t .
- Parameters – The scoring engine takes a parameter to determine the frequency of scoring/length of the scoring window as specified below.
- Outputs – The Scoring Engine outputs accuracy as the percentage of examples correctly classified by classifier. It also computes the κ statistic (described below), two variants of this statistic (κ_M , and κ_T), and metadata specific to each classifier such as ensemble member weights at the time of scoring.

During evaluation the classifier outputs in response to each input, $C_{it}(x_t)$, and the composite classification $C_t(x_t)$, was compared with $L_t(x_t)$. If the results are equal, it is considered a correct result. Otherwise, it is tallied as a missed classification. The counts in each category are summarized in a table, known as a “confusion matrix,” as shown in Table 3.1.

Predicted Class	Ground Truth	
	class1	class2
class1	A	B
class2	C	D

Table 3.1: Confusion Matrix a.k.a Contingency Table

The values in the main diagonal (A and D) are the number of correctly classified examples in the evaluation interval. The other values (B and C) are mis-classifications where C is the number of examples that should have been labeled “class1” that were labeled “class2,” and B is the number of examples that should have been labeled “class2” that were labeled “class1.” The kappa statistic is then computed from these counts as:

$$\begin{aligned}
 p_0 &= \left(\frac{(A + D)}{(A + B + C + D)} \right) \\
 \kappa &= \frac{p_0 - p_r}{1 - p_r},
 \end{aligned}
 \tag{3.3}$$

where p_r is the probability of a random classifier producing a correct response. The κ value ranges from [0,1] where 1 indicates the classifier is always correct and if the classifier is equivalent to the random classifier. For the selected classifiers in this study, κ was not consistently supported so the main measure used in comparison is simply the prequential accuracy p_0 . A brief comparison of κ and p_0 values is shown in a section of Chapter 4.

During prequential evaluation, the classifiers are continually updated, for ensemble methods (DWM, AWE5, AUE5) the contribution of the member classifiers is adapted based on the details of the specific algorithm described in the “Selected Supervised Classifiers” section of Chapter 2. These ensembles track error rates independently of the external score recorded in the experiment. Adaptive ensembles change member classifier parameters over time to mitigate performance degradation as described in the next subsection.

Adaptive Mitigation Strategies

This portion of the framework is shown as “Feedback Mitigation” in Figure 3.1. The concept drift mitigation strategy is specific to each algorithm as described in Chapter 2 and is recounted in the summary list below.

1. Dynamic Weighted Majority (DWM) (Kolter & Maloof, 2007): If an individual classifier has fallen below its acceptable accuracy rate its contribution to the ensemble is down-weighted. It can also be excluded from the ensemble entirely (equivalent to a zero weight). The classifier is then updated and re-evaluated at the next test interval. If the retrained classifier would result in improved performance, it is included in the ensemble at a higher weight again.
2. The Accuracy Updated Ensemble Algorithm (AUE) (Brzezinski & Stefanowski, 2014): This approach is designed to maximize the performance of incremental learners. It dispatches the various ensemble members across “chunks”, i.e. subsets of the data stream that have been partitioned for examination by individual classifiers. As in the (DWM) method above, accuracy is monitored for degradation. The least accurate classifier is substituted out for a new classifier trained on a more recent chunk. The number of classifiers in the ensemble is also constrained by available memory and a user-specified memory limit.
3. Accuracy Weighted Ensemble (AWE) (H. Wang et al., 2003): This approach computes an estimate of the reliability of its members in terms of the cost of its current-error rate or the benefit of its correct classifications. The weights of each classifier’s vote in the final ensemble decision are adjusted on this basis with the high-performing classifiers contributing more to the final combined classification result.

The effect of running these mitigations on throughput rates was determined by comparing the average classification rate of each algorithm to the speed at which the Drifting Concept

Generator is able to supply instances to be classified. This represents a percentage of real-time processing, where 100% indicates that items are being processed as fast as they arrive. This can be used as a simple measure of the expected slow-down in system processing when this algorithm is applied to the input stream.

Experiments and Result Formats

Since machine learning and pattern recognition libraries such as MOA are readily available, and the concept-drift strategies were selected from methods supported in the literature as discussed in Chapter 2, the implementation of the classification and ensemble algorithms used in the experiment framework did not need to proceed from scratch. However, the deployment and testing of the component algorithms and creating an overall framework to evaluate the set of algorithms under controlled conditions was a non-trivial task and thus contributed to the overall complexity of the research effort. Results were collected in text-based form wherever possible to allow for human inspection and further analysis with a wide range of tools.

The classifiers varied widely in their intrinsic speeds and accuracies. Thus it was important to establish a standard range of parameters which were feasible to execute for all candidates in the investigation. Prior to the collection of performance data, many pilot studies were conducted to develop and test the overall framework and establish reasonable parameters for the free-parameters described above. The outcomes of the pilot studies are briefly described below. Outcomes of the main experiment and analysis are then discussed thoroughly in Chapters 4 and 5. Additional figures, results, and source code are provided in appendices.

Pilot Study Key Outcomes

This experiment phase was used to tune free parameters of the model-based input generator such as the appropriate number of attributes per instance and drift speed to make the major trials feasible on the available hardware. This phase was also used to test the experiment framework for bugs and create any practical experimental scripts needed for smooth operation and data collection in subsequent steps. The chief output of this first phase was an experiment control script that can run experiments under various conditions and parameters required for the rest of the study with minimal low-level re-programming.

Next, general performance characteristics such as run-time and accuracy on small sets of inputs were used to determine the fixed parameters that would be used for the rest of the data collection. The most important output of this second phase was a set of performance bounds and delimitations for setting the conditions and parameters required to collect the data needed to answer the central research questions of this study.

The stream length was standardized to 10,000,000 instances per trial. Two levels of dimensionality (`numAttributes`) were selected for the instances - 10 and 50. Dimensionality greater than 50 resulted in unacceptable run times. The 50-dimensional task resulted in runtimes ranging from less than 30 to over 150 minutes to execute the 10^8 classifications. This was considered the limit for feasibly repeating the conditions while varying other conditions such as the drift rate. Three drift rates were selected. The control case was no drift with the `driftRate` parameter set to 0. A drift rate of 0.001 was selected as the “low drift” condition since it was the minimal drift suggested by the MOA GUI built-in function. A `driftRate` of 0.010 was selected as “high drift” since, in the pilot studies, drift rates at this level or higher invariably reduced the classifier’s accuracy to the level of chance (50%).

With these experimental parameters established, the main data collection was executed to obtain data points across the combinatorial gamut of the values selected.

Main Experiment – Performance Envelopes Under Concept Drift Conditions

This experiment phase was the culmination of the study in which the trade-off of accuracy and throughput created by the corrective action of adaptive mitigations is analyzed in detail.

Raw data was streamed to standard output by the MOA framework and collected in comma-separated values (CSV) files. These raw data files were subsequently filtered and processed in Excel to generate summary statistics of the relevant output columns. Further analysis and visualization was achieved using Matlab. The output of analysis in this phase was a characterization of each candidate strategy for concept drift mitigation against the baselines of maximal theoretic accuracy and throughput. The trade-off relationships are presented as performance envelopes - a joint plot of throughput vs. accuracy. The results of this empirical analysis will provide the means for system designers to select an appropriate methodology based on the accuracy and throughput available to that approach. The differences observed in the time-series of each classifier algorithm's accuracy is also shown as a series plots of p_0 at each prequential sample point. In Chapter 4, the findings across the various experimental conditions are discussed quantitatively. The implications of these results and further factors to consider when choosing one algorithm over another are discussed in Chapter 5.

Experiment Resources

The experiments described in this report require only computer hardware and software resources. No data collected directly from human subjects were utilized in this study. The data sources were produced via real-time simulation using open-source tools. Thus, no licensing or data ownership/privacy issues are of concern.

Four major trials were conducted on a two personal workstations and one “gaming level” laptop. The CPU and memory resources used are as follows:

- Computer 1 (PC Workstation):
 - Windows 10
 - 8 CPU threads, Intel Core i7 920 @ 2.67 GHz
 - 20.0 GB DDR3 RAM

- Computer 2 (PC Gaming Laptop):
 - Fedora (Scientific Spin) Linux 25
 - 8 CPU threads, Intel Core i7-3630QM @2.4 GHz
 - 8.0 GB DDR3 RAM

- Computer 3 (Mac Pro Workstation):
 - MacOS 10.12
 - 12 CPU threads, 6-Core Intel Xeon E5 @3.5 GHz
 - 16.0 GB DDR3 RAM

Experiments were managed and conducted using Python scripts and Java executables on Windows and Linux. Additional software resources include integrated development environments and tools including Eclipse, Microsoft Excel, Microsoft Paint3D, and Matlab. The MOA framework and associated WEKA framework were used for the implementation and

execution of published pattern-recognition algorithms such as the binary stream classifiers described in this chapter and Chapter 2. The software products used for this research were provided under academic license agreements or available as open-source packages from the Internet.

This study was not designed to investigate distributed computing optimizations. Thus, networking resources are not considered essential to the operation or evaluation of these experiments. However, multiprocessor computing techniques may be applied to improve the execution of similar studies in the future.

Chapter 4

Results

Overview

The experiments performed in this study consisted of a raw-data collection phase followed by several stages of analysis in which the raw data was cross-compared to investigate the effects of the manipulated experiment parameters (dimensionality, and drift rate) on the measured performance-related variables (accuracy, throughput). The mean accuracy of a trial was computed from the 1000 prequential sample measurements generated during the experiment. The raw throughput is obtained by taking the total CPU seconds reported for a run and dividing by the instance count of 10,000,000. Further analysis and perspectives on the data are detailed below in discussions accompanying the data plots. Larger versions of the plots are included in Appendix C.

Performance of Selected Algorithms on Concept-Drifting Data Streams

Performance on a Single Data Stream

The first set of results shows the performance of all six classifier algorithms during a single trial - Trial B. Figure 4.1 shows Trial B's mean accuracy on 10-Dimensional instances.

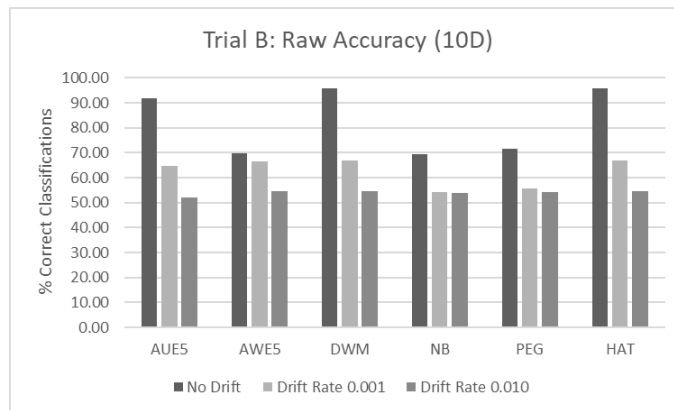


Figure 4.1: Trial B Accuracy on 10-Dimensional Instances

The first apparent trend is that accuracy rates consistently decrease as drift rate increases. The top performers under conditions of no drift are DWM, HAT, and AUE5 with accuracy in the 90% range. The remaining classifiers only manage to score around 70%. Under high drift (0.010) conditions all the classifiers are basically degraded to performing at chance level (50%). At the (0.001) drift levels the classifiers are all impacted but the effect on the high performers is more dramatic since the lower performers were not very accurate to begin with.

Figure 4.2 shows the same measurements for the 50-Dimensional case. Interestingly, there is a systematic increase in the accuracy of many of the classifiers in the higher dimensional space. These algorithms are all binary classifiers and thus the nature of the problem is fundamentally one of discriminating and sorting input instances into one of two categories. Even though the patterns generated in the 50-dimensional problem are numerically more complex, the algorithms now have more features to work with. Thus the overall ability

of each algorithm to find and use information in the features to correctly discriminate the target classes is improved. Because of this overall improvement in performance, the dynamic range of the concept drift effect is also increased. The systems performance is systematically reduced but many are able to achieve at least 80% accuracy and approach their performance in the non-drifting conditions.

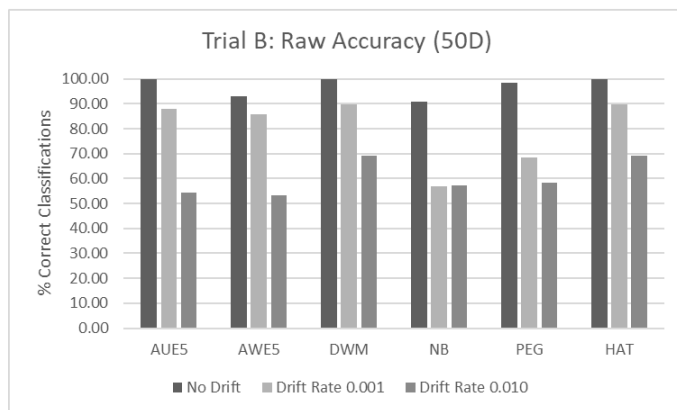


Figure 4.2: Trial B Accuracy on 50-Dimensional Instances

Because of this effect of dimensionality on classifier accuracy, one way to focus more closely on the effect of concept drift is to examine the relative reduction in a classifier’s accuracy with respect to its maximum theoretical performance on this classification problem. If we consider the no-drift condition as a baseline for each classifier then the drift conditions result in some percentage reduction of each classifier’s accuracy. Figure 4.3 displays this scaled version of accuracy on 10-dimensional instances. With this view we can see that AWE5 is the most robust in that it has the least relative reduction under the drift conditions. The NB and PEG classifiers appear to be doing well under this metric. However, it is important to note that this does not indicate that they are high-performing since their no-drift accuracy was not good to begin with (70%).

Figure 4.4 repeats this for the 50-dimensional inputs. As in the other view, the dynamic range of the drift effect is more apparent. We see that NB and PEG actually are not as

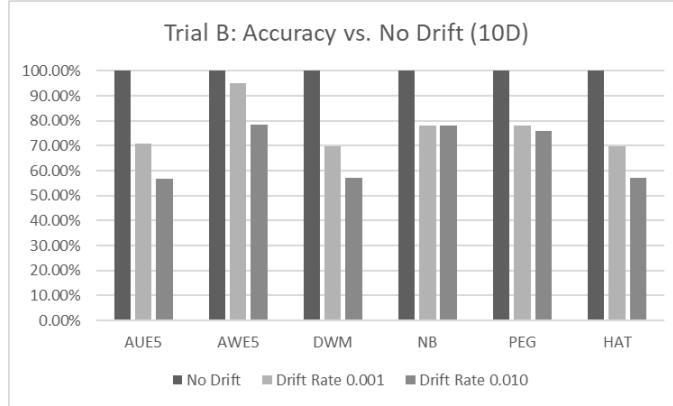


Figure 4.3: Trial B Relative Accuracy on 10-Dimensional Instances

robust to drift as the adaptive classifiers. Under both 10-dimensional and 50-dimensional conditions, the performance of the DWM and the HAT seem similar. This is an indicator that the HAT members of the DWM are probably more influential than the others which makes sense given that NB and PEG are the weaker classifiers according to what we have reviewed to this point.

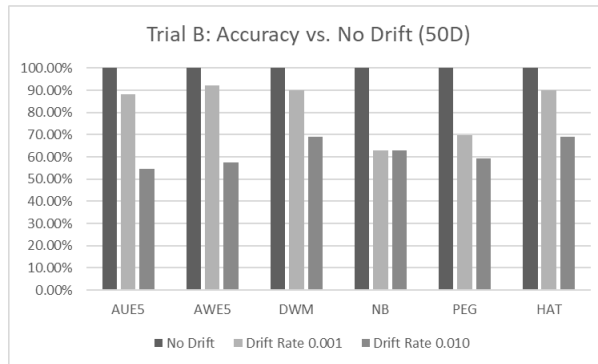


Figure 4.4: Trial B Relative Accuracy on 50-Dimensional Instances

The next measure of interest, throughput, is shown for Trial B in the following charts. First, the raw throughput in instances classified per second is shown for the 10-dimensional case in Figure 4.5

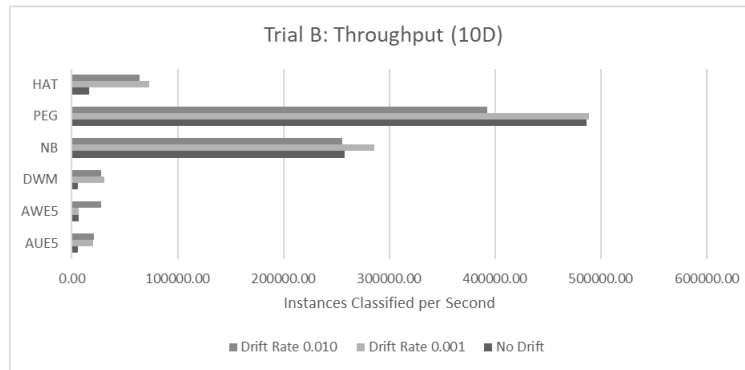


Figure 4.5: Trial B Throughput on 10-Dimensional Instances

Figure 4.6, shows the 50-dimensional raw throughput. Note that the scales in these images are very different. The complexity of generating 50 attributes-per-instance slows down classification considerably however the relative performance of the algorithms is approximately the same with PEG and NB being the fastest by far.

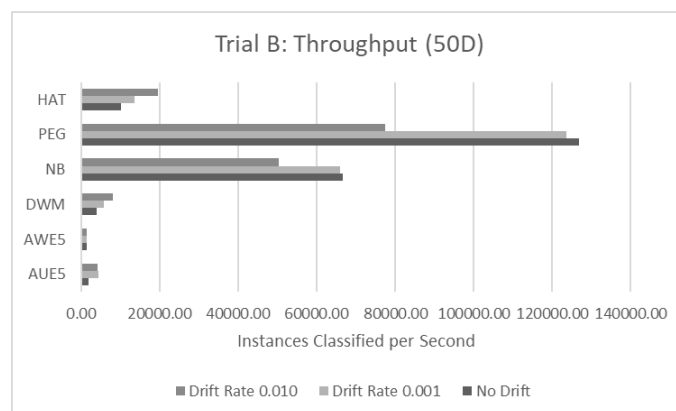


Figure 4.6: Trial B Throughput on 50-Dimensional Instances

With this result in mind, an important factor to consider when evaluating the throughput is that the stream generator rate of instance production must be accounted for since the classifiers can only operate as fast as the generator presents them with instances to classify.

