5-10-2010

An ensemble-based computational approach for incremental learning in non-stationary environments related to schema- and scaffolding-based human learning

Ryan Elwell

AN ENSEMBLE-BASED COMPUTATIONAL APPROACH FOR INCREMENTAL LEARNING IN NON-STATIONARY ENVIRONMENTS RELATED TO SCHEMA- AND SCAFFOLDING-BASED HUMAN LEARNING

by
Ryan Alexander Elwell

A thesis

Submitted in partial fulfillment of the requirements of the
Master of Science Degree
of
The Graduate School
at
Rowan University
January 13, 2010

Thesis Chair: Robi Polikar, Ph.D.

ABSTRACT

Ryan A. Elwell
AN ENSEMBLE-BASED COMPUTATIONAL APPROACH FOR INCREMENTAL LEARNING IN NON-STATIONARY ENVIRONMENTS RELATED TO SCHEMA- AND SCAFFOLDING-BASED HUMAN LEARNING
2008/09
Robi Polikar, Ph.D.
Master of Science in Engineering

The principal dilemma in a learning process, whether human or computer, is adapting to new information, especially in cases where this new information conflicts with what was previously learned. The design of computer models for incremental learning is an emerging topic for classification and prediction of large-scale data streams undergoing change in underlying class distributions (definitions) over time; yet currently, they often ignore significant foundational learning theory that has been developed in the domain of *human* learning. This shortfall leads to many deficiencies in the ability to organize existing knowledge and to retain relevant knowledge for long periods of time. In this work, we introduce a unique computer-learning algorithm for incremental knowledge acquisition using an ensemble of classifiers, Learn^{++}.NSE (Non-Stationary Environments), specifically for the case where the nature of knowledge to be learned is evolving. Learn++.NSE is a novel approach to evaluating and organizing existing knowledge (classifiers) according to the most recent data environment. Under this architecture, we address the learning problem at both the learner and supervisor end, discussing and implementing three main approaches: knowledge weighting/organization, forgetting prior knowledge, and change/drift detection. The framework is evaluated on a variety of canonical and real-world data streams (weather prediction, electricity price prediction, and spam detection). This study reveals the catastrophic effect of forgetting prior knowledge, supporting the organization technique proposed by Learn^{++}.NSE as the most consistent performer during various drift scenarios, while also addressing the sheer difficulty in designing a system that strikes a balance between maintaining all knowledge and making decisions based only on relevant knowledge, especially in severe, unpredictable environments which are often encountered in the real-world.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Dr. Robi Polikar for providing invaluable technical and intellectual support and encouragement throughout my education and research. Special thanks are also due to Michael Muhlbaier, who is responsible for the conception of the Learn^{++}.NSE algorithm, and to Matthew Karnick for his helpful assistance in Matlab coding and understanding fundamentals of our research. To my fellow graduate students in the Signal Processing & Pattern Recognition Laboratory and virtual reality labs, I thank you for your friendship, assistance, and generally for keeping me sane.

I thank my parents for their continued love and support in everything non-technical. I am also grateful for the Rowan University ECE department faculty. Specifically, the following deserve special thanks for showing their genuine care about my engineering future: Dr. Peter Jansson, Dr. Ravi Ramachandran, Dr. Linda Head, and Dr. Shreekanth Mandayam. I thank the Dean's Office staff and the College of Engineering faculty for their regard for my well-being which has extended even beyond my education, even to my personal needs.

Finally, I attribute all my accomplishments to my Father God in heaven, who has blessed me not only with success, but also with all my abilities. He gives me comfort not only in this life, but also thereafter because of His Son Jesus Christ, who has made the way for my salvation and eternal life through His life, death, and resurrection. Furthermore, by His Spirit, I have all the motivation to pursue research in engineering, for "I can do all things through Him who strengthens me" (Philippians 4:13).