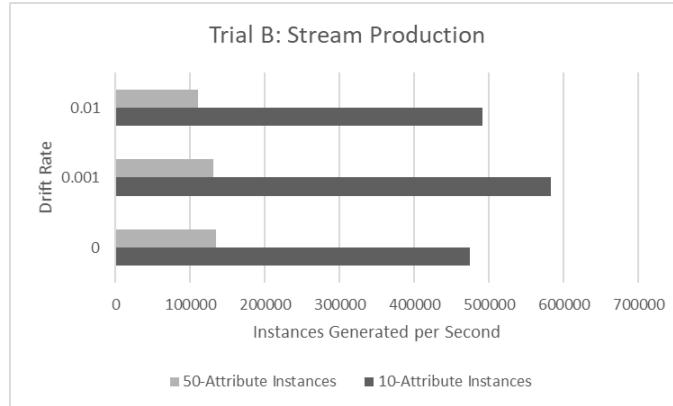


Figure 4.7: Trial B Stream Instance Generation Rate

Figure 4.7 shows the stream generation rate for Trial B as determined by the benchmarking software described in the previous chapter. Using these results the relative throughput for each classifier was computed and is shown in Figures 4.8 and 4.9.

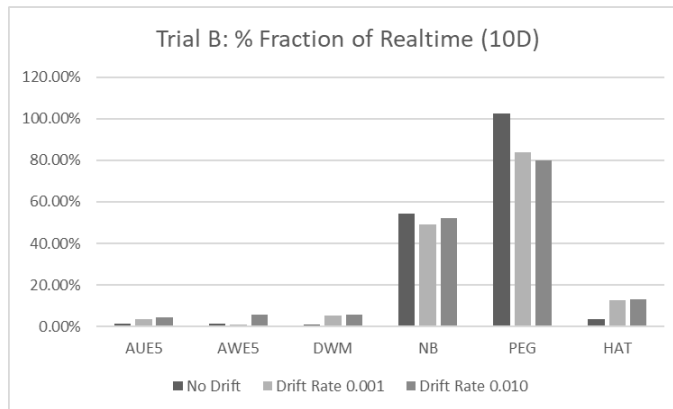


Figure 4.8: Trial B Scaled Throughput on 10-Dimensional Instances

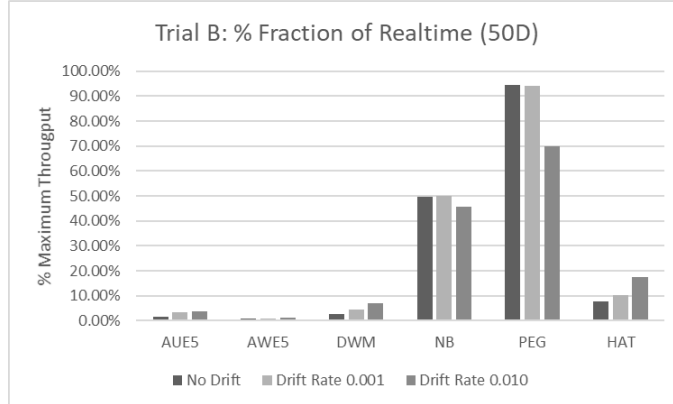


Figure 4.9: Trial B Scaled Throughput on 50-Dimensional Instances

A score of 100% on these evaluations indicates that the classifier can produce answers as fast as it receives instances.

An interesting result of this analysis is that some of the classifiers (AUE5, DWM, and HAT) showed an unexpected increase in throughput under drift conditions. While the stream generator also showed an increase in throughput during drift conditions, this only happened for the 10-dimensional case and was not monotonic. The 50-dimensional case actually shows a decreasing rate of instance production but the same classifiers still exhibited a systematic increase in throughput under drift. Also, as described above, the classifier throughput shown is computed *relative to this rate* and shown as a percentage of the theoretical maximum (real-time) processing speed. Thus, the effect is not explained by fluctuations in the stream generation.

A possible reason for the increased throughput of these particular classifiers is indicated by the fact that the algorithms explicitly designed for drift correction show more of the effect. The specific strategies for drift mitigation are different but they all have mechanisms to either remove or down-weight low performing members of the ensemble.

The pruning of poorly performing models would result in speed up since fewer computations would be executed per classification event. The HAT was not an adaptive ensemble

but is an adaptive algorithm nonetheless and also prunes the decision tree as the experiment proceeds resulting in a simplified model that would also account for the speed up in classification throughput.

Average performance across all streams

In this section we examine the performance metrics as averaged across all trials. Note that the coefficients of variance for the accuracy measures for all the algorithms across all major trials on the various platforms were low as shown in figure 4.10.

CV Accuracy (All Trials)				
Drift Rate				Input Attributes
Algorithm	0	0.001	0.010	10
AUE5	0.13	0.01	0.01	
AWE5	0.07	0.01	0.02	
DWM	0.01	0.01	0.01	
NB	0.08	0.02	0.02	
PEG	0.06	0.02	0.04	
HAT	0.01	0.01	0.01	
Algorithm	0	0.001	0.010	
AUE5	0.05	0.00	0.02	
AWE5	0.03	0.00	0.02	
DWM	0.00	0.01	0.01	
NB	0.03	0.05	0.05	
PEG	0.01	0.00	0.03	
HAT	0.00	0.01	0.01	

Figure 4.10: Coefficient of Variance (CV) of Accuracy across Experiment Trials

Thus, the machine used to run the trial, and the random seed did not have a significant effect. The mean accuracy across all trials is expected to be representative of true performance regardless of the different hardware used to collect the data. For the rest of the discussion in this chapter, “accuracy” will refer to this across-trial mean. Examining the averaged results over repeated trials across various platforms and data streams serves to verify the trends discussed above.

Figure 4.11 shows the accuracy result for the 10-dimensional classification problem. In the 10-dimensional case AUE, DWM and HAT consistently dominate as long as there is no

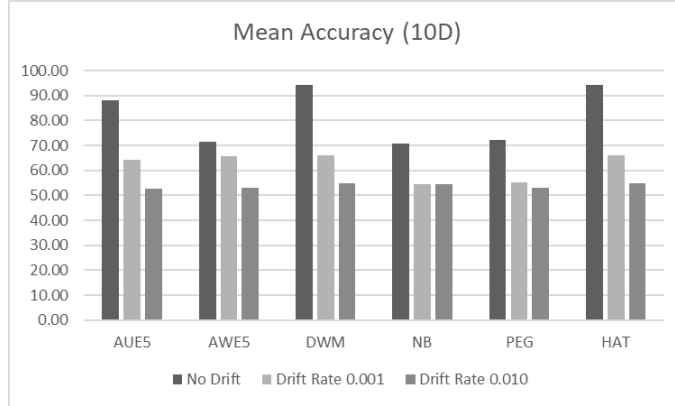


Figure 4.11: Mean Accuracy on 10-Dimensional Instances

drift. Again, even moderate drifts severely impact the performance of all of these algorithms on this classification problem.

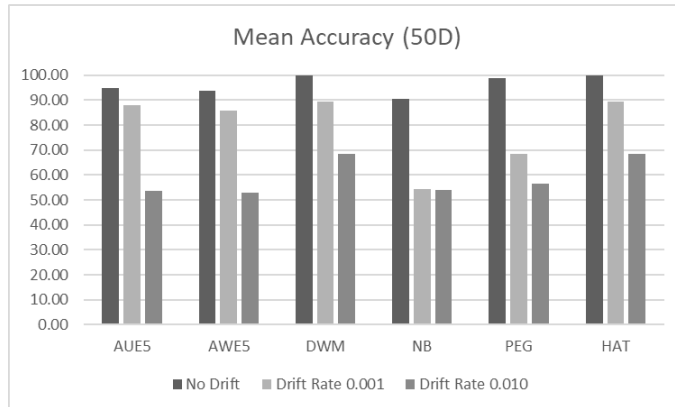


Figure 4.12: Mean Accuracy on 50-Dimensional Instances

Figure 4.12 shows the accuracy results for the 50-dimensional case. The trend of increased accuracy in the higher-dimensional case also showed to be consistent. The collective data also verified that NB was overall the least robust to drift. This is expected since although the Bayesian classifier, by its nature, constantly updates the classification model based on

the new stream examples, it is not specifically designed to counteract concept drift like the adaptive ensembles.

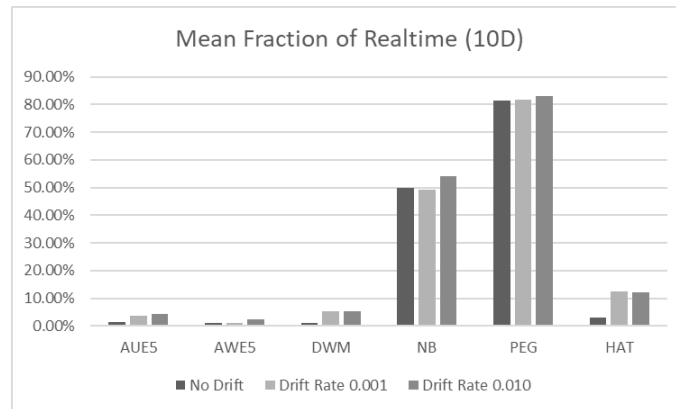


Figure 4.13: Mean Throughput as a Fraction of Real-Time Processing (10D)

Figure 4.13 shows the average relative throughput results for the 10-dimensional case. The NB and PEG are consistently faster and there is still an apparent speed up in some of the classifiers under drift conditions as discussed above.

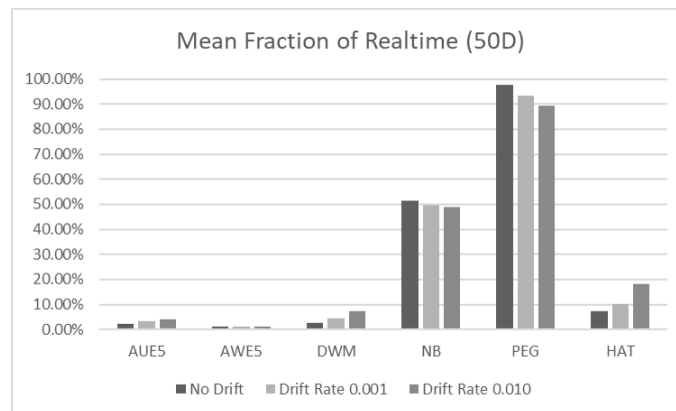


Figure 4.14: Mean Throughput as a Fraction of Real-Time Processing (50D)

Figure 4.14 shows the average relative throughput results for the 50-dimensional case. The pattern is similar to the 10 dimensional case. NB and PEG still dramatically dominate in terms of throughput. Because of this, it is hard to see the detailed performance of the

other classifiers. The next figures remedy this by excluding PEG and NB while zooming in to show the others.

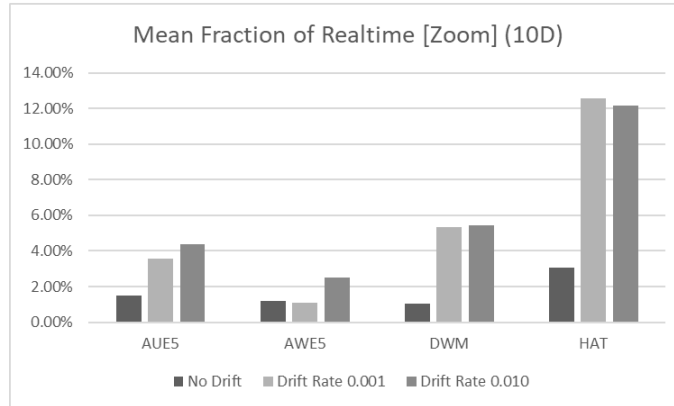


Figure 4.15: Mean Throughput of slower algorithms (10D)

Figure 4.15 shows the average relative throughput results for the 10-dimensional case on the selected classifiers.

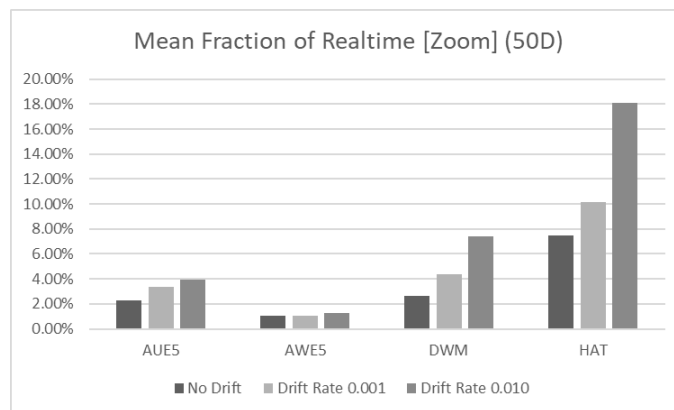


Figure 4.16: Mean Throughput of slower algorithms (50D)

Figure 4.16 shows the average relative throughput results for the 10-dimensional case on the selected classifiers.

In general, the speedup of adaptive classifiers under drift conditions is still apparent but not consistent. The change in model complexity due to pruning while adapting in response to challenge from the concept drift is still the prime suspect as discussed above.

The Eponymous Performance Envelopes

These figures summarize the performance of each classifier by rendering them as points in a two-dimensional space. Note that the accuracy axis starts at 50% since any performance at or below that value is considered a complete failure - no better than chance. The line of 50% throughput and 75% correct classification rate are highlighted to divide the space into quadrants. Classifiers which fall in the upper right quadrant are most desirable since they are both fast and accurate. Conversely the lower left quadrant indicates the poorest performance from a classifier algorithm.

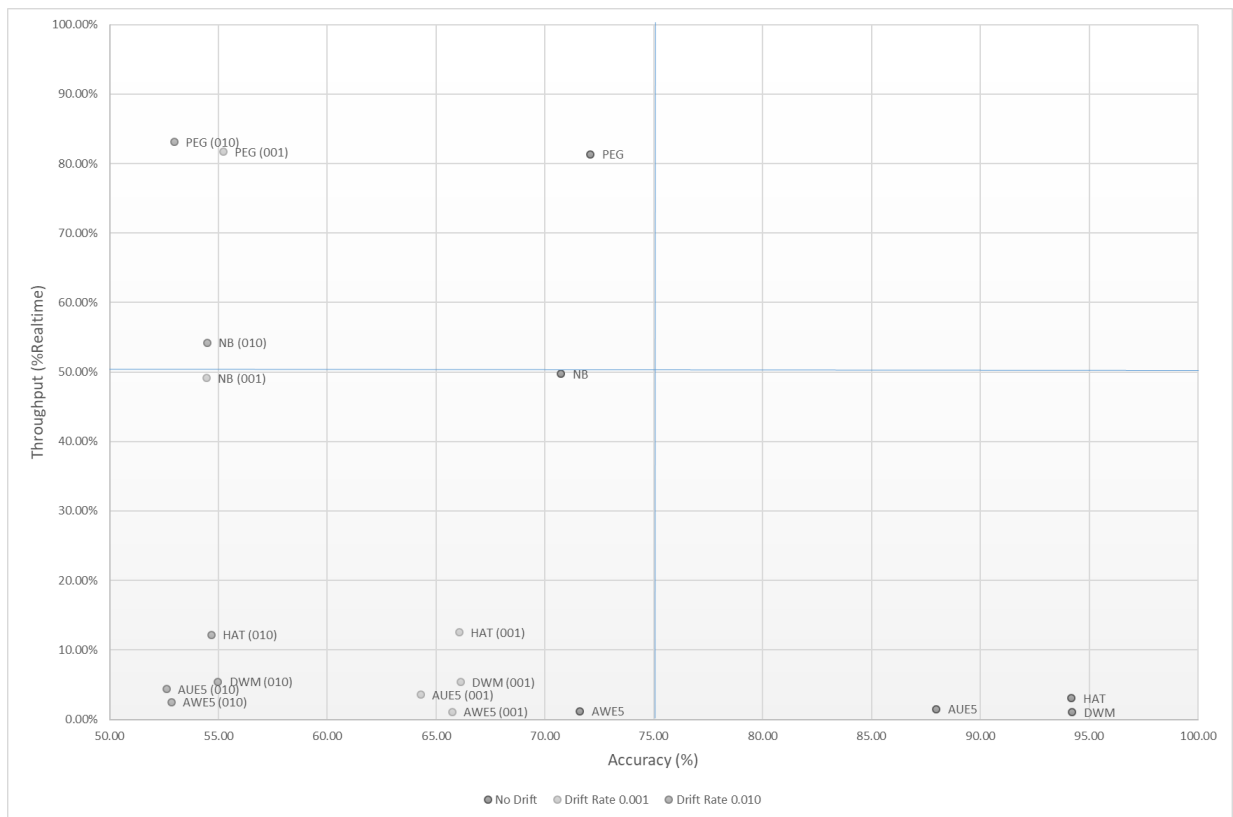


Figure 4.17: Accuracy vs. Throughput (10D)

Figure 4.17 shows the performance summary diagram for the 10-dimensional case. The bold lines aid in visually ranking the candidate classifiers. The trends discussed above such as the relatively high accuracy of HAT, the throughput costs of ensemble methods, and the sensitivity of all algorithms to concept drift are all apparent at-a-glance in this visualization. More importantly, The four quadrants of the diagram provide a visual classification scheme to rank the candidates in terms of the performance measures of interest. The results of the PEG and NB classifiers tend towards the upper left indicating that they are fast but not very accurate. Adding drift only makes things worse for these two algorithms. HAT and DWM appear on the lower right, along with AUE5. These are clear examples of a speed-accuracy trade-off. At throughputs of less than 10% real-time their accuracy might not be worth the cost in speed. These would not work for a time-sensitive system. The effect of drifts is clear in this rendering. All of the algorithms' scores are shifted left and most to the lower-left in drift conditions indicating that even the adaptive ensembles could not recover acceptable performance on this problem under even mild drift conditions.

Figure 4.18 shows the performance summary diagram for the 50 dimensional case. As was apparent in the results discussed above, the classifiers performed differently on the higher-dimensional problem. The results of the PEG and NB classifiers are notably improved in the non-drift conditions. Adding drift notably damages the PEG accuracy but does not reduce throughput very much. The NB classifier is rendered practically unusable by any drift. As is indicated by the cluster of results on the lower right, the adaptive ensembles do well to maintain performance under the (0.001) drift conditions. However, a drift rate of (0.010) pushes all of the algorithms performance ratings into the lower-left quadrant showing that they become unusable at this level of concept drift.

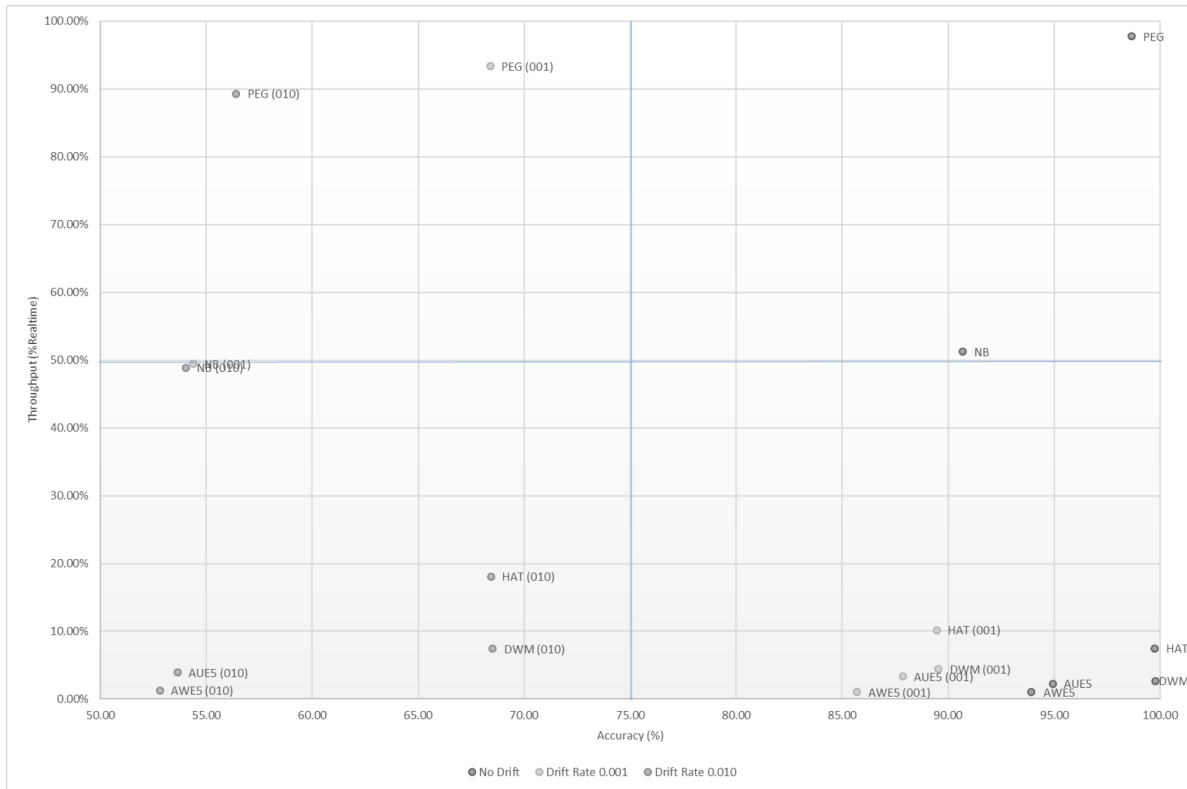


Figure 4.18: Accuracy vs. Throughput (50D)

Time-series Characteristics of Classifier Error-rates

Additional insights into classifier performance can be gained by examining the time course of their error rates.

Looking at the performance of Pegasos SVM over time in figures 4.19 and 4.20, the damaging effect of drift on accuracy is clearly apparent. Pegasos has no active drift correction and so, especially in figure 4.20, systematically decaying accuracy can be noted in response to the continuous drift provided by the drift data generator until its average performance is not much better than a uniformly random selection of one of the two classes (50%).

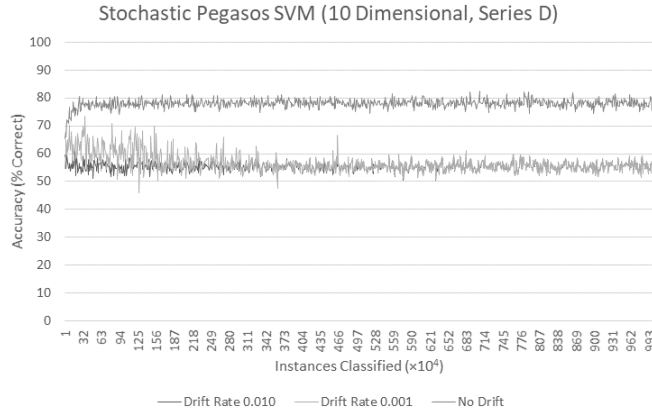


Figure 4.19: Pegasos SVM Time Series (10D)

In Figures 4.21 and 4.22, there is a more constant performance level in the evolution of the error rate in the NB classifier at 10 and 50 dimensions respectively.

While the Bayesian classifier does not have explicit drift correction, it is adaptive by nature in its operation in that it is constantly updating its model based on the latest evidence whether there is a drift or not. This is further evident in the 50 dimensional case where we see wider oscillations in the drift cases since the model is struggling to keep up with the shifting ground truth.

Figures 4.23 and 4.24 show the time courses of HAT, the first of the classifiers with explicit drift correction.

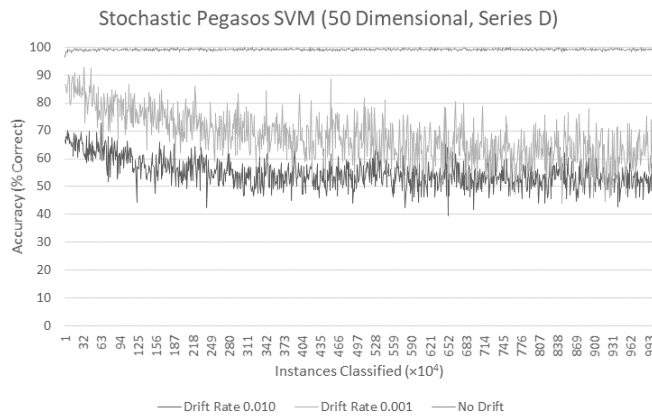


Figure 4.20: Pegasos SVM Time Series (50D)

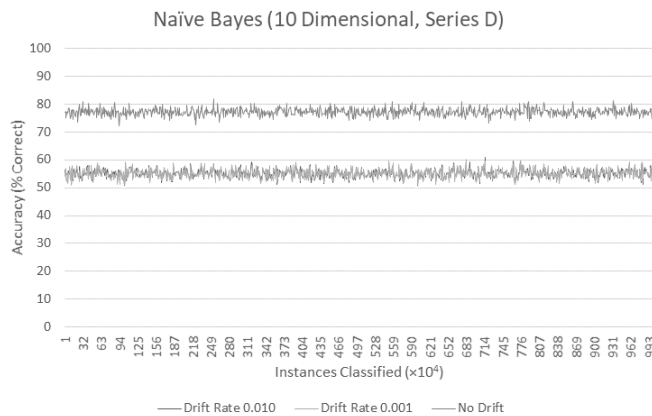


Figure 4.21: Naive Bayes Time Series (10D)

As in the NB case, the accuracy of HAT does not decay past a certain point in each of the drifts. However, it is never corrected back to its non-drift level especially when the drift becomes high.

We now turn to DWM, the first of the ensemble methods. In the figures 4.25 and 4.26,

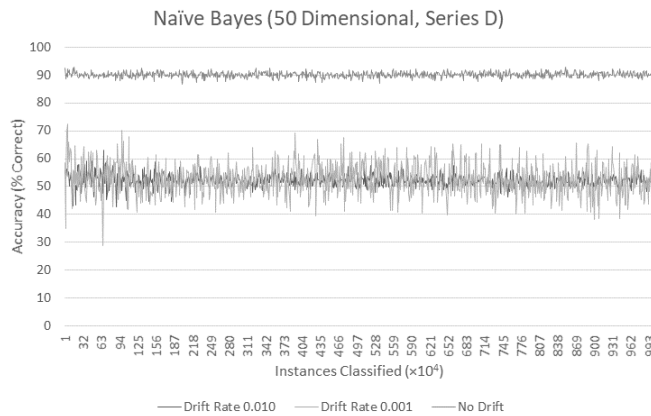


Figure 4.22: Naive Bayes Time Series (50D)

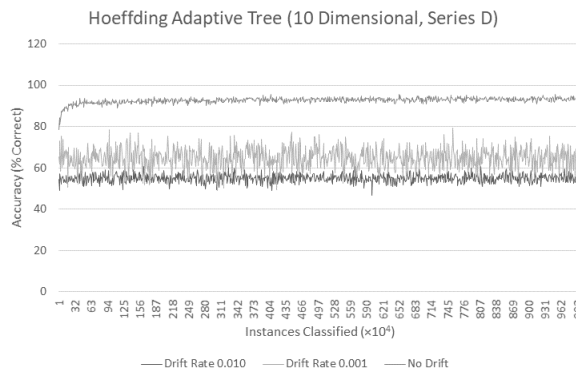


Figure 4.23: Hoeffding Adaptive Tree Time Series (10D)

we see much the same effect as in the single HAT. Accuracy measures under drift conditions are prevented from decaying and remain in a fairly acceptable range but never achieve the levels seen with no drift.

The ensembles that are directly related to the estimated accuracy of their members, shown in 4.27 through 4.30, exhibit the most robust drift correction. In the case of AUE,

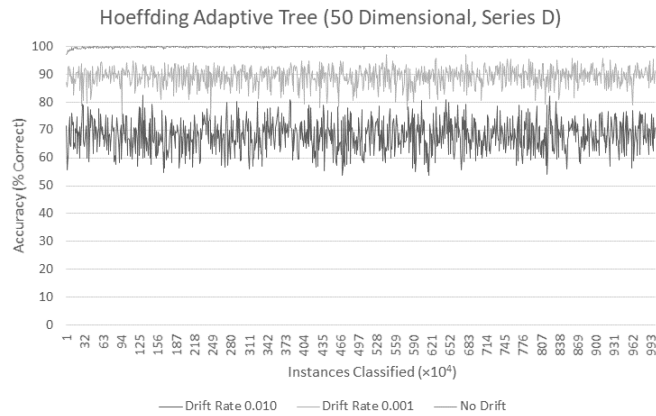


Figure 4.24: Hoeffding Adaptive Tree Time Series (50D)

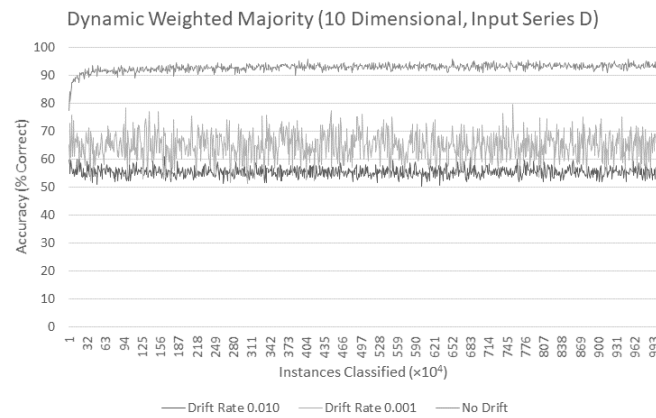


Figure 4.25: Dynamic Weighted Majority Time Series (10D)

the performance under drift correction is close to the non-drift level.

For the 10 dimensional case shown in 4.27, the AWE classifier only performs at around 80% accuracy even without drift. The effect of concept drift, as seen in the previous sections, is a systematic reduction in average accuracy. The time-series for drift conditions in AWE

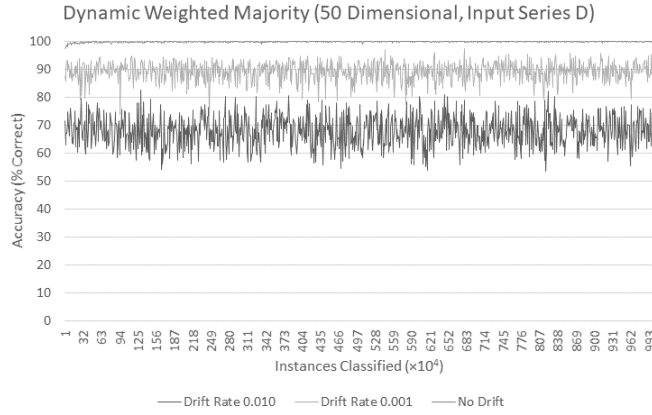


Figure 4.26: Dynamic Weighted Majority Time Series (50D)

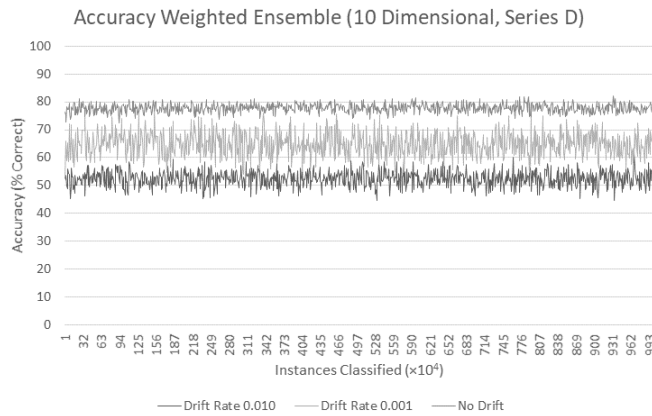


Figure 4.27: Accuracy Weighted Ensemble Time Series (10D)

also show that, in general, the trend in each series is a deflection about a certain constant level instead of a steady decay as in PEG. Thus the AWE algorithm is attempting to correct itself over time but not necessarily succeeding especially when the drift becomes high (0.010).

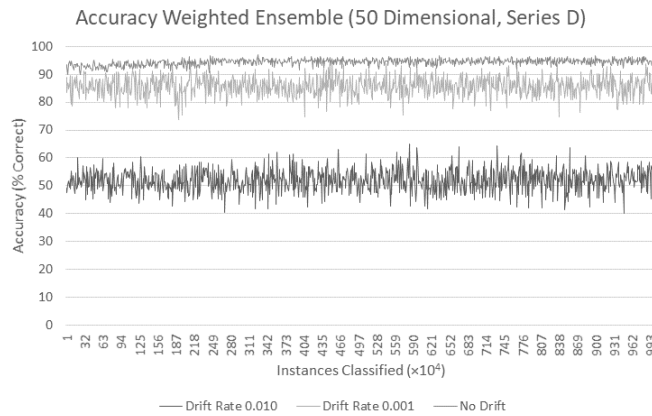


Figure 4.28: Accuracy Weighted Ensemble Time Series (50D)

In the 50 dimensional case shown in 4.28, the AWE classifier accuracy is much higher without drift. The AWE algorithm is also able to compensate for a (0.001) drift rate over time but is still reduced to unacceptable levels when the drift rate reaches (0.010).

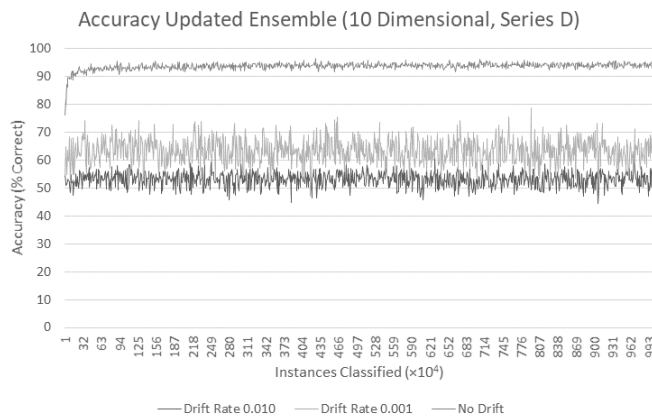


Figure 4.29: Accuracy Updated Ensemble Time Series (10D)

Figure 4.29 shows the time-series for AUE in 10 dimensions. It shows a similar time

evolution to AWE in that the performance under drift conditions is kept from decaying but not improved back to the levels seen without drift.

Figure 4.30 shows the 50-dimensional time-series for AUE. Of all the classifiers, this one is best able to correct the accuracy in (0.001) drift. The high-drift condition still overwhelms the ability of the correction mechanism to mitigate the reduction in both instantaneous and average accuracy.

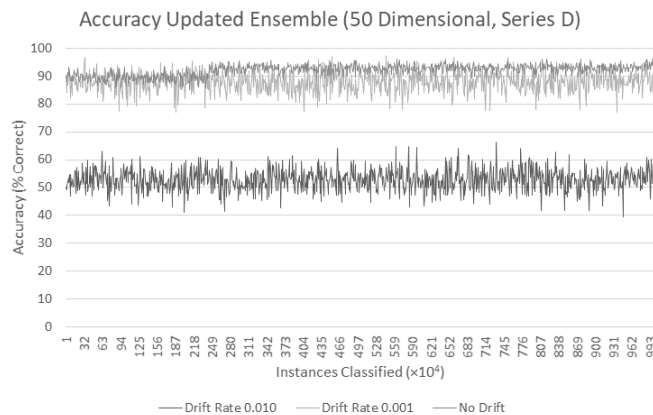


Figure 4.30: Accuracy Updated Ensemble Time Series (50D)

Kappa Statistics vs. Raw Accuracy

The MOA documentation (Bifet & Kirkby, 2009) suggests that p_0 may not reflect the true accuracy of streams since as the stream evolves, class imbalances may be present that are induced by the random order of presentation. The κ statistic is suggested as a less biased alternative, and MOA’s prequential evaluation function computes κ for classification algorithms that support it. Since the computation of κ was not uniformly supported in the classifiers selected for this study, p_0 was used in this research with the understanding that class imbalances could introduce a bias. This bias is expected to be slight since the simulated data is configured to produce balanced classes as discussed in Chapter 3.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

Results of all experiments consistently showed that increasing levels of concept drift and input dimensionality had adverse effects on performance. The various algorithms responded differently in both accuracy and throughput modulation. Some approaches were clearly better than others in terms of the trade-off between speed and accuracy. However, the average values displayed on the Accuracy-Throughput performance envelope plots are not indicative of the detailed adaptive response of the classifiers. The single point summaries do not give insight into the temporal dynamics of increases in error rate over time and the speed and degree to which the adaptive mechanisms compensate for the concept drift.