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Incremental Learning 2

1.2 Motivation of Thesis: Non-Stationary Environments 3

1.3 Objective of Thesis: The Learn^{++}.NSE Algorithm 3

1.4 Scope of Thesis: Data with Non-Stationary Environments 4

1.5 Organization of Thesis 5

CHAPTER 2: BACKGROUND 6

2.1 The Problem of Concept Drift 6

2.2 Schema Theory for Knowledge Acquisition 11

2.2.1 Schemata Construction 13

2.2.2 Schemata Activation 15

2.3 Scaffolding Theory 16

CHAPTER 3: LEARNING IN NON-STATIONARY ENVIRONMENTS 18

3.1 Combining Human and Computational Intelligence Learning Models 18

3.2 Online & Batch Learning 24

3.3 Ensemble of Classifiers for Learning Concept Drift 25

3.3.1 Ensemble Organization & Evaluation (Weighting & Voting) 27

3.3.2 Controlled Forgetting (Ensemble Pruning) 30

3.4 Concept Drift Detection 34

3.4.1 Drift Descriptors 37

3.4.2 Measuring Drift 42

3.4.3 Problematizing (Windowing and Instance Selection) 45

CHAPTER 4: LITERATURE REVIEW **48**

4.1 Ensemble-Based Methods 48

4.1.1 FLORA. *48*

4.1.2 Streaming Ensemble Algorithm (SEA) *49*

4.1.3 Classifier Ensemble Approach for Mining Concept-Drifting Data *51*

4.1.4 Conceptual Clustering & Prediction *52*

4.1.5 Dynamic Weighted Majority (DWM) *53*

4.1.6 Knowledge-Based Sampling (KBS)-Stream Algorithm *55*

4.1.7 Adaptive Classifiers-Ensemble (ACE) *57*

4.2 Drift Detection Methods 60

4.2.1 Early Drift Detection Method *60*

4.2.2 Computational Intelligence-CUSUM (CI-CUSUM) *61*

4.3 Methods Related to Our Work 64

4.3.1 Adaboost *64*

4.3.2 Learn^{++} *66*

CHAPTER 5: THE LEARN^{++}.NSE ALGORITHM **69**

5.1 Algorithm Description 70

5.2 Controlled Forgetting 77

5.2.1 Permanent Pruning *77*

5.2.2 Temporary Pruning *80*

5.3 Drift Detection 82

5.3.1 Fading *89*

5.3.2 Dynamic Sigmoid Adjustment *89*

CHAPTER 6: EXPERIMENTATION AND RESULTS **94**

6.1 Motivation & Organization 94

6.2 Experimental Procedure & Performance Evaluation 96

6.3 Synthetic Data 97

6.3.1 *Random Gaussian Drift* ... *97*

6.3.2 *Triangular Gaussian Drift* ... *99*

6.3.3 *Random Gaussian Drift with Class Addition & Removal* ... *100*

6.3.4 *Non-Gaussian Drift (Checkerboard Dataset)* ... *101*

6.3.5 *Concept Change (Streaming Ensemble Algorithm Dataset)* ... *103*

6.4 Real-World Data ... 104

6.4.1 *New South Wales Electricity Dataset* ... *106*

6.4.2 *Nebraska Weather Dataset* ... *108*

6.4.3 *Spam Dataset* ... *111*

6.5 General Performance and Base Classifier Analysis ... 112

6.6 Classifier Ensemble vs. Single Classifier ... 120

6.7 Ensemble Weighting Comparison ... 121

6.8 Online vs. Batch Learning ... 130

6.9 Controlled Forgetting (Permanent) ... 135

6.10 Controlled Forgetting (Temporary) ... 142

6.11 Drift Detection ... 147

6.11.1 *Sigmoid Characterization for Appropriate Parameter Selection* ... *152*

6.11.2 *Scaffolding Techniques Using Drift Detection* ... *156*

CHAPTER 7: CONCLUSIONS ... 164

7.1 Contributions of This Work ... 164

7.2 Summary of Experimental Findings ... 167

7.3 Recommendations for Future Work ... 170

REFERENCES ... 172

APPENDICES **179**

Appendix A: Hellinger Distance for Synthetic Datasets 179

Appendix B: Weighting Method Comparison 182

Appendix C: DWM Characterization and Comparison 184

Appendix D: Pruning Characterization 185

Appendix E: Drift Detection 196

Appendix F: Sigmoid Characterization for Support Vector Machine (SVM) 200

LIST OF FIGURES

Figure 2.1: Breakdown of schema terminology 13
Figure 3.1: Computer learning model paralleling human learning theory 20
Figure 3.2: Correlation between schema and computational learning 23
Figure 3.3: Virtual concept drift due to insufficient data 36
Figure 3.4: False negative drift detection using $P(x)$ 42
Figure 4.1: Streaming Ensemble Algorithm (SEA) pseudocode 50
Figure 4.2: Conceptual Clustering & Prediction (CCP) pseudocode 52
Figure 4.3: Pseudocode for Dynamic Weighted Majority (DWM) Ensemble Construction 54
Figure 4.4: Knowledge-Based Sampling (KBS) Stream Algorithm Pseudocode 57
Figure 4.5: Pseudocode for Adaptive Classifiers Ensemble (ACE) algorithm 59
Figure 4.6: Data parameter log-likelihood CUSUM test for drift detection 62
Figure 4.7: Adaboost.M1 Algorithm 65
Figure 4.8: Learn^{++} Algorithm Pseudocode 67
Figure 5.1: Learn^{++}.NSE Algorithm Pseudocode 73
Figure 5.2: The effect of sigmoid error weight for a single classifier over time [78] 75
Figure 5.3: Time-based sigmoid error weighting 76
Figure 5.4: Permanent Controlled Forgetting 78
Figure 5.5: Temporary Controlled Forgetting 81
Figure 5.6: Performance-based CUSUM drift detection using log-likelihood 85
Figure 5.7: Drift detection procedure 88
Figure 5.8: Learn^{++}.NSE Fading pseudocode 89
Figure 5.9: Illustration of dynamic sigmoid cutoff parameter b 91
Figure 5.10: Dynamic sigmoid adjustment pseudocode 92
Figure 6.1: Classification Learning Scenario 95
Figure 6.2: Prediction Learning Scenario 96
Figure 6.3: Graphical representation of 4-class Gaussian drift 98
Figure 6.4: Graphical representation of 3-class triangular drift (single rotation) 99

Figure 6.5: Graphical representation of 4-class Gaussian drift with class addition/removal 100
Figure 6.6: Rotating checkerboard dataset (single rotation, $\alpha = 0\ to\ \pi$) 102
Figure 6.7: Variable α parameter for rotating checkerboard dataset 103
Figure 6.8: SEA Concepts (labeled testing data) 104
Figure 6.9: Percent class ("up") instances per window 107
Figure 6.10: Class-dependent Hellinger distance and drift rate with window size of 200 for electricity pricing data 108
Figure 6.11: Class-independent Hellinger distance and drift rate with window size of 200 for electricity pricing data 108
Figure 6.12: Seasonal observation of class balance (120 days per window) 110
Figure 6.13: Class-dependent Hellinger distance and drift rate with window size of 120 for weather data 110
Figure 6.14: Class-independent Hellinger distance and drift rate with window size of 120 for weather data 110
Figure 6.15: Percent of positive ("spam") class instances per window 112
Figure 6.16: Base classifier comparison for random Gaussian drift data 114
Figure 6.17: Base classifier comparison for triangular Gaussian drift data 114
Figure 6.18: Base classifier comparison for random Gaussian drift data with class addition/removal 114
Figure 6.19: Base classifier comparison for SEA concept data 114
Figure 6.20: Base classifier comparison for checkerboard data (constant drift) 116
Figure 6.21: Base classifier comparison for checkerboard data (pulsing drift) 116
Figure 6.22: Base classifier comparison for checkerboard data (exponential drift) 116
Figure 6.23: Base classifier comparison for checkerboard data (sinusoidal drift) 116
Figure 6.24: Base classifier comparison for weather prediction data, m=30 118
Figure 6.25: Base classifier comparison for electricity pricing prediction data, m=24 118
Figure 6.26: Base classifier comparison for spam prediction data, m=20 118
Figure 6.27: ANOVA comparison for base classifier analysis 119
Figure 6.28: Weighting comparison for random Gaussian drift data 122
Figure 6.29: Weighting comparison for triangular Gaussian drift data 122