The robustness of each classifier's dynamic response to concept drift is another dimension on which the algorithms can and should be compared when making a selection. This is best assessed by an examination of the time-course of the evolving error-rate as in the latter figures in Chapter 4. When comparing against a standard input stream, spectral analysis can also be used as shown in Figures 5.1 and 5.2 to characterize different algorithms' temporal response to the shifting class definitions.

The similarity between stream classifiers' performance can also be compared quantitatively by examining correlations in their error rates over time as shown in Figure 5.3. While the high correlation observed between ensemble classifiers and their member algorithms can be expected, there are a few other interesting observations to note. The AUE and AWE show

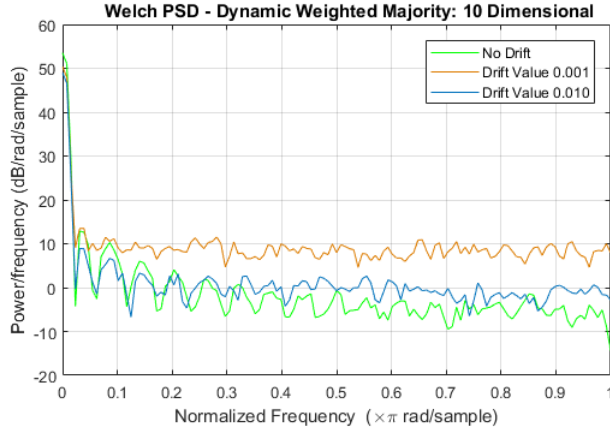


Figure 5.1: Welch PSD for DWM, 10 Dimensional input stream

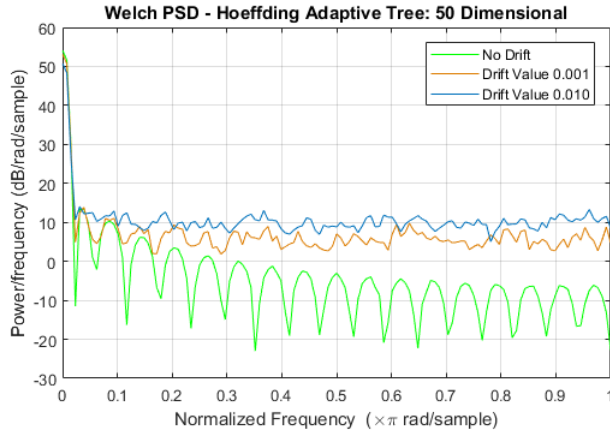


Figure 5.2: Welch PSD for HAT, 50 Dimensional input stream

correlated errors even though they are only conceptually similar at a high level. The details of their algorithms and even their specific definitions of “acceptable accuracy” are distinct. The very high correlation between DWM and HAT indicates that the HAT members are probably dominating the results of that ensemble. Given that the throughput of the single HAT is higher than DWM, it can provide the same accuracy at higher throughput. This was indicated in the 2D performance envelope plots and further shows the utility of such graphical summaries as aids in algorithmic selection.

It is important to note that using these time-series methods for comparisons is only valid if all classifiers are exposed to exactly the same input sequence. While this cannot be expected in the real world, it is still useful for pre-deployment evaluation and testing.

In some cases, performance was surprisingly better on higher-dimensional than lower-dimensional inputs. As discussed in the previous chapter, this could be related to operational features of the classifier models, some of which may gain additional information from the increased available attributes of each example.

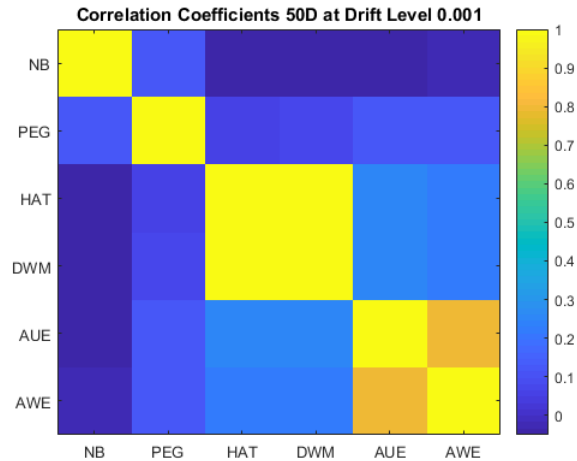


Figure 5.3: Correlations across classifiers, 50 Dimensional input stream at drift level (0.001)

The run times for this study, including pilot studies, exceeded 100 computer-hours. This made an extensive search of the parameter space infeasible. The comprehensive evaluation reported here was only performed on what could be considered “edge cases.” Thus if the run times could be reduced by using more powerful HPC architectures or improving the algorithm implementations further insights can be gained at a wider range of operating conditions.

Implications

In most of the classifiers examined in this study, the average classification levels were damaged beyond a reasonable level of acceptability, even by mild concept drifts. The AWE and AUE had the best resistance to concept drift at (0.001) but still were dramatically reduced in effectiveness at level (0.010). This implies that the concept drift mitigations have room for improvement. The lack of classifiers with scores in the upper right quadrant of the performance diagrams shown in figures 4.17 and 4.18 under drift conditions shows that the

accuracy and throughput truly become a trade-off in these situations.

This study also show that a clear difference is evident in classifiers that actively correct for drift and those that do not. In tests of classifiers without active drift-correction mechanisms, the accuracy quickly became unacceptable and stayed that way for the rest of the trial. However, the most powerful adaptive classifiers such as AWE, and AUE were able to correct themselves to near-baseline levels of performance.

Recommendations

The results of this study point to several directions for further research. First, it would be extremely useful to investigate improved overall throughput with HPC implementations (e.g. GPU accelerated versions) of the streaming classifiers. This would allow a detailed, expanded exploration of the concept drift parameter space beyond the boundary conditions studied here.

Another possible area to investigate further is the nature and limits of concept drift. For example, a pertinent question for different forms of drift is “At what point does drift frequency become too high to create meaningful patterns?”

Finally, there is clearly room for new or improved algorithms for more robust, active drift detection and correction. These would be able to operate successfully even when the drift rate is high yet there is still a pattern available to be learned. A possible avenue for future research is to vary the weights in the training window to give higher value to newer example instances. More stringent detection of classifier degradation may also help improve the overall response and mitigation of the drifts.

Summary

This dissertation report presented a comparative study of several supervised binary classification algorithms designed to maintain accuracy in the presence of concept drift. The accuracy-throughput trade-off diagram provides a clear summary of relative performances of the classifiers examined and can be easily used to select the appropriate algorithm for the application of interest. With the continued rise of “Big Data,” The performance of data-stream classifiers is of growing importance. Thus we need to adopt a principled approach to comprehensively evaluating the expected performance characteristics of candidate algorithms. This study is a beginning toward that end. It is my hope that this methodology will continue to be applied and become a valuable aid for system engineers designing high-performance stream analytics.

Appendix A

Algorithm Notation Guide

Algorithms are presented using the `algorithmicx` package (Janos, 2005) in a language neutral pseudocode format. For example, Euclid's greatest common denominator algorithm:

Algorithm A.0.1 Euclid's algorithm (Janos, 2005)

```
1: procedure EUCLID( $a, b$ ) ▷ The g.c.d. of a and b
2:    $r \leftarrow a \bmod b$ 
3:   while  $r \neq 0$  do ▷ We have the answer if r is 0
4:      $a \leftarrow b$ 
5:      $b \leftarrow r$ 
6:      $r \leftarrow a \bmod b$ 
7:   end while
8:   return  $b$  ▷ The gcd is b
9: end procedure
```

Appendix B

Source Code

The scripts used to launch experiments implemented straightforward calls to the MOA CLI and currently have no internal logic or error checking. A more sophisticated version can be re-factored from these prototypes if future experiments are pursued.

The Top Down Experiment Framework (TDEF.py) is a simple Python script that was used to marshal and document the command line options for each experimental conditions to allow for unattended data collection].

The StreamBenchmark.py script is a wrapper to MOA's stream speed evaluator and was used to establish a baseline for stream generation speed on each architecture used in the study.

The StreamExample.py script is a wrapper to MOA's stream writer and was used to create a few low dimensional data sets to explore visual analogies of the 10 and 50 dimensional streams used in the experiments.

```
1 # TDEF_main: Top-Down Experiment Framework (entry point)
2 #
3 # Author Stefan Joe-Yen(username:Amidan)
4 # Revision Date: July 2017
5 # Summary: The main experiment run file for "Performance Envelopes
6 #           of Adaptive Ensemble Stream Classifiers"
7 #           This is a preliminary solution and should be refactored
8 #           in the future if re-use is needed
9 #           *NOTE* This is hard-coded with Seed 2 for TRIAL B
10 #
11 import os
12
13 # Welcome Message
14 print("\n=== Welcome to TDEF ===\n")
15
16 # Accuracy Updated Ensembe Series with 5 member classifiers
17 print("\n=== Collecting Data for Algorithm 1 with 10 Dimensional
18 #           Input (AUE5) ===\n")
19 print("\n=== (AUE5) Drift Rate - None ===\n")
20 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
```

```

    .DoTask "EvaluatePrequential -l (meta.AccuracyUpdatedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5) -s(generators.
    RandomRBFGeneratorDrift -r 2 -i 2)-i 1000000 -f 10000" > AUE5-
    T00b.txt')
19 print("\n=== (AUE5) Drift Rate - Standard ===\n")
20 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.AccuracyUpdatedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5) -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -i 1000000 -f 10000"
    > AUE5-T01b.txt')
21 print("\n=== (AUE5) Drift Rate - High ===\n")
22 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.AccuracyUpdatedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5) -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -i 1000000 -f 10000"
    > AUE5-T02b.txt')
23
24 print("\n=== Collecting Data for Algorithm 1 with 50 Dimensional
    Input (AUE5) ===\n")
25 print("\n=== (AUE5) Drift Rate - None ===\n")
26 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.AccuracyUpdatedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5) -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50 -i 1000000 -f 10000" >
    AUE5-T03b.txt')
27 print("\n=== (AUE5) Drift Rate - Standard ===\n")
28 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.AccuracyUpdatedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5) -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i 1000000 -f
    10000" > AUE5-T04b.txt')
29 print("\n=== (AUE5) Drift Rate - High ===\n")
30 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.AccuracyUpdatedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5) -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -i 1000000 -f
    10000" > AUE5-T05b.txt')
31
32
33 # Accuracy Weighted Ensemble Series with 5 member classifiers
34 print("\n\n=== Collecting Data for Algorithm 2 with 10 Dimensional
    Input (AWE5) ===\n")
35 print("\n=== (AWE5) Drift Rate - None ===\n")
36 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.AccuracyWeightedEnsemble -l
    trees.HoeffdingAdaptiveTree -n 5.0) -s(generators.
    RandomRBFGeneratorDrift -r 2 -i 2)-i 1000000 -f 10000" > AWE5-

```

```

T00b.txt')
37 print("\n=== (AWE5) Drift Rate - Standard ===\n")
38 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
.DoTask "EvaluatePrequential -l (meta.AccuracyWeightedEnsemble -l
trees.HoeffdingAdaptiveTree -n 5.0) -s (generators.
RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -i 10000000 -f 10000"
> AWE5-T01b.txt')
39 print("\n=== (AWE5) Drift Rate - High ===\n")
40 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
.DoTask "EvaluatePrequential -l (meta.AccuracyWeightedEnsemble -l
trees.HoeffdingAdaptiveTree -n 5.0) -s (generators.
RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -i 10000000 -f 10000"
> AWE5-T02b.txt')
41
42 print("\n\n=== Collecting Data for Algorithm 2 with 50 Dimensional
Input (AWE5) ===\n")
43 print("\n=== (AWE5) Drift Rate - None ===\n")
44 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
.DoTask "EvaluatePrequential -l (meta.AccuracyWeightedEnsemble -l
trees.HoeffdingAdaptiveTree -n 5.0) -s (generators.
RandomRBFGeneratorDrift -r 2 -i 2 -a 50 -i 10000000 -f 10000" >
AWE5-T03b.txt')
45 print("\n=== (AWE5) Drift Rate - Standard ===\n")
46 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
.DoTask "EvaluatePrequential -l (meta.AccuracyWeightedEnsemble -l
trees.HoeffdingAdaptiveTree -n 5.0) -s (generators.
RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i 10000000 -f
10000" > AWE5-T04b.txt')
47 print("\n=== (AWE5) Drift Rate - High ===\n")
48 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
.DoTask "EvaluatePrequential -l (meta.AccuracyWeightedEnsemble -l
trees.HoeffdingAdaptiveTree -n 5.0) -s (generators.
RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -i 10000000 -f
10000" > AWE5-T05b.txt')
49
50
51 # Dynamic Weighted Majority Series
52 print("\n\n=== Collecting Data for Algorithm 3 with 10 Dimensional
Input (DWM) ===\n")
53 print("\n=== (DWM) Drift Rate - None ===\n")
54 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
.DoTask "EvaluatePrequential -l (meta.WeightedMajorityAlgorithm -
l trees.HoeffdingAdaptiveTree,trees.HoeffdingAdaptiveTree,
functions.SPegasos,functions.SPegasos,bayes.NaiveBayes) -s (
generators.RandomRBFGeneratorDrift -r 2 -i 2) -i 10000000 -f
10000" > DWM-T00b.txt')
55 print("\n=== (DWM) Drift Rate - Standard ===\n")

```

```

56 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.WeightedMajorityAlgorithm -
    l trees.HoeffdingAdaptiveTree,trees.HoeffdingAdaptiveTree,
    functions.SPegasos,functions.SPegasos,bayes.NaiveBayes) -s (
    generators.RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -i
    10000000 -f 10000" > DWM-T01b.txt')
57 print("\n=== (DWM) Drift Rate - High ===\n")
58 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.WeightedMajorityAlgorithm -
    l trees.HoeffdingAdaptiveTree,trees.HoeffdingAdaptiveTree,
    functions.SPegasos,functions.SPegasos,bayes.NaiveBayes) -s (
    generators.RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -i
    10000000 -f 10000" > DWM-T02b.txt')
59
60
61 print("\n\n=== Collecting Data for Algorithm 3 with 50 Dimensional
    Input (DWM) ===\n")
62 print("\n=== (DWM) Drift Rate - None ===\n")
63 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.WeightedMajorityAlgorithm -
    l trees.HoeffdingAdaptiveTree,trees.HoeffdingAdaptiveTree,
    functions.SPegasos,functions.SPegasos,bayes.NaiveBayes) -s (
    generators.RandomRBFGeneratorDrift -r 2 -i 2 -a 50) -i 10000000 -
    f 10000" > DWM-T03b.txt')
64 print("\n=== (DWM) Drift Rate - Standard ===\n")
65 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.WeightedMajorityAlgorithm -
    l trees.HoeffdingAdaptiveTree,trees.HoeffdingAdaptiveTree,
    functions.SPegasos,functions.SPegasos,bayes.NaiveBayes) -s (
    generators.RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i
    10000000 -f 10000" > DWM-T04b.txt')
66 print("\n=== (DWM) Drift Rate - High ===\n")
67 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l (meta.WeightedMajorityAlgorithm -
    l trees.HoeffdingAdaptiveTree,trees.HoeffdingAdaptiveTree,
    functions.SPegasos,functions.SPegasos,bayes.NaiveBayes) -s (
    generators.RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -i
    10000000 -f 10000" > DWM-T05b.txt')
68
69 # Naive Bayes Series
70 print("\n=== Collecting Data for Algorithm 4 with 10 Dimensional
    Input (NB) ===\n")
71 print("\n=== (NB) Drift Rate - None ===\n")
72 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l bayes.NaiveBayes -s(generators.
    RandomRBFGeneratorDrift -r 2 -i 2)-i 10000000 -f 10000" > NB-T00b
    .txt')

```

```

73 print("\n=== (NB) Drift Rate - Standard ===\n")
74 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l bayes.NaiveBayes -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -i 10000000 -f 10000"
    > NB-T01b.txt')
75 print("\n=== (NB) Drift Rate - High ===\n")
76 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l bayes.NaiveBayes -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -i 10000000 -f 10000"
    > NB-T02b.txt')
77
78 print("\n=== Collecting Data for Algorithm 4 with 50 Dimensional
    Input (NB) ===\n")
79 print("\n=== (NB) Drift Rate - None ===\n")
80 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l bayes.NaiveBayes -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50 -i 10000000 -f 10000" >
    NB-T03b.txt')
81 print("\n=== (NB) Drift Rate - Standard ===\n")
82 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l bayes.NaiveBayes -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i 10000000 -f
    10000" > NB-T04b.txt')
83 print("\n=== (NB) Drift Rate - High ===\n")
84 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l bayes.NaiveBayes -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -i 10000000 -f
    10000" > NB-T05b.txt')
85
86
87 # Stochastic Pegasos SVM Series
88 print("\n=== Collecting Data for Algorithm 5 with 10 Dimensional
    Input (PEG) ===\n")
89 print("\n=== (PEG) Drift Rate - None ===\n")
90 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l functions.SPegasos -s(generators.
    RandomRBFGeneratorDrift -r 2 -i 2)-i 10000000 -f 10000" > PEG-
    T00b.txt')
91 print("\n=== (PEG) Drift Rate - Standard ===\n")
92 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l functions.SPegasos -s (generators
    .RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -i 10000000 -f
    10000" > PEG-T01b.txt')
93 print("\n=== (PEG) Drift Rate - High ===\n")
94 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l functions.SPegasos -s (generators
    .RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -i 10000000 -f