Figure 6.30: Weighting comparison for random Gaussian drift data with class addition/removal....122
Figure 6.31: Weighting comparison for SEA concepts data....122
Figure 6.32: Weighting comparison for checkerboard data (constant drift)....123
Figure 6.33: Weighting comparison for checkerboard data (pulsing drift)....123
Figure 6.34: Weighting comparison for checkerboard data (exponential drift)....123
Figure 6.35: Weighting comparison for checkerboard data (sinusoidal drift)....123
Figure 6.36: Weighting comparison for weather prediction data m=30....124
Figure 6.37: Weighting comparison for electricity pricing prediction data, m=24....124
Figure 6.38: Weighting comparison for spam prediction data, m=20....124
Figure 6.39: ANOVA comparison for ensemble weighting methods (SVM)....126
Figure 6.40: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for triangular Gaussian drift data....128
Figure 6.41: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (constant drift)....129
Figure 6.42: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (pulsing drift)....129
Figure 6.43: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (sinusoidal drift)....129
Figure 6.44: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (exponential drift)....129
Figure 6.45: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for SEA concepts data....130
Figure 6.46: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for weather prediction data (training window: 2 weeks)....130
Figure 6.47: DWM weight characterization, random Gaussian drift data....131
Figure 6.48: DWM weight characterization, triangular Gaussian drift data....131
Figure 6.49: DWM weight characterization, random Gaussian drift data with class add/subtract....131
Figure 6.50: DWM weight characterization, SEA concepts data....131
Figure 6.51: DWM weight characterization, checkerboard data (sinusoidal drift)....132

Figure 6.52: DWM weight characterization, Weather prediction data (m=30) 132
Figure 6.53: DWM weight characterization, electricity pricing prediction data (m=24) 132
Figure 6.54: DWM weight characterization, spam prediction data (m=20) 132
Figure 6.55: ANOVA characterization for Dynamic Weighted Majority update period (Naïve Bayes) 134
Figure 6.56: Short-term memory (k=5) for random Gaussian drift data with class addition subtraction 136
Figure 6.57: Medium-term memory (k=25) for random Gaussian drift data with class addition subtraction 136
Figure 6.58: Long-term memory (k=100) for random Gaussian drift data with class addition subtraction 136
Figure 6.59: ANOVA comparison for random Gaussian drift data (class addition/removal) with varying ensemble size 136
Figure 6.60: Short-term memory (k=5) for SEA concept data 137
Figure 6.61: Medium-term memory (k=25) for SEA concept data 137
Figure 6.62: Long-term memory (k=100) for SEA concept data 137
Figure 6.63: ANOVA comparison for SEA concept data with varying ensemble size .. 137
Figure 6.64: Short term memory (k=5) for checkerboard data (constant drift) 138
Figure 6.65: Medium term memory (k=25) for checkerboard data (constant drift) 138
Figure 6.66: Long-term memory (k=100) for checkerboard data (constant drift) 138
Figure 6.67: ANOVA comparison for checkerboard data (constant drift) with varying ensemble size 138
Figure 6.68: Short term memory (k=5) for checkerboard data (pulsing drift) 139
Figure 6.69: Medium term memory (k=25) for checkerboard data (pulsing drift) 139
Figure 6.70: Long-term memory (k=100) for checkerboard data (pulsing drift) 139
Figure 6.71: ANOVA comparison for checkerboard data (pulsing drift) with varying ensemble size 139
Figure 6.72: Short term memory (k=5) for weather prediction data (m=30) 140
Figure 6.73: Medium term memory (k=25) for weather prediction data (m=30) 140
Figure 6.74: Long-term memory (k=100) for weather prediction data (m=30) 140

Figure 6.75: ANOVA comparison for weather prediction data (m=30) with varying ensemble size 140
Figure 6.76: Short term memory (k=5) for spam prediction data (m=20) 141
Figure 6.77: Medium term memory (k=25) for spam prediction data (m=20) 141
Figure 6.78: Long-term memory (k=100) for spam prediction data (m=20) 141
Figure 6.79: ANOVA comparison for spam prediction data (m=20) with varying ensemble size 141
Figure 6.80: Temporary pruning comparison for random Gaussian drift data 143
Figure 6.81: Temporary pruning comparison for triangular Gaussian drift data 143
Figure 6.82: Temporary pruning comparison for random Gaussian drift data with class addition/removal 143
Figure 6.83: Temporary pruning comparison for SEA concepts data 143
Figure 6.84: Temporary pruning comparison for checkerboard data (constant drift) 144
Figure 6.85: Temporary pruning comparison for checkerboard data (pulsing drift) 144
Figure 6.86: Temporary pruning comparison for checkerboard data (exponential drift) 144
Figure 6.87: Temporary pruning comparison for checkerboard data (sinusoidal drift) ..144
Figure 6.88: Temporary pruning comparison for weather prediction data (m=30) 145
Figure 6.89: Temporary pruning comparison for electricity pricing data prediction (m=24) 145
Figure 6.90: Temporary pruning comparison for spam data prediction (m=20) 145
Figure 6.91: ANOVA comparison for temporary pruning comparison (SVM) 147
Figure 6.92: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on SEA concepts data 150
Figure 6.93: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on checkerboard data (constant drift) 150
Figure 6.94: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on checkerboard data (pulsing drift) 151
Figure 6.95: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on checkerboard data (sinusoidal drift) 151
Figure 6.96: Time-weighting sigmoid characterization for random Gaussian drift data ($a = 0.5$) 153

Figure 6.97: Time-weighted sigmoid characterization for triangular Gaussian drift data ($a = 0.5$) 153
Figure 6.98: Time-weighted sigmoid characterization for random Gaussian drift data with class addition/removal ($a = 0.5$) 153
Figure 6.99: Time-weighted sigmoid characterization for SEA concepts data ($a = 0.5$) 153
Figure 6.100: Time-weighted sigmoid characterization for checkerboard data (constant drift) ($a = 0.5$) 154
Figure 6.101: Time-weighted sigmoid characterization for checkerboard data (pulsing drift) ($a = 0.5$) 154
Figure 6.102: Time-weighted sigmoid characterization for checkerboard data (exponential drift) ($a = 0.5$) 154
Figure 6.103: Time-weighted sigmoid characterization for checkerboard data (sinusoidal drift) ($a = 0.5$) 154
Figure 6.104: ANOVA comparison for time-weighted sigmoid characterization (b parameter, $a = 0.5$) (Naïve Bayes) 155
Figure 6.105: Scaffolding comparison for random Gaussian drift data with class addition/removal using Naïve Bayes 157
Figure 6.106: Scaffolding comparison for SEA Concepts data using Naïve Bayes 157
Figure 6.107: Scaffolding comparison for checkerboard data (constant drift) using Naive Bayes 158
Figure 6.108: Scaffolding comparison for checkerboard data (pulsing drift) using Naive Bayes 158
Figure 6.109: Scaffolding comparison for checkerboard data (sinusoidal drift) using Naive Bayes 158
Figure 6.110: Scaffolding comparison for electricity pricing data (m=15) using Naive Bayes 158
Figure 6.111: Scaffolding comparison for checkerboard data (constant drift) using SVM 159
Figure 6.112: Scaffolding comparison for checkerboard data (sinusoidal drift) using SVM 159

Figure 6.113: Scaffolding comparison for checkerboard data (pulsing drift) using SVM 159
Figure 6.114: Scaffolding comparison for SEA concepts using SVM 159
Figure 6.115: ANOVA comparison for scaffolding techniques using Naïve Bayes classifier 160
Figure 6.116: ANOVA comparison for scaffolding techniques using SVM classifier ... 162
Figure 7.1: Homology of supervised human learning and Learn^{++}.NSE computational learning 165

LIST OF TABLES

Table 6.1:Parametric equations for drifting Gaussian data....98
Table 6.2: Parametric equations for triangular Gaussian drift data....99
Table 6.3: Parametric equations for 4-class Gaussian drift with class addition/removal 101
Table 6.4: Learning scenarios for electricity pricing data....108
Table 6.5: Weather data measurements and event indicators....109
Table 6.6: Learning scenarios for weather data....111
Table 6.7: Dynamic sigmoid adjustment procedure....157

LIST OF NOMENCLATURE

General Nomenclature:

t: time index

x_t: feature vector

ω_t: class label

c: number of classes

$P_t(\omega_j | x)$: probability distribution of class j

$\mathfrak{D}_t$: training dataset $\{x_t, \omega_t\}$ drawn from $P_t(\omega_j | x)$

Learn^{++}.NSE Algorithm Nomenclature:

m_t: number of instances in training dataset $\mathfrak{D}_t$

D_t : penalty distribution

$\varepsilon_t(k)$: error of classifier k on training data $\mathfrak{D}_t$ (using penalty distribution)

σ_t: time-weighted sigmoid (slope parameter a, cutoff parameter b)