```

```

10000" > PEG-T02b.txt')
95
96 print("\n=== Collecting Data for Algorithm 5 with 50 Dimensional
    Input (PEG) ===\n")
97 print("\n=== (PEG) Drift Rate - None ===\n")
98 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l functions.SPegasos -s (generators
    .RandomRBFGeneratorDrift -r 2 -i 2 -a 50 -i 10000000 -f 10000" >
    PEG-T03b.txt')
99 print("\n=== (PEG) Drift Rate - Standard ===\n")
100 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l functions.SPegasos -s (generators
    .RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i 10000000 -f
    10000" > PEG-T04b.txt')
101 print("\n=== (PEG) Drift Rate - High ===\n")
102 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l functions.SPegasos -s (generators
    .RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -i 10000000 -f
    10000" > PEG-T05b.txt')
103
104
105 # Hoeffding Adaptive Tree Series
106 print("\n=== Collecting Data for Algorithm 6 with 10 Dimensional
    Input (HAT) ===\n")
107 print("\n=== (HAT) Drift Rate - None ===\n")
108 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s(
    generators.RandomRBFGeneratorDrift -r 2 -i 2)-i 10000000 -f
    10000" > HAT-T00b.txt')
109 print("\n=== (HAT) Drift Rate - Standard ===\n")
110 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (
    generators.RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -i
    10000000 -f 10000" > HAT-T01b.txt')
111 print("\n=== (HAT) Drift Rate - High ===\n")
112 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (
    generators.RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -i
    10000000 -f 10000" > HAT-T02b.txt')
113
114 print("\n=== Collecting Data for Algorithm 6 with 50 Dimensional
    Input (HAT) ===\n")
115 print("\n=== (HAT) Drift Rate - None ===\n")
116 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (
    generators.RandomRBFGeneratorDrift -r 2 -i 2 -a 50 -i 10000000 -f
    10000" > HAT-T03b.txt')

```

```

117 print("\n=== (HAT Drift Rate - Standard ===\n")
118 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (
    generators.RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i
    10000000 -f 10000" > HAT-T04b.txt')
119 print("\n=== (HAT) Drift Rate - High ===\n")
120 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (
    generators.RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -i
    10000000 -f 10000" > HAT-T05b.txt')
121
122
123 # Termination Message
124 print("\n=== TDEF Exiting ===\n")
125
126 # Example With Seeds
127 # "EvaluatePrequential -l functions.SPegasos -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -i 10000000 -f
    10000"

```

Listing B.1: TDEF script

```

1 # Stream Speed Test
2 #
3 # Author Stefan Joe-Yen(username:Amidan)
4 # Revision Date: July 2017
5 # Summary: The stream benchmarking tool for "Performance Envelopes
    of Adaptive Ensemble Stream Classifiers"
6 #
    This is a preliminary solution and should be refactored
    in the future to collect and average data automatically (5
    samples per condition on each machine )
7 #
    *NOTE* This is hard-coded with Seed 2 (default no-args)
    for TRIAL B
8 #
9 import os
10
11 print("\n=== Stream Speed Test Starting ===\n")
12
13 print("\n=== Measuring Stream Generator Speed (Drift Rate None,
    Attributes 10) ===\n")
14 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2) -g 10000000" >
    DriftGenSpeed000d10b1.txt')
15 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2) -g 10000000" >
    DriftGenSpeed000d10b2.txt')

```

```

16 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2) -g 10000000" >
    DriftGenSpeed000d10b3.txt')
17 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2) -g 10000000" >
    DriftGenSpeed000d10b4.txt')
18 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2) -g 10000000" >
    DriftGenSpeed000d10b5.txt')
19
20
21 print("\n=== Measuring Stream Generator Speed (Drift Rate 0.001,
    Attributes 10) ===\n")
22 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed001d10b1.txt')
23 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed001d10b2.txt')
24 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed001d10b3.txt')
25 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001) -g 10000000" >
    DriftGenSpeed001d10b4.txt')
26 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001) -g 10000000" >
    DriftGenSpeed001d10b5.txt')
27
28
29 print("\n=== Measuring Stream Generator Speed (Drift Rate 0.010,
    Attributes 10) ===\n")
30 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed010d10b1.txt')
31 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -g 10000000" >

```



```

    DriftGenSpeed010d10b2.txt')
32 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed010d10b3.txt')
33 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed010d10b4.txt')
34 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2) -g 10000000" >
    DriftGenSpeed010d10b5.txt')
35
36
37 print("\n=== Measuring Stream Generator Speed (Drift Rate None,
    Attributes 50) ===\n")
38 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed000d50b1.txt')
39 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed000d50b2.txt')
40 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed000d50b3.txt')
41 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed000d50b4.txt')
42 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed000d50b5.txt')
43
44
45 print("\n=== Measuring Stream Generator Speed (Drift Rate 0.001,
    Attributes 50) ===\n")
46 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed001d50b1.txt')
47 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.

```

```

    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed001d50b2.txt')
48 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed001d50b3.txt')
49 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed001d50b4.txt')
50 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.001 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed001d50b5.txt')
51
52
53 print("\n=== Measuring Stream Generator Speed (Drift Rate 0.010,
    Attributes 50) ===\n")
54 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed010d50b1.txt')
55 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed010d50b2.txt')
56 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed010d50b3.txt')
57 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed010d50b4.txt')
58 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
    .DoTask "MeasureStreamSpeed -s (generators.
    RandomRBFGeneratorDrift -s 0.010 -r 2 -i 2 -a 50) -g 10000000" >
    DriftGenSpeed010d50b5.txt')
59
60
61 print("\n=== Stream Speed Test Done ===\n")

```

Listing B.2: StreamBenchmark script

```

1 # Stream Speed Test
2 #
3 # Author Stefan Joe-Yen(username:Amidan)
4 # Revision Date: July 2017

```

```

5 # Summary: The stream writer tool for low-D example figures for "
  Performance Envelopes of Adaptive Ensemble Stream Classifiers"
6 #       This is a preliminary solution and should be refactored
  in the future with centralized param. passing
7 #       *NOTE* This is hard-coded with Seed 1 (default no-args)
  for TRIAL A
8 #
9 import os
10
11 print("\n=== Stream Demonstration Create Low Dimension Examples ===\n")
12
13 print("\n=== Dump 2D Data to File (Drift Rate None, Attributes 2)
  ===\n")
14 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
  .DoTask "WriteStreamToARFFFFile -s (generators.
  RandomRBFGeneratorDrift -a 2) -f 2D-NoDriftExample.arff -m 5000"')
15
16 print("\n=== Dump 2D Data to File (Drift Rate Low(0.001), Attributes
  2) ===\n")
17 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
  .DoTask "WriteStreamToARFFFFile -s (generators.
  RandomRBFGeneratorDrift -s 0.001 -a 2) -f 2D-LowDriftExample.arff
  -m 5000"')
18
19 print("\n=== Dump 2D Data to File (Drift Rate High(0.010),
  Attributes 2) ===\n")
20 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
  .DoTask "WriteStreamToARFFFFile -s (generators.
  RandomRBFGeneratorDrift -s 0.010 -a 2) -f 2D-HighDriftExample.
  arff -m 5000"')
21
22 print("\n=== Dump 1D Data to File (Drift Rate None, Attributes 2)
  ===\n")
23 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
  .DoTask "WriteStreamToARFFFFile -s (generators.
  RandomRBFGeneratorDrift -k 2 -a 1 -n 2) -f 1D-NoDriftExample.arff
  -m 1000"')
24
25 print("\n=== Dump 1D Data to File (Drift Rate Low(0.001), Attributes
  2) ===\n")
26 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa
  .DoTask "WriteStreamToARFFFFile -s (generators.
  RandomRBFGeneratorDrift -s 0.001 -k 2 -a 1 -n 2) -f 1D-
  LowDriftExample.arff -m 1000"')
27

```

```
28 print("\n=== Dump 1D Data to File (Drift Rate High(0.010),  
    Attributes 2) ===\n")  
29 os.system('java -Xmx4G -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa  
    .DoTask "WriteStreamToARFFFile -s (generators.  
    RandomRBFGeneratorDrift -s 0.010 -k 2 -a 1 -n 2) -f 1D-  
    HighDriftExample.arff -m 1000"')  
30  
31 print("\n=== Stream Demo File Creator Done ===\n")
```

Listing B.3: StreamExample script

Appendix C

Additional Results, Figures, and Tables

Full Sized Figures

This section reiterates some of the figures and discussion from Chapter 4 at full-page size to show the detail in images.

The Eponymous Performance Envelopes

These figures summarize the performance of each classifier by rendering them as points in a two-dimensional space. Note that the accuracy axis starts at 50% since any performance at or below that value is considered a complete failure - no better than chance. The lines of 50% throughput and 75% correct classification rate are highlighted to divide the space into quadrants. Classifiers which fall in the upper right quadrant are most desirable since they are both fast and accurate. Conversely the lower left quadrant indicates the poorest performance from a classifier algorithm.

Figure C.1 shows the performance trade-off diagram for the 10-dimensional inputs.

Figure C.2 shows the performance trade-off diagram for the 50-dimensional inputs.

Time-series Characteristics of Classifier Error-rates

Additional insights into classifier performance can be gained by examining the time course of their error rates.

Looking at the performance of Pegasos SVM over time in figures C.3 and C.4 we can clearly see the damaging effect of drift on accuracy. Pegasos has no active drift correction and so especially in figure C.4 systematically decaying accuracy can be noted.

In figures C.5 and C.6 we observe a flatter time course. While the Bayesian classifier does not have explicit drift correction, it is adaptive by nature in its operation in that it is constantly updating its model based on the latest evidence whether there is a drift or not. This is further evident in the 50 dimensional case where we see wider oscillations in the drift cases since the model is struggling to keep up with the shifting ground truth.

Figures C.7 and C.8 show the time courses of HAT, the first of the classifiers with explicit drift correction. as in the NB case, the accuracy does not decay past a certain point in each of the drifts. However, it is never corrected back to its non-drift level especially when the drift becomes high.

In the figures for DWM: C.9 and C.10, we see much the same effect as in the single HAT. Accuracy levels under drift conditions are prevented from decaying and remain in a fairly acceptable range but never achieve the levels seen with no drift.

The ensembles that are directly related to the estimated accuracy of their members shown in C.13 through C.14. exhibit the most robust drift correction some of which, in the AUE case, are close to the non-drift performance levels.

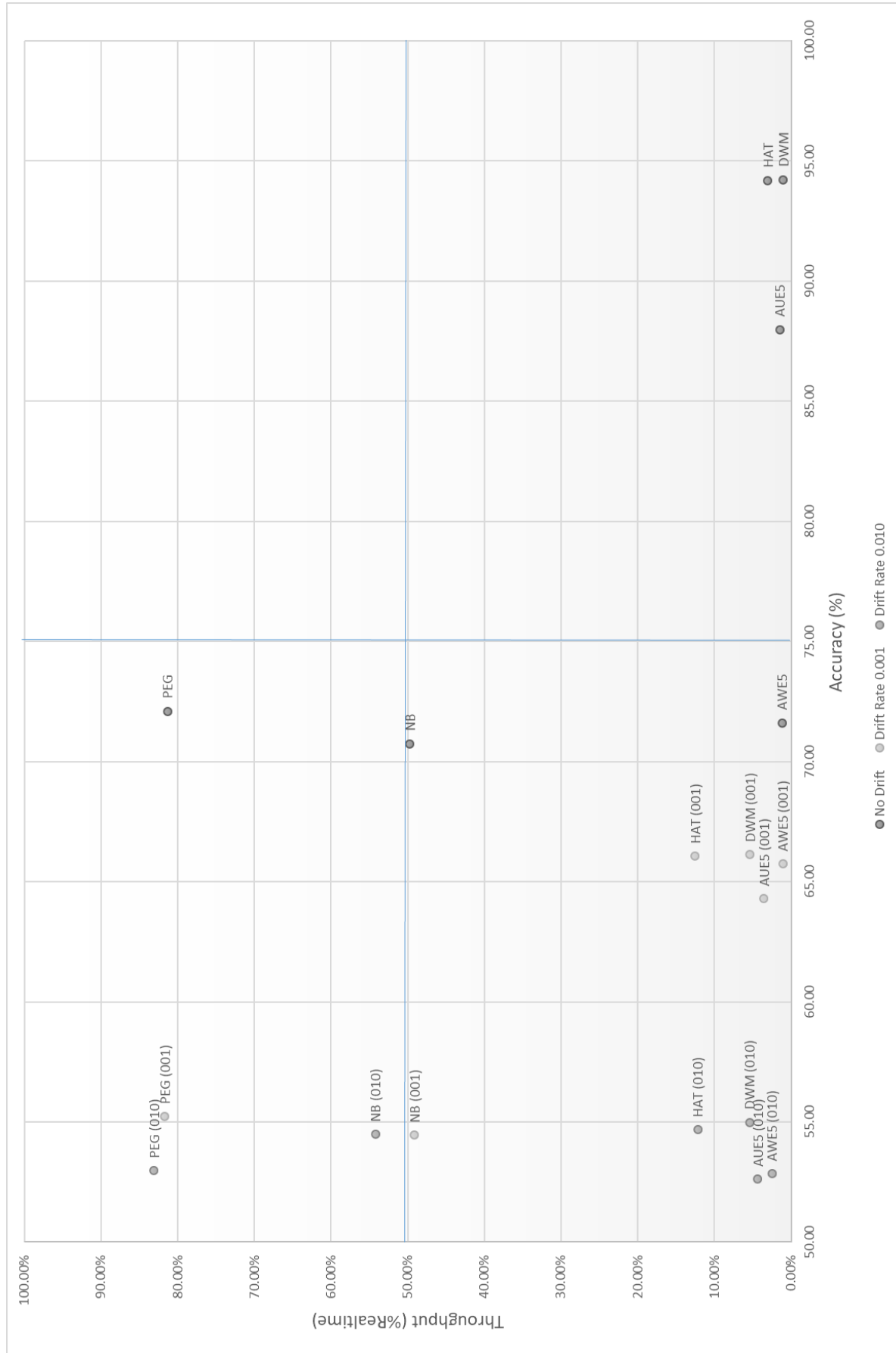


Figure C.1: Accuracy vs. Throughput Trade-off (10D)

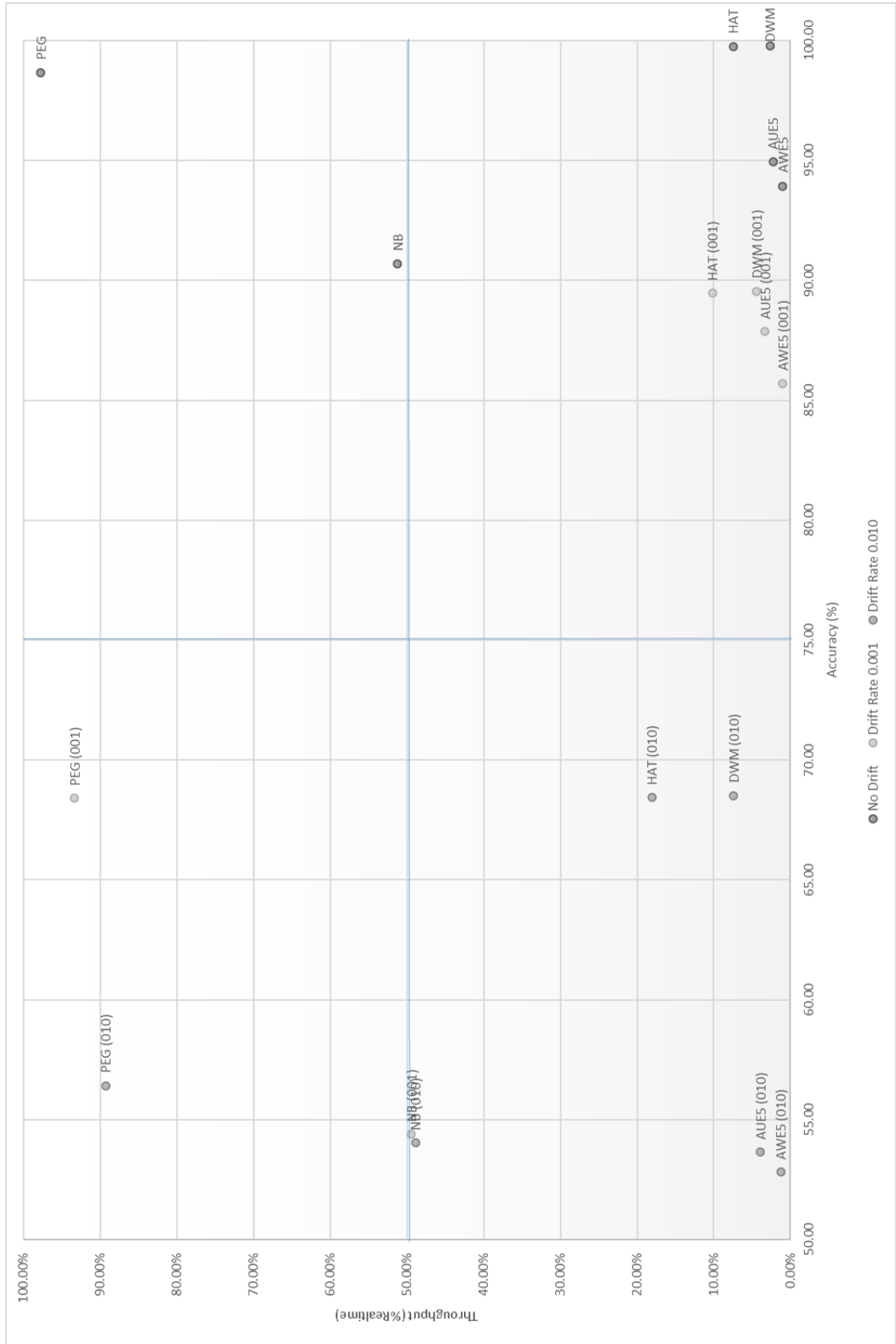


Figure C.2: Accuracy vs. Throughput Trade-off (50D)

Stochastic Pegasos SVM (10 Dimensional, Series D)

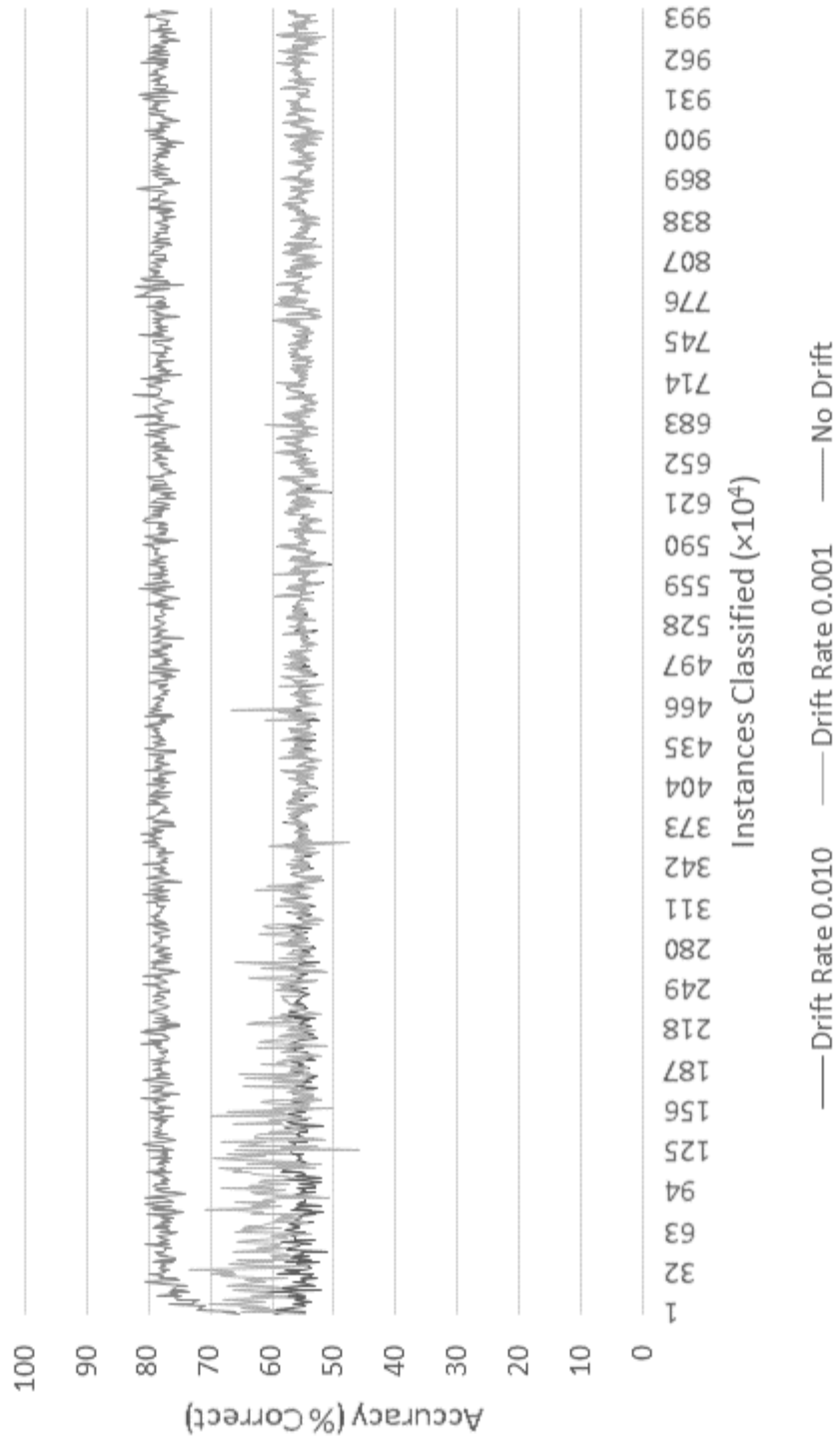


Figure C.3: Pegasos SVM Time Series (10D)

Stochastic Pegasos SVM (50 Dimensional, Series D)

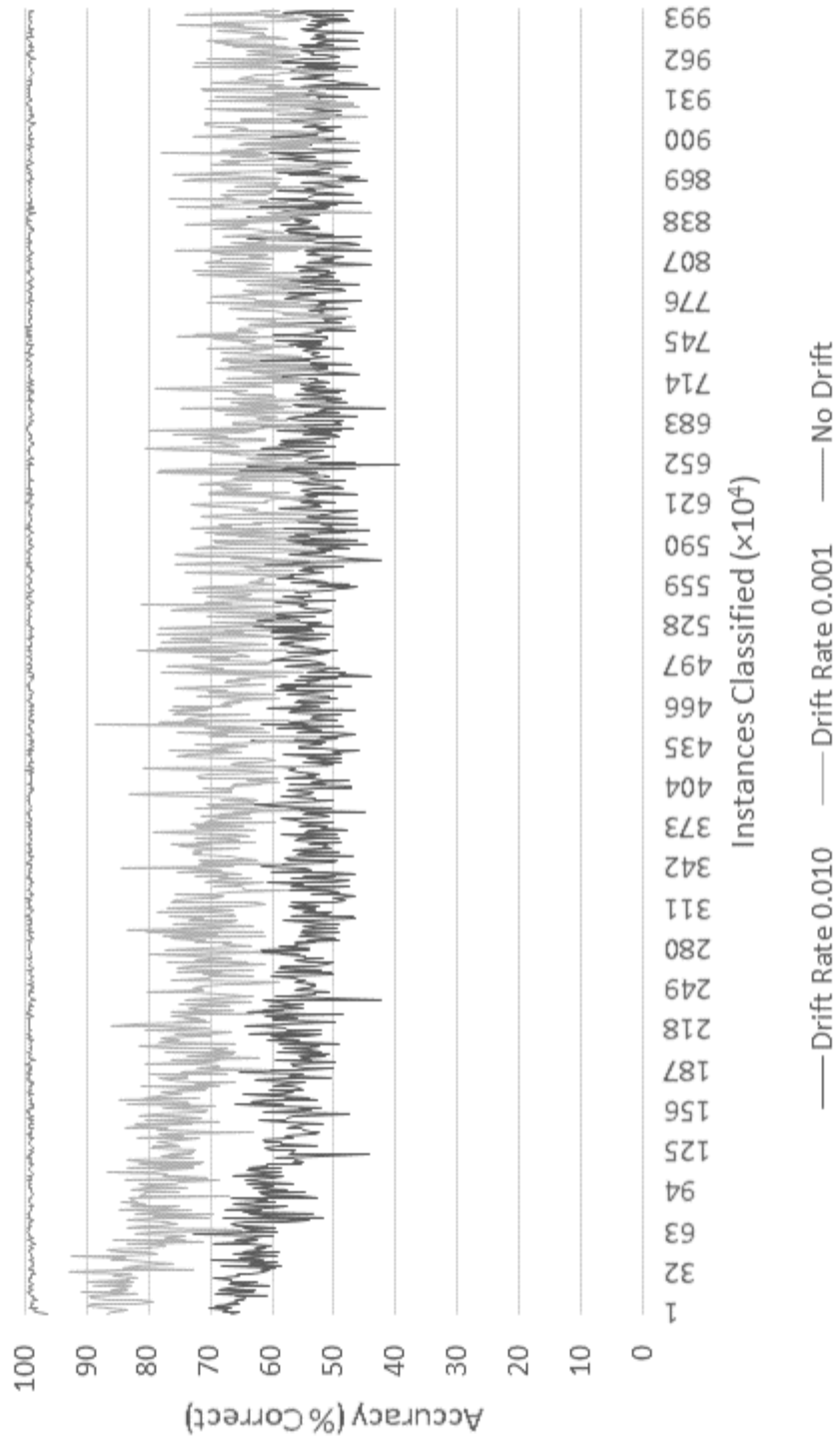


Figure C.4: Pegasos SVM Time Series (50D)

Naïve Bayes (10 Dimensional, Series D)

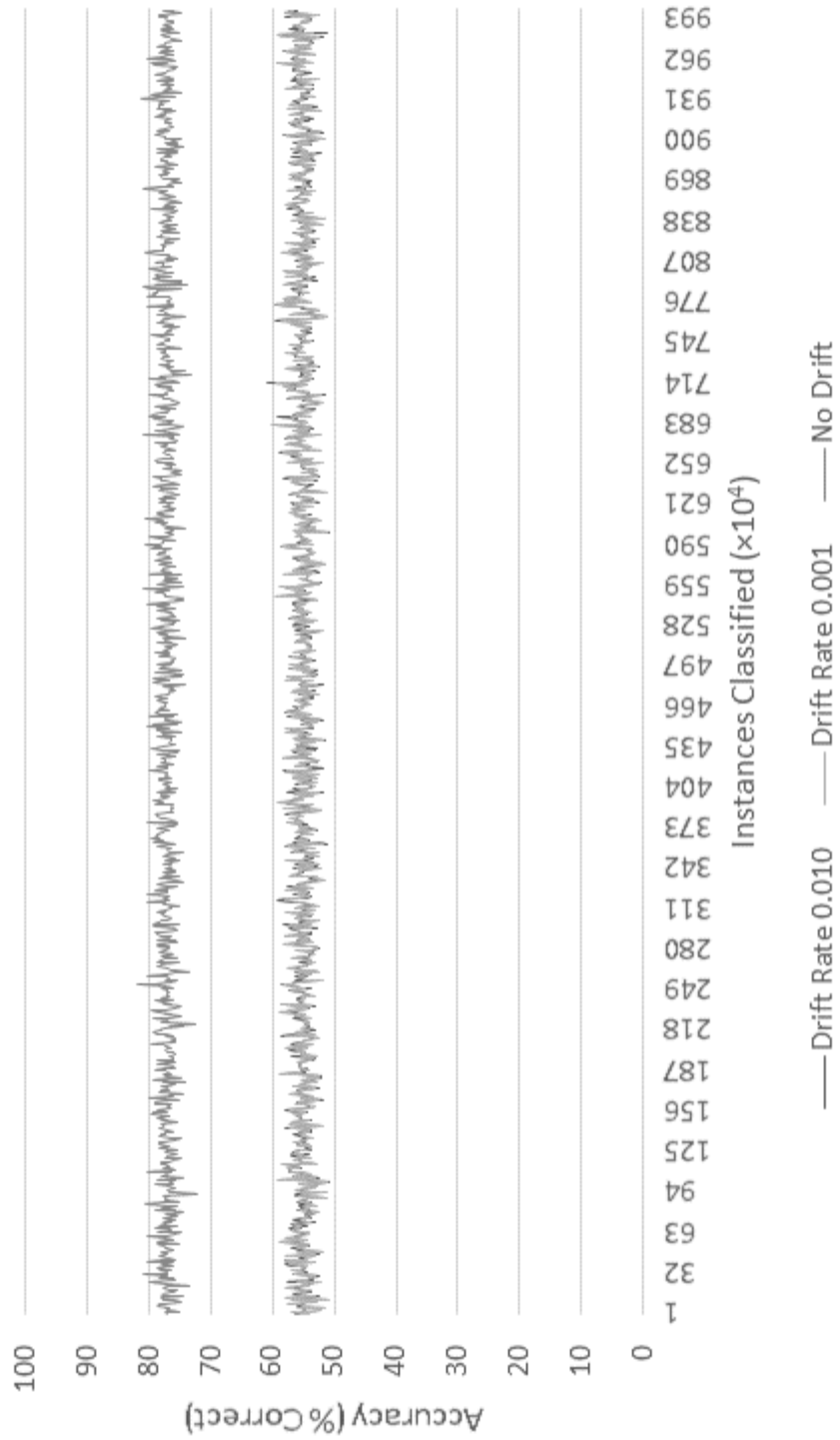


Figure C.5: Naive Bayes Time Series (10D)

Naïve Bayes (50 Dimensional, Series D)

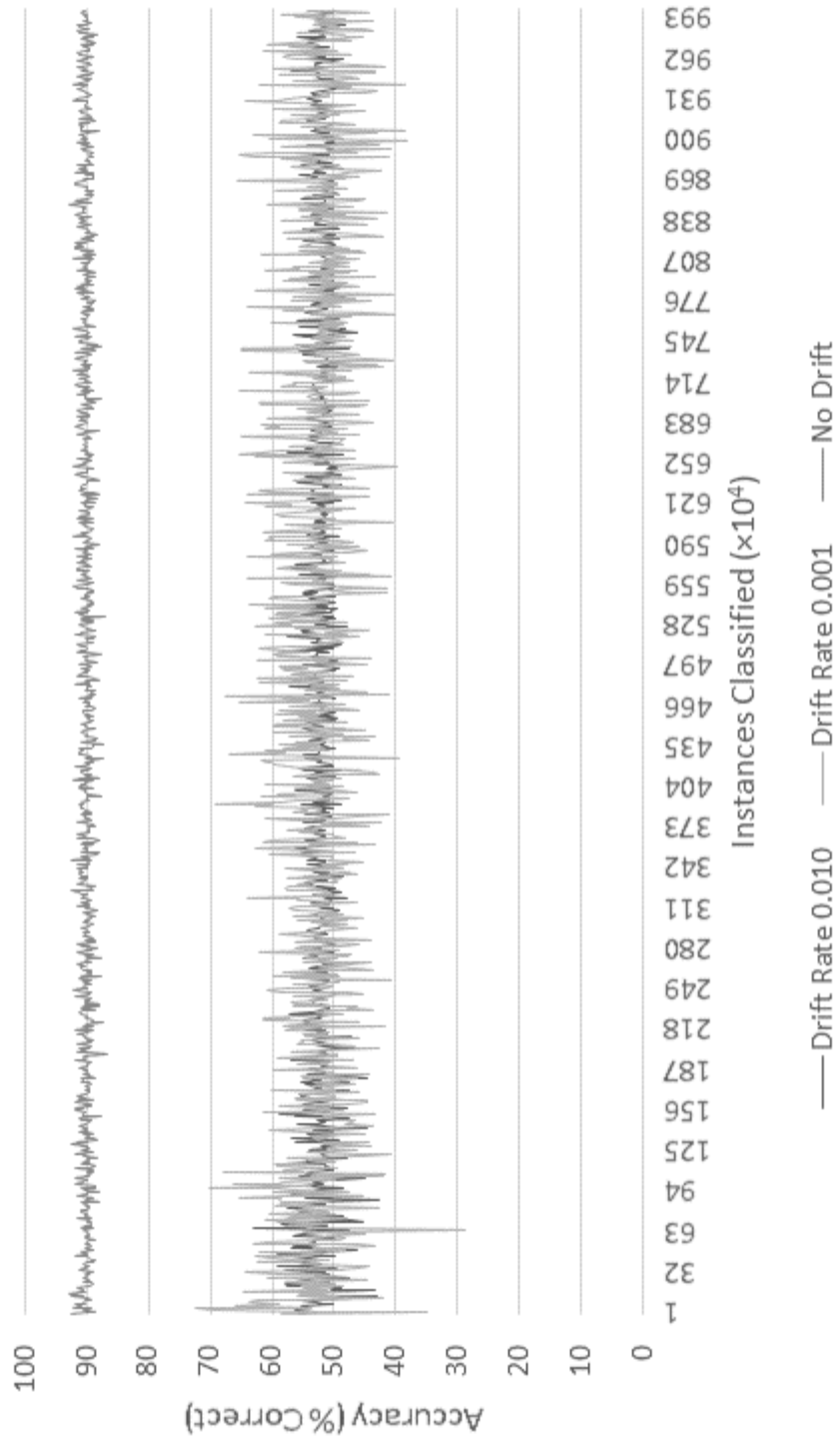


Figure C.6: Naive Bayes Time Series (50D)

Hoeffding Adaptive Tree (10 Dimensional, Series D)

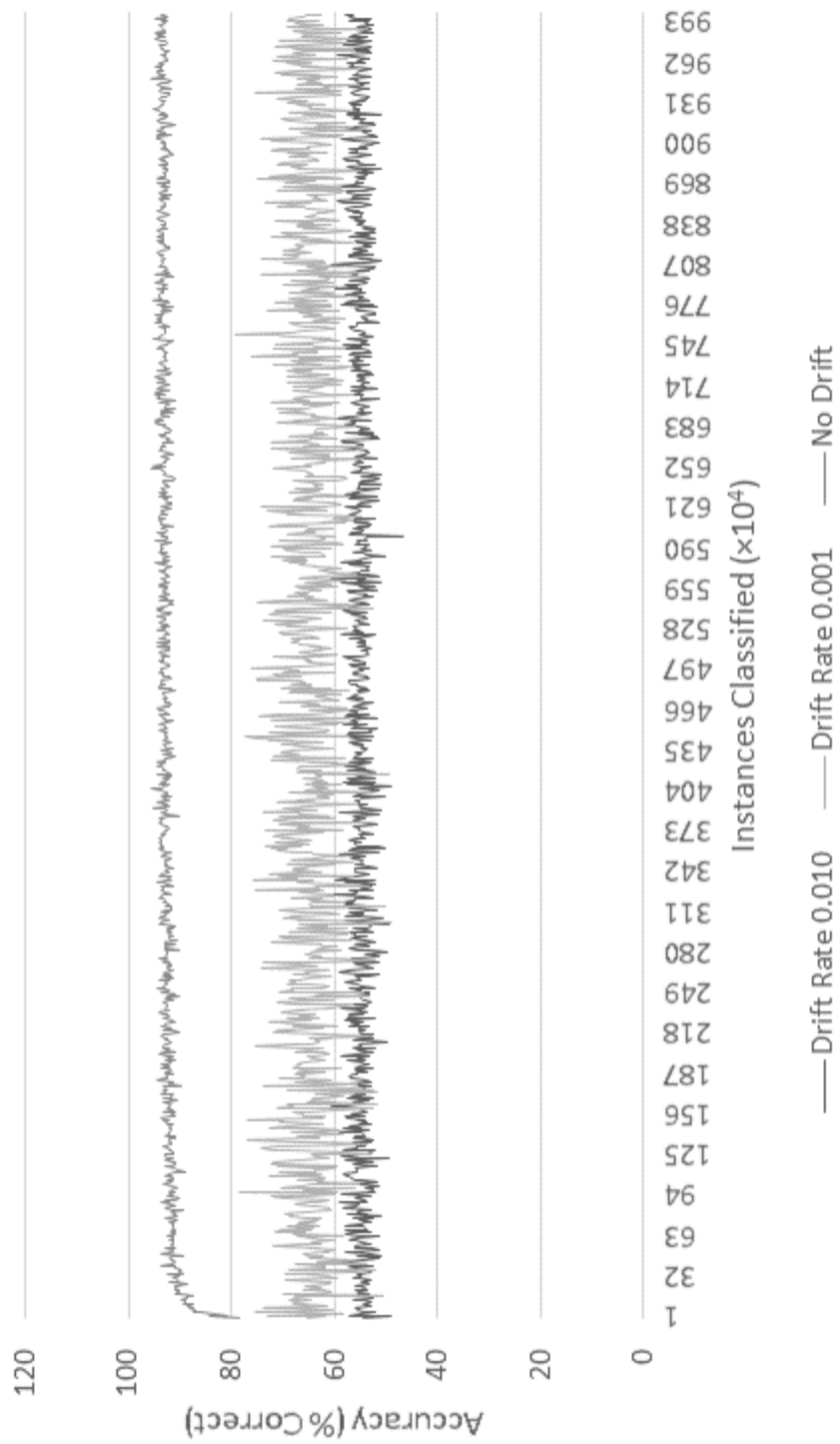


Figure C.7: Hoeffding Adaptive Tree Time Series (10D)