$\beta_t(k)$: normalized error of classifier k on $\mathfrak{D}_t$

$\bar{\beta}_t(k)$: time-weighted error of classifier k on $\mathfrak{D}_t$

$W_t(k)$: voting weight of classifier k at time t

Ensemble Pruning:

s: maximum ensemble size

$A_t(k)$: accuracy of classifier k on $\mathfrak{D}_t$

$A_t^{l,\alpha}$: lower confidence bound $(100(1-\alpha)\%)$ on classifier accuracy

Drift Detection:

CS: configuration size (number of batches) for drift detection

M: training size for drift detection classifier

P_θ: hypothesis for drift detection log-likelihood test (null: θ^0, alternative $\theta^{1,2,etc.}$)

R: Log-likelihood of null vs. alternative hypothesis

t_{drift} : time index for initialization of drift detection test

η: minimum observed log-likelihood

g: difference between current log-likelihood R and minimum η

h: threshold for drift detection test

Dynamic Sigmoid Adjustment:

b_o: optimality constant for sigmoid cutoff

b_s: stability constant for sigmoid cutoff

b_d: drift constant for sigmoid cutoff

ϕ_d: drift factor approaching optimality constant

ϕ_s: stability factor for approaching stability constant

γ: stability threshold

t_{stable} : number of time steps observed in testing phase before drift is detected

CHAPTER 1: INTRODUCTION

One of the primary and longstanding goals of computational intelligence has been to closely approximate brain-like decision-making in handling large-scale data. In this particular area, significant progress has been made over the last several years. Supervised learning is a common computational-intelligence model which, in certain aspects, resembles human-like intelligence, where a computer is trained with examples of both the representation (features) of a specific concept, as well as the correct (class) label for that concept. These features discriminate one class from another, and the classifier is able to classify new unlabeled data instances based on what has been learned. In its most primitive form, supervised learning is a one-time process in which the knowledge stored in a classifier is not adapted or supplemented after the training. For instance, learning the patterns of a typed character is an example of such a learning problem. Yet, in many applications, one-time learning is impractical as data are acquired in consecutive batches. Learning the same patterns for handwritten characters of different people or of different alphabets can be an example of a scenario where data become available in batches in an incremental fashion. Solving this problem by removing old knowledge and re-training on only the latest information would be wasteful, if not detrimental - if previously acquired knowledge is still relevant - whereas storing all prior data for re-training would be nearly impossible due to the steady increase of required memory and classifier training time. One possible solution to this dilemma is incremental learning, for which *ensemble-based* learning systems offer a natural solution by extending a single classifier model to a collection of models and enabling comprehension of complex and copious datasets.

1.1 Incremental Learning

In applications where data become available over a period of time, possibly in a streaming fashion, it is desirable to supplement a single computational model with the information acquired from newly-arriving data, thus creating a constructible knowledge base; this is commonly known as *incremental learning.*

Incremental acquisition of data leads to two design considerations for storing knowledge. The first consideration is related to the availability of prior data and is associated with the aforementioned shortfalls of single-classifier models. Memory and training limitations suggest that prior data should not be required to be used for future training after it is used once; therefore, the knowledge carried by such data must be stored in the knowledge base. The second concern pertains to the desire to maintain balance in learning novel information and retaining existing knowledge. Whether by design or by the sheer constraints of the classifier model, there is a tendency toward either (1) maintaining old knowledge at the cost of slow learnability of new information, or (2) focusing on learning the most recent information at the cost of forgetting old knowledge. This tradeoff has been coined as the *stability-plasticity dilemma* [1], where "stability" describes retaining existing knowledge, and "plasticity" refers to learning new knowledge. Incremental learning is defined as learning new information from additional data using an incrementally-updated model, where at any given time, the model has access only to the currently available data and the parameters of the existing model.

In addition to the stability-plasticity dilemma, there is an underlying fundamental assumption which renders basic incremental learning models short of true cognitive functionality – this is the assumption that all incoming (new) data are drawn from an environment where feature distributions (definitions for a particular class) are stationary. More formally, in a stationary environment, the new data are ensured to come from an unknown, but fixed distribution.

1.2 Motivation of Thesis: Non-Stationary Environments

The incremental learning problem can be extended to a case where the environment does not remain stable through time. In other words, the definition of a particular class may change at any point in time, rendering prior knowledge useless. *Concept drift* is the term which describes a change over time in probability distribution from which the features of a particular class are drawn. Such a change is characterized by a shift in the decision boundary between classes. An environment with data which undergo concept drift is called a *non-stationary environment.* Learning in a non-stationary environment adds another level of complication to the stability and plasticity dilemma, and is the main focus of this research.

1.3 Objective of Thesis: The Learn^{++}.NSE Algorithm

To address the problem of learning in non-stationary environments, we create a model based on the human learning theory (specifically schema theory and scaffolding theory) and draw parallels between theory for human learning and supervised computer learning. Schema theory describes a body of knowledge which is continually updated and modified as information is acquired through new experiences, operating under the assumption that current knowledge may conflict with what is to be learned from a new experience. Scaffolding theory describes the role of the supervisor in improving the learning process which involves monitoring both incoming data and learner performance in order to improve the learning process. The human learning model can be applied to formulate a set of guidelines for computational learning in non-stationary environments. This thesis describes an architecture known as the Learn^{++}.NSE algorithm, capable of (1) building a knowledge base from previously-learned data, (2) adding new knowledge trained on incoming labeled (classified) data, (3) identifying prior knowledge which is relevant in the current learning environment, (4) and using such knowledge to make predictions/interpretations of unlabelled data. Furthermore, the architecture can be augmented with a supervisor to detect changes in the environment to enhance the learning process.

1.4 SCOPE OF THESIS: DATA WITH NON-STATIONARY ENVIRONMENTS

The development of the Learn^{++}.NSE algorithm extends the capabilities of an incremental learner to handle large quantities streaming data, which are ever-prevalent and increasing in the world. Just as the human brain must adapt its knowledge over long periods of time as is encounters new or changing situations and experiences, so must a large-scale computational learner expect to encounter environments which undergo change. The adaptive knowledge architecture proposed in the Learn^{++}.NSE algorithm is capable of handling a variety of concept drifts or changes which are encountered in large scale data streams. Learn^{++}.NSE is unique in its ability to use prior knowledge in scenarios where environments are recurring. Thus, it is designed specifically to make the best possible prediction of unknown data within the newest environment. In this work, we present and evaluate Learn^{++}.NSE algorithm with respect to its ability to (1) incrementally build up a knowledge base (ensemble of classifiers) with incoming data, (2) organize and utilize prior knowledge according to its relevance in the current environment regardless of when the knowledge was created, (3) forget prior knowledge to increase plasticity (at the cost of stability); and (4) be adapted to actively detect changes in the environment to improve the learning process and dynamically alter learning parameters when drift occurs. The aforementioned behaviors are observed in both synthetic and real-world concept-drifting datasets with varying complexity.

1.5 ORGANIZATION OF THESIS

Chapter 2 details the background of the engineering basis and philosophical model for learning in non-stationary environments. Chapter 3 ties the human and computer learning models together into a set of guidelines for learning and provides a survey of computer learning approaches that meet such guidelines. Chapter 4 is a literature review of algorithms designed for learning in non-stationary environments as well as ensemble-based algorithms which lay the foundation for our incremental learning algorithm. Chapter 5 describes an adaptation of the algorithm for non-stationary environments, named Learn^{++}.NSE, which is designed specifically to meet the requirements set forth by the model from Chapter 3. Discussion of results based on both synthetic and real-world experiments can be found in Chapter 6. Finally, a summary of conclusions and suggestions for future work are laid out in Chapter 7.

CHAPTER 2: BACKGROUND

In this chapter, we introduce the fundamental problem of learning in a non-stationary environment where the information to be learned is changing over time. The *cause* and *perception* of this change is convoluted, and will be discussed in terms of both the human and computer learning approaches. Traces of the relationship between human and computer learning models can be seen in prior research; however, few parallels have been drawn between any particular human cognitive model and a computer model designed for changing environments. We seek to establish this connection with computational learning as we discuss the principles wrought from the field of human learning psychology. These elementary principles are then translated to form functional guidelines for non-stationary learning, around which framework we can both interpret and build upon novel computer models.

2.1 THE PROBLEM OF CONCEPT DRIFT

The fundamental problem with many incremental learning environments is change. This is not simply additional or complementary knowledge used to build upon an environment's description. Rather, the problem of a *concept-drifting* environment is one in which previously-seen concepts or class definitions have innately changed so much in their distribution that it results in conflict between current and prior definitions. Not only may class-definitions be changing, but they may be evolving such that the true decision boundary between them changes to accommodate the new definitions. Such an environment is known as a *non-stationary* environment.

The following is a comprehensive description of concept drift terminologies. Throughout this discussion, we will use the real-world example of rain classification ("is it raining today?") based on a real-time weather measurements (features) such as temperature, humidity, etc. Also, we consider these definitions in the context of a particular time of the year (the *environment*). First, we pose a fundamental classification question as follows: "Given that the temperature is $65°F$ and that humidity is 80%, what is the probability that it is raining?" We define *concept drift* as a situation where the likelihood of a class (like rain) for a specific feature set/definition (like temperature) is increasing or decreasing over time.

The fundamental classification question can be formalized using the Bayes probability theory $P(\omega|x) = \frac{P(x|\omega)P(\omega)}{P(x)}$ for the posterior probability of observing a class ω given a feature x. The probability $P(\omega|x)$ is dependent on $P(x|\omega)$, the likelihood or conditional probability of class ω given an observation x; $P(\omega)$, the prior probability of any class being observed; and $P(x)$, the feature-dependent probability (evidence). Formally, concept drift is defined as any case where the posterior probability changes over time, that is, $P(\omega|x)_{t+1} \neq P(\omega, x)_t$. A more in-depth look at this definition is key to understanding the different aspects of the definition of concept drift. It also provides important context to the task of concept drift *detection*.

The term $P(x)$ describes the feature-based probabilities of the data and answers the question, e.g., "What is the probability of observing a temperature of $65°$?" Observing $P(x)$ over time allows us to see general changes in the environment.

However, an observation of change in $P(x)$ is an insufficient indicator of true concept drift (shift in decision boundaries) because of its independence of the class labels. For example the probability of observing 65° may not change, and yet we may find that the class labels are changing depending on the time of year (e.g. in August, 65° may corresponds to rain and in February it may correspond to no rain). This is not to say that $P(x)$ cannot be an indicator of concept drift, for a change in overall distribution of the features often means that the true decision boundaries are shifting as well.

The term $P(x|\omega)$ describes the conditional probability of observing a feature x within a particular class ω. Using our weather example, this would correspond to asking the question: "provided that it is raining, what is the probability that the temperature is 65°?" This likelihood measurement is a data-dependent probability and is governed by the data instances which have been seen in the past; these are the same instances that are used to train a classifier and hypothesize a decision boundary between classes. A shift in likelihood would seem to indicate that the class labels are changing in some way. For instance, provided that it is raining, the probability of observing a temperature of 65° may be different in February than it is in August. We will later see that this is an example of true concept drift, but before making a general assumption, we assert that it is not until the distribution of one class shifts such that the true class boundaries are altered that we can call it real concept drift. Class drift *without* overlapping of true class boundaries is known as *virtual* concept drift [2], and merely shows that the learner is being provided with additional data from the *same* environment. Virtual drift is the result of an incomplete representation of the true distribution in the current data. The key difference here is that real drift requires replacement learning (old knowledge is irrelevant) whereas virtual drift requires supplemental learning (adding to the current knowledge).

The final term, $P(\omega)$, defines the class-based probabilities, that is, "what is the probability that it will rain?" This term relates class balance to the overall distribution. With regard to concept drift, we first note that since there is no relation to the features, observing $P(\omega)$ does not reveal information about the decision boundaries between classes. Yet it does reveal another fundamental aspect of non-stationary environments dealing with class imbalance. Real-world data streams are seldom balanced with regard to class occurrences, and this balance may in fact change over time. For example, the number of rain occurrences may change over the course of a year. Class imbalance is known to negatively impact classification performance ([3-6]), and is an inherent incremental learning problem that needs to be addressed.

We see that these individual probabilities ($P(x|\omega)$, $P(\omega)$, and $P(x)$) are, in certain cases, insufficient in explaining true drift in the class definitions at the time in which it is occurring. In conclusion, since we cannot truly know whether or not real drift has occurred until after the fact (when the true distribution has been observed), we must therefore consider each of these symptoms separately and use them as tentative indicators of drift. Section 3.4 provides an in-depth look at the aspect of concept drift detection techniques, in which we revisit terms such as virtual, real, and perceived drift.

Concept drift can be viewed in a more abstract sense as an obstacle caused by insufficient features in a given dataset. This is often called a *hidden context*. That is to say, there is a common thread within the data that provides