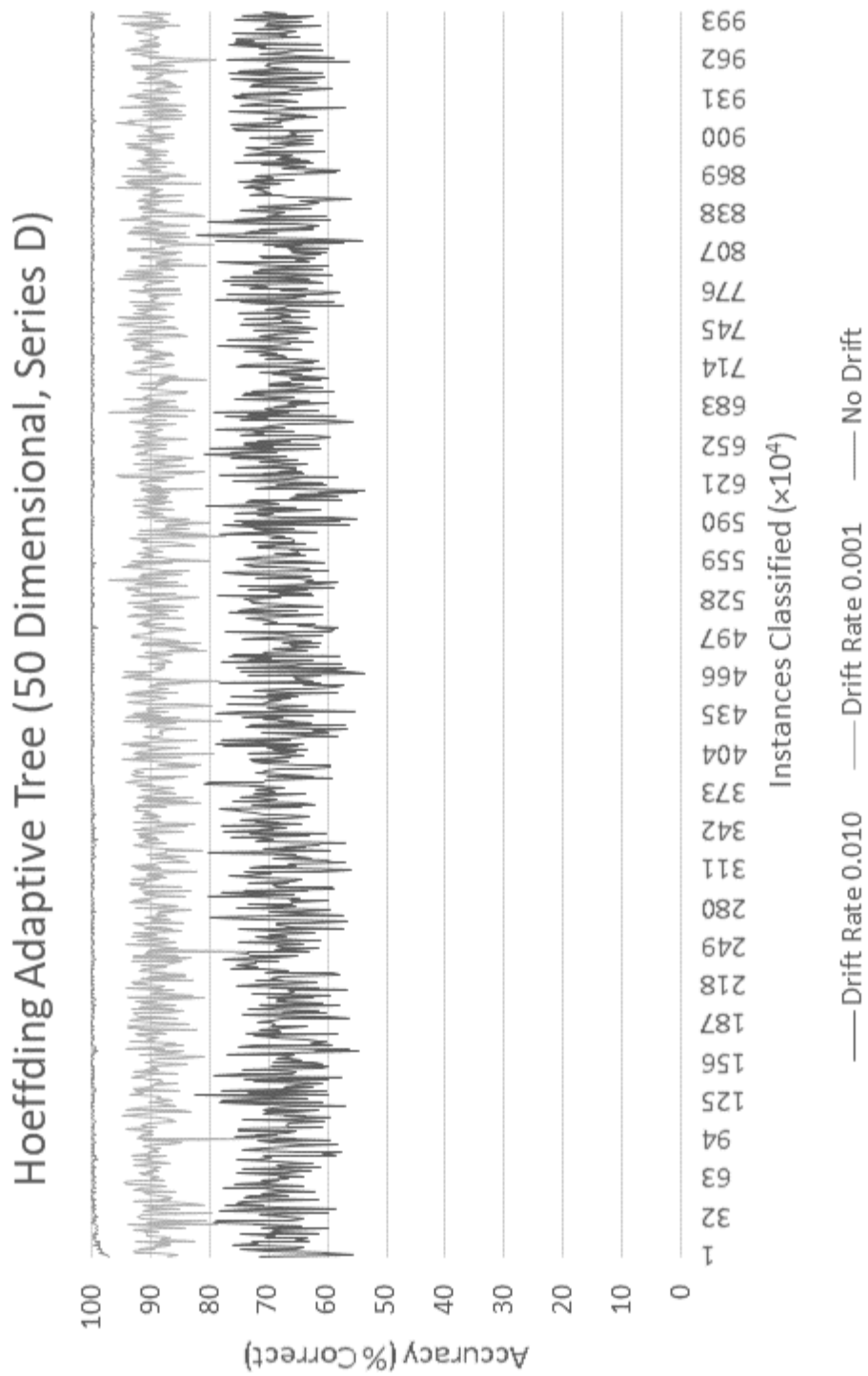


Figure C.8: Hoeffding Adaptive Tree Time Series (50D)

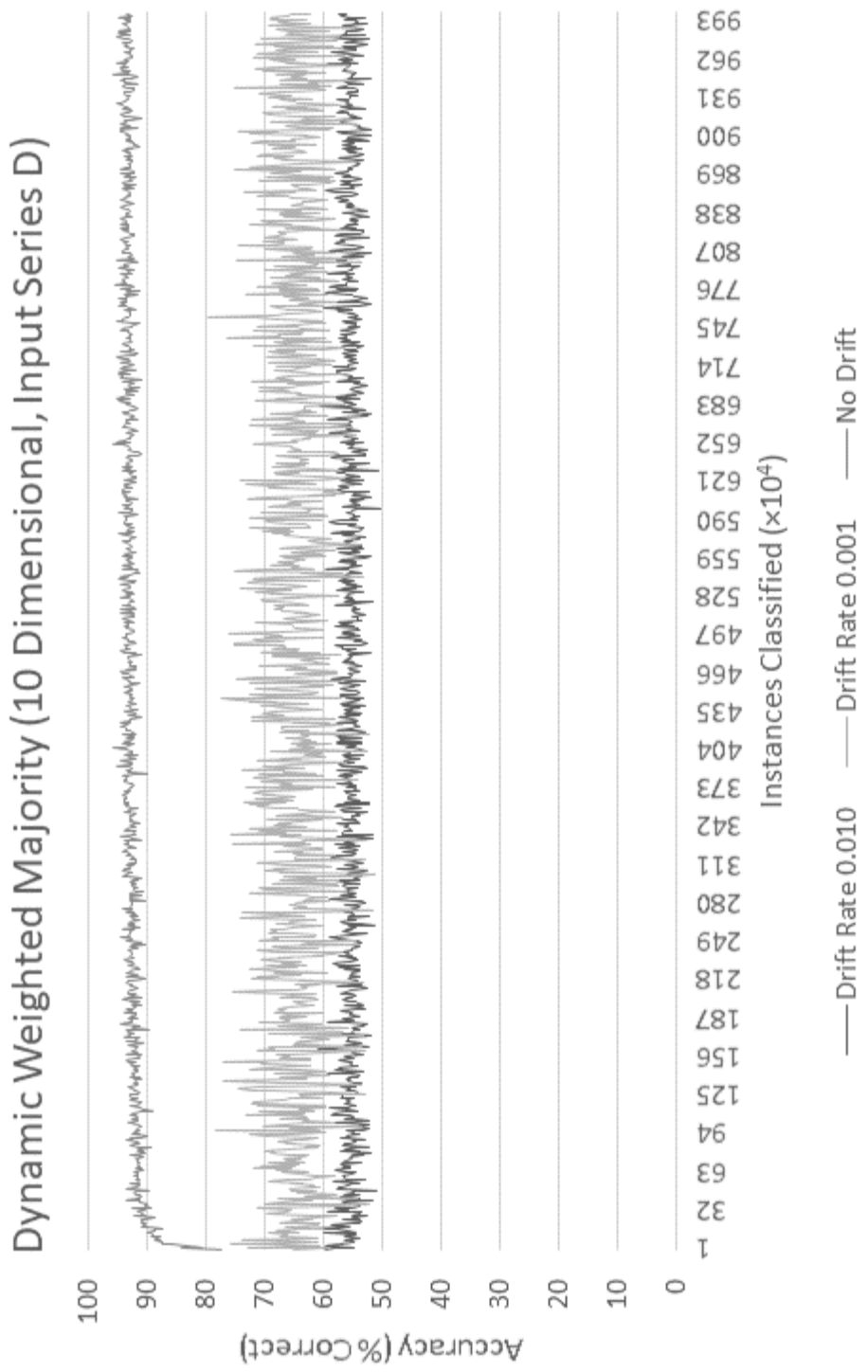


Figure C.9: Dynamic Weighted Majority Time Series (10D)

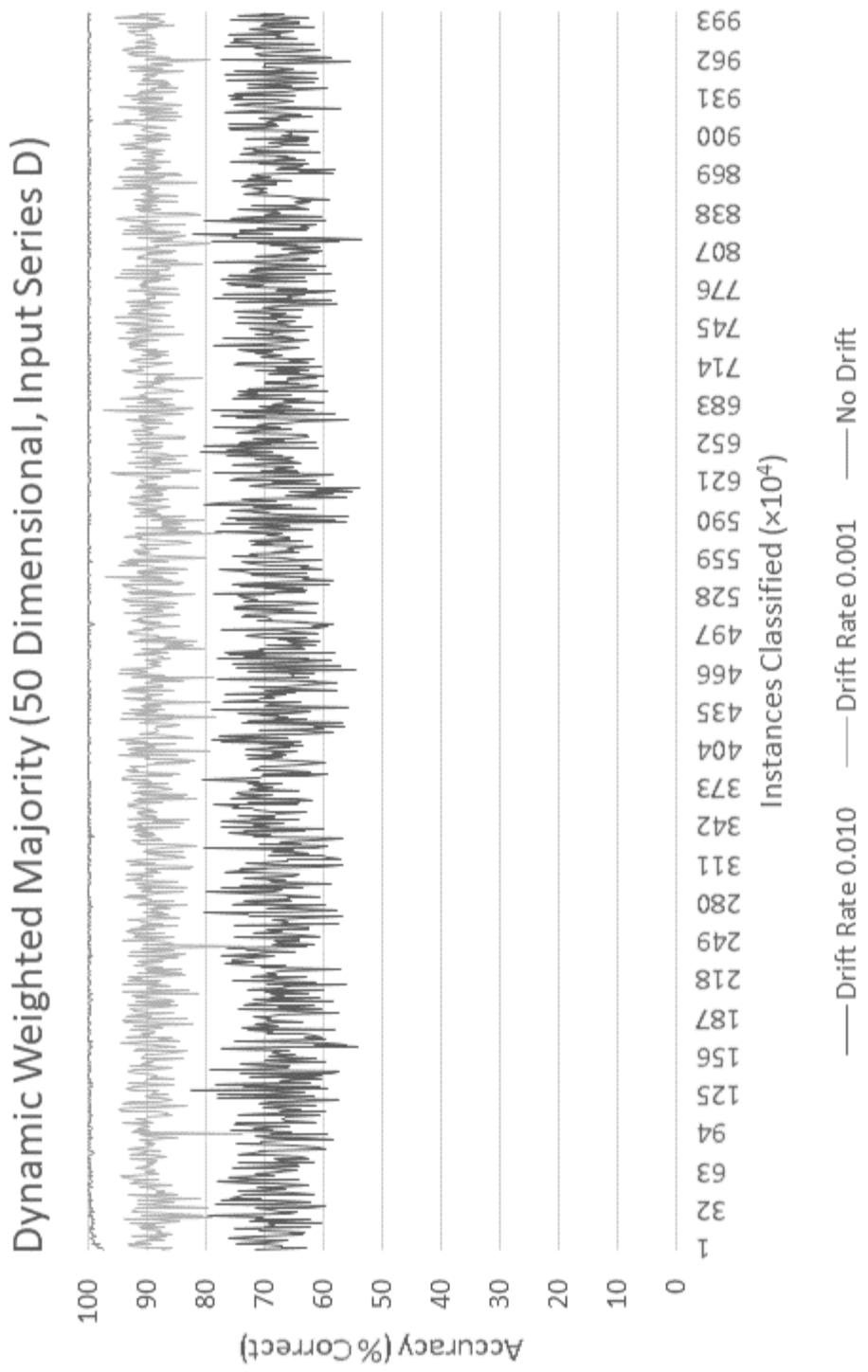


Figure C.10: Dynamic Weighted Majority Time Series (50D)

Accuracy Weighted Ensemble (10 Dimensional, Series D)

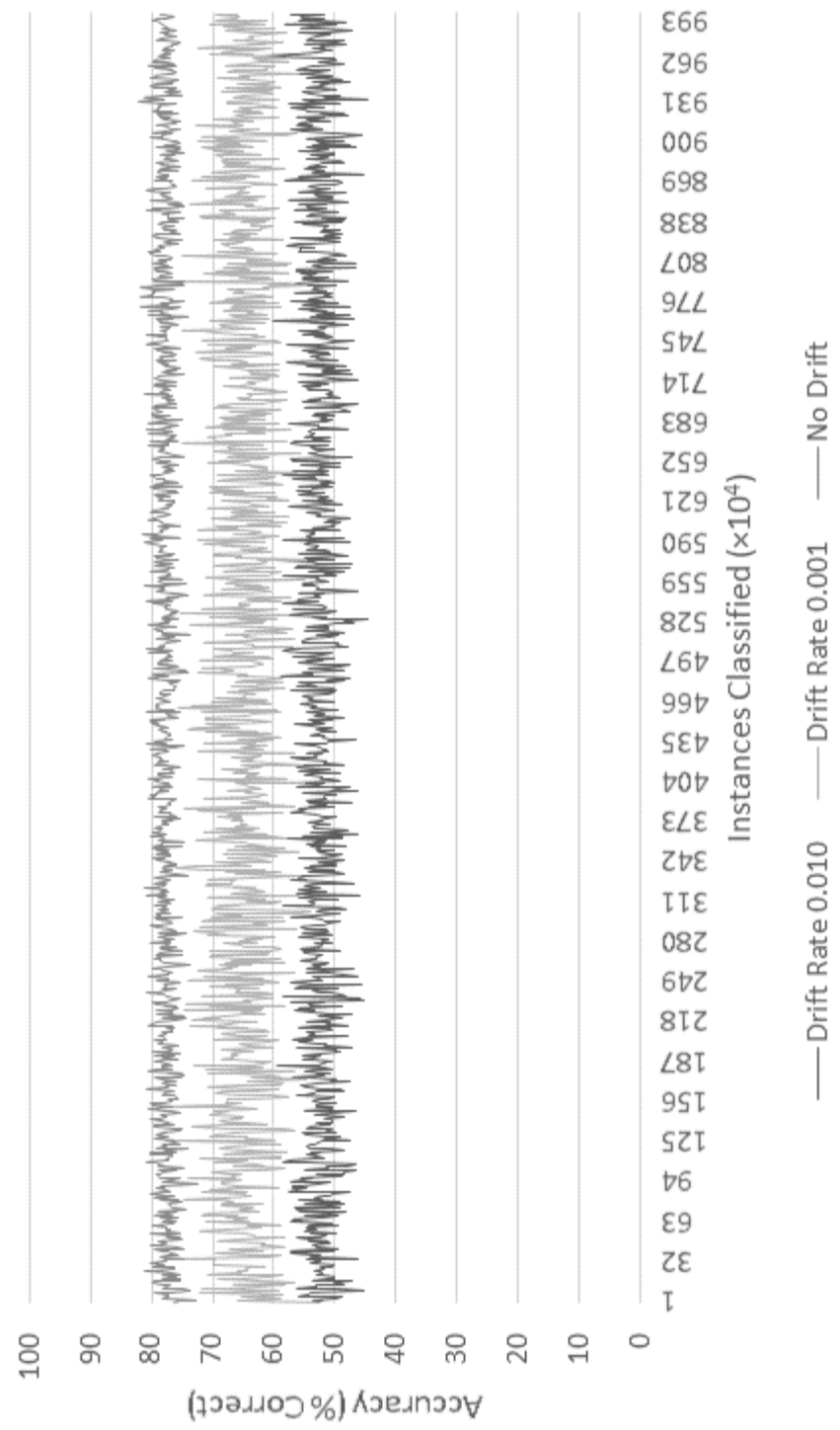


Figure C.11: Accuracy Weighted Ensemble Time Series (10D)

Accuracy Weighted Ensemble (50 Dimensional, Series D)

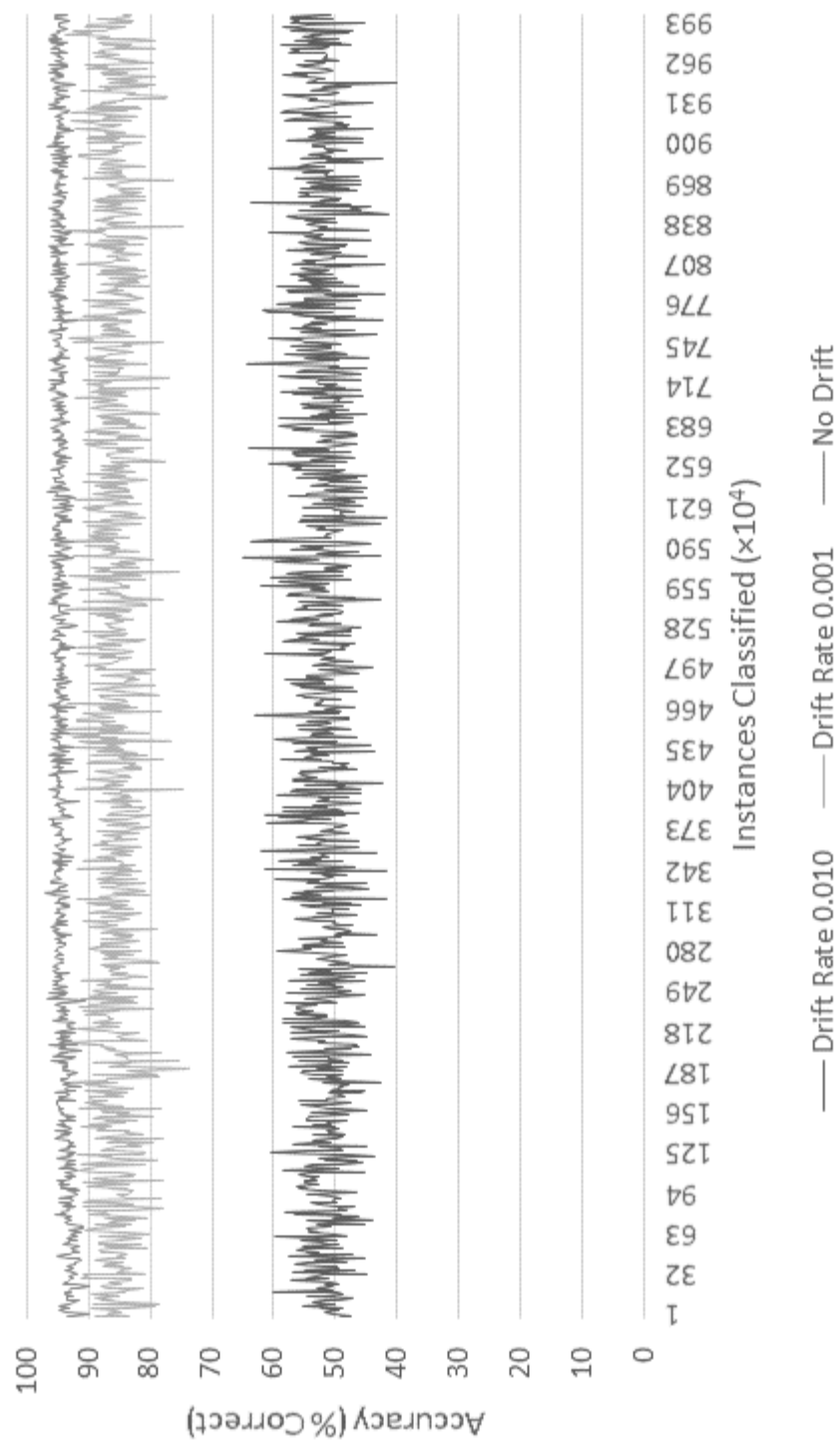


Figure C.12: Accuracy Weighted Ensemble Time Series (50D)

Accuracy Updated Ensemble (10 Dimensional, Series D)

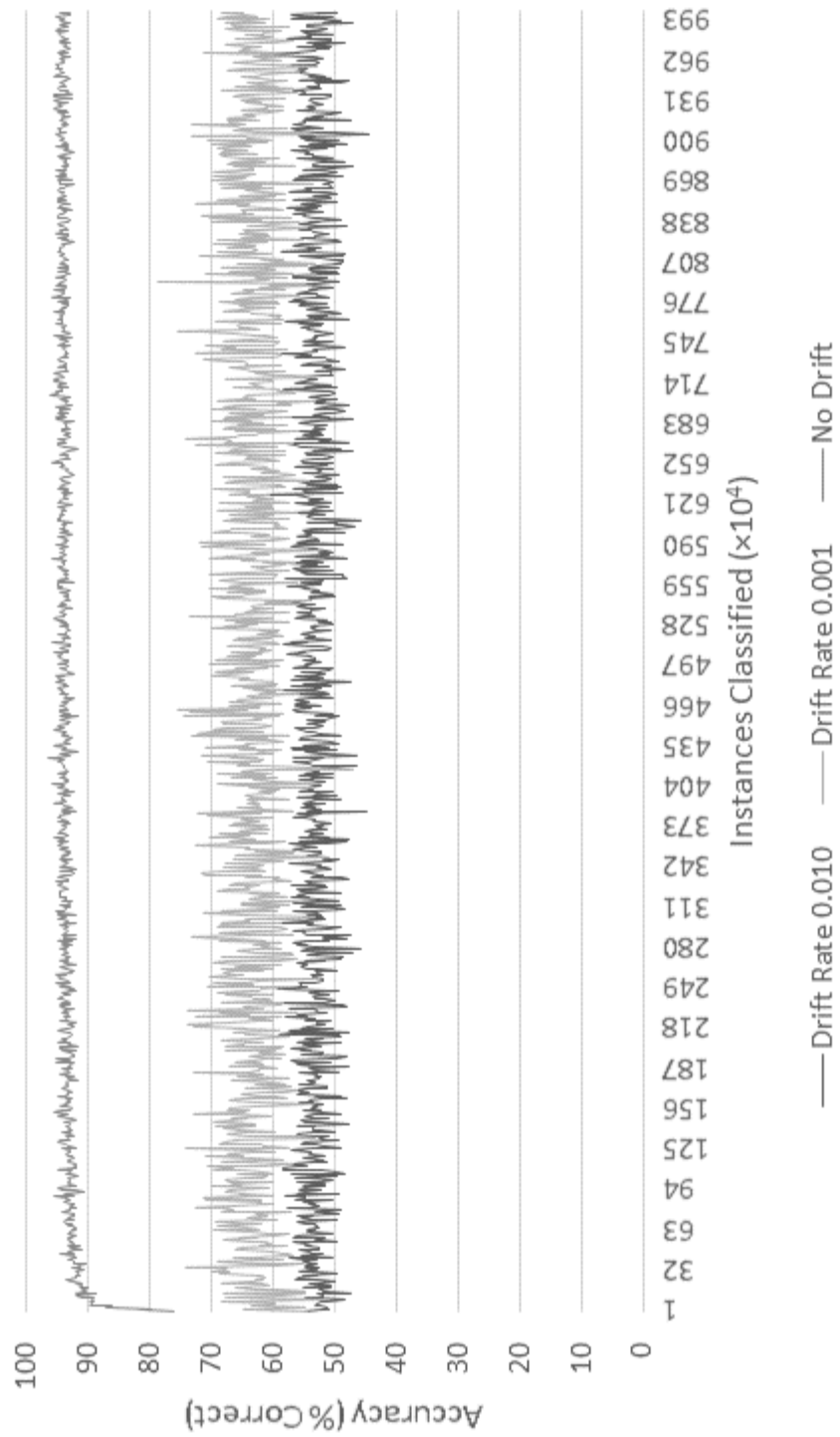


Figure C.13: Accuracy Updated Ensemble Time Series (10D)

Accuracy Updated Ensemble (50 Dimensional, Series D)

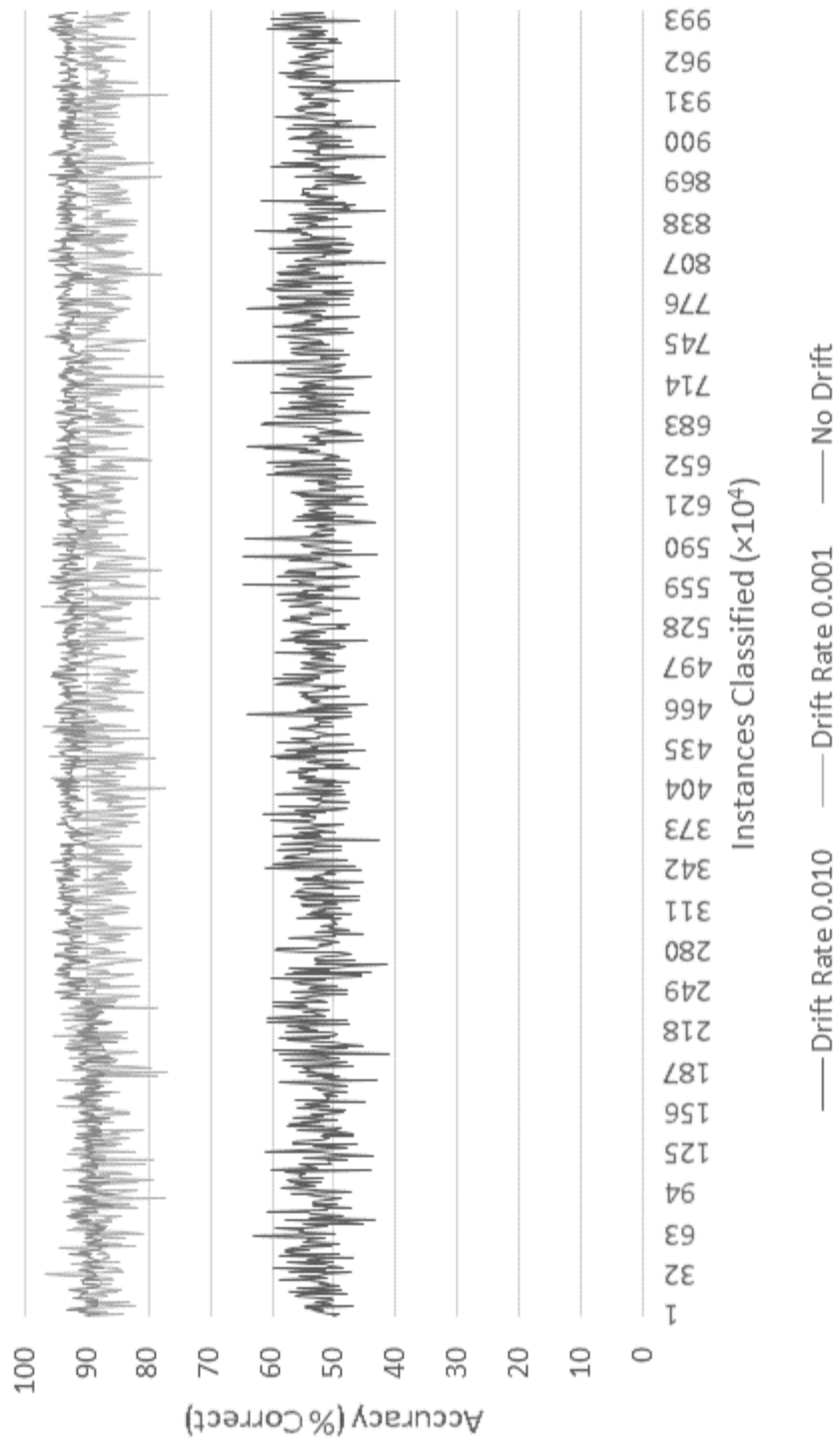


Figure C.14: Accuracy Updated Ensemble Time Series (50D)

References

- Abdualrhman, M. A. A., & Padma, M. C. (2015). Benchmarking concept drift adoption strategies for high speed data stream mining. In *Emerging Research in Electronics, Computer Science and Technology (ICERECT), 2015 International Conference on* (pp. 364–369). IEEE.
- Aggarwal, C. C. (2003). Towards systematic design of distance functions for data mining applications. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 9–18). New York, NY, USA: ACM. doi: <http://doi.acm.org/10.1145/956750.956756>
- Aggarwal, C. C. (2006). On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32nd international conference on very large data bases* (pp. 607–618). VLDB Endowment.
- Alexandridis, A., Sarimveis, H., & Bafas, G. (2003). A new algorithm for online structure and parameter adaptation of RBF networks. *Neural Networks*, 16(7), 1003–1018.
- Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., & Morales-Bueno, R. (2006). Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams* (Vol. 6, p. 77-86).
- Banker, R. D., Charnes, A., & Cooper, W. W. (1984). Some models for estimating technical and scale inefficiencies in data envelopment analysis. *Management Science*, 30(9), 1078–1092.
- Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36, 105-142.
- Becker, H., & Arias, M. (2007). Real-time ranking with concept drift using expert advice. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 86–94). New York, NY, USA: ACM. doi: <http://doi.acm.org/10.1145/1281192.1281205>
- Bifet, A. (2010). Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams. In *Proceeding of the 2010 conference on adaptive stream mining: Pattern learning and mining from evolving data streams* (pp. 1–212). Amsterdam, The Netherlands, The Netherlands: IOS Press.
- Bifet, A., & Gavaldà, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM international conference on data mining* (Vol. 7, p. 2007). SIAM.
- Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. *Advances in Intelligent Data Analysis VIII*, 249–260.
- Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., & Seidl, T. (2010). MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering.

- Bifet, A., & Kirkby, R. (2009). *Data stream mining a practical approach*.
- Bifet, A., Read, J., Zliobaite, I., Pfahringer, B., & Holmes, G. (2013). Pitfalls in benchmarking data stream classification and how to avoid them. In H. Blockeel, K. Kersting, S. Nijssen, & F. Zelezny (Eds.), *Machine learning and knowledge discovery in databases: European conference, ecml pkdd 2013, prague, czech republic, september 23-27, 2013, proceedings, part i* (pp. 465–479). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-40988-2_30 doi: 10.1007/978-3-642-40988-2_30
- Bishop, C. M. (2007). *Pattern Recognition and Machine Learning* (1st ed.) (No. 0). Springer.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5-32.
- Brown, G., & Kuncheva, L. I. (2010). “Good” and “bad” diversity in majority vote ensembles. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5997 LNCS, 124–133. doi: 10.1007/978-3-642-12127-2-13
- Brzezinski, D., & Stefanowski, J. (2014, January). Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1), 81–94. doi: 10.1109/TNNLS.2013.2251352
- Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2), 121–167.
- Chang, C.-C., & Lin, C.-J. (2011, May). LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), 27:1—27:27. doi: <http://doi.acm.org/10.1145/1961189.1961199>
- Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, 71–80. doi: 10.1145/347090.347107
- Dulhare, U. N., & Premchand, P. (2010). Discovery of Patterns Using DataStream Engine. *Computer Research and Development, International Conference on*, 0, 17–19. doi: <http://doi.ieeecomputersociety.org/10.1109/ICCRD.2010.39>
- Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.*, 15(1), 3133–3181.
- Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: a review. *SIGMOD Rec.*, 34(2), 18–26. doi: <http://doi.acm.org/10.1145/1083784.1083789>
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. *Advances in Artificial Intelligence—SBIA 2004*, 66–112.
- Gu, F., Zhang, G., Lu, J., & Lin, C.-T. (2016). Concept drift detection based on equal density estimation. In *Neural Networks (IJCNN), 2016 International Joint Conference on* (pp. 24–30). IEEE.
- Gu, X.-F., Xu, J.-W., Huang, S.-J., & Wang, L.-M. (2014). An improving online accuracy updated ensemble method in learning from evolving data streams. In *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on* (pp. 430–433). IEEE.
- Haque, A., Khan, L., Baron, M., Thuraisingham, B., & Aggarwal, C. (2016). Efficient handling of concept drift and concept evolution over Stream Data. In *Data Engineering*

- (*ICDE*), *2016 IEEE 32nd International Conference on* (pp. 481–492). IEEE.
- Janos, S. (2005). The algorithmicx package [Computer software manual]. szaszjanos@users.sourceforge.net.
- Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*(1), 59–69.
- Kolter, J. Z., & Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, *8*(Dec), 2755–2790.
- Krawczyk, B., & Woźniak, M. (2015). Reacting to different types of concept drift with adaptive and incremental one-class classifiers. In *Cybernetics (CYBCONF), 2015 IEEE 2nd International Conference on* (pp. 30–35). IEEE.
- Kuncheva, L. I. (2008). Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In *2nd workshop SUEMA* (pp. 5–10).
- Lampariello, F., & Scia drone, M. (2001). Efficient training of rbf neural networks for pattern recognition. *Neural Networks, IEEE Transactions on*, *12*(5), 1235–1242.
- Lavaire, J. D. D., Singh, A., Yousef, M., Singh, S., & Yue, X. (2015). Dimensional scalability of supervised and unsupervised concept drift detection: An empirical study. In *Big Data (Big Data), 2015 IEEE International Conference on* (pp. 2212–2218). IEEE.
- Loeffel, P.-X., Marsala, C., & Detyniecki, M. (2015). Classification with a reject option under Concept Drift: the Droplets Algorithm. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on* (pp. 1–9). IEEE.
- Maciel, B. I. F., Santos, S. G. T. C., & Barros, R. S. M. (2015, November). A Lightweight Concept Drift Detection Ensemble. In *2015 IEEE 27th international conference on tools with artificial intelligence* (pp. 1061–1068). IEEE. doi: 10.1109/ICTAI.2015.151
- Masud, M., Gao, J., Khan, L., Han, J., & Thuraisingham, B. M. (2010). Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, *99*(PrePrints). doi: <http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.61>
- Minku, L. L., White, A. P., & Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, *22*, 730–742. doi: <http://doi.ieeecomputersociety.org/10.1109/TKDE.2009.156>
- Oza, N., & Russell, S. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 359–364).
- Rodríguez, J. J., & Kuncheva, L. I. (2008). Combining Online Classification Approaches for Changing Environments. In *Proceedings of the 2008 joint IAPR international workshop on structural, syntactic, and statistical pattern recognition* (pp. 520–529). Berlin, Heidelberg: Springer-Verlag. doi: 10.1007/978-3-540-89689-0_56
- Ruiz, C., Menasalvas, E., & Spiliopoulou, M. (2009). C-denstream: Using domain knowledge on a data stream. In J. Gama, V. Costa, A. Jorge, & P. Brazdil (Eds.), *Discovery science* (Vol. 5808, p. 287–301). Springer Berlin / Heidelberg. (10.1007/978-3-642-04747-3_23)
- Sarnelle, J., Sanchez, A., Capó, R., Haas, J., & Polikar, R. (2015). Quantifying the limited and gradual concept drift assumption. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). IEEE.

- Schapire, R. E. (2003). Measures of Diversity in Classifier Ensembles. *Machine Learning*, 51(2), 181–207. doi: 10.1049/ic:20010105
- Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on machine learning* (pp. 807–814). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/1273496.1273598> doi: 10.1145/1273496.1273598
- Smith, M., & Ciesielski, V. (2016). Adapting to concept drift with genetic programming for classifying streaming data. In *Evolutionary Computation (CEC), 2016 IEEE Congress on* (pp. 5026–5033). IEEE.
- Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427–437.
- Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 226–235). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/956750.956778> doi: 10.1145/956750.956778
- Wang, W., Men, C., & Lu, W. (2008). Online prediction model based on support vector machine. *Neurocomputing*, 71(4), 550–558.
- Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition* (3rd ed.) (No. 0). Morgan Kaufmann.
- Wolpert, D., & Macready, W. (2005, December). Coevolutionary Free Lunches. *IEEE Transactions on Evolutionary Computation*, 9(6), 721–735. doi: 10.1109/TEVC.2005.856205
- Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1), 67–82.
- Zheng, Z., & Padmanabhan, B. (2007, November). Constructing Ensembles from Data Envelopment Analysis. *INFORMS Journal on Computing*, 19(4), 486–496. doi: 10.1287/ijoc.1060.0180
- Zliobaite, I., Bifet, A., Pfahringer, B., & Holmes, G. (2014, January). Active Learning With Drifting Streaming Data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1), 27–39. doi: 10.1109/TNNLS.2012.2236570
- Zliobaite, I., Kuncheva, L., & Others. (2009). Determining the training window for small sample size classification with concept drift. In *Data mining workshops, 2009. ICDMW'09. IEEE international conference on* (pp. 447–452). IEEE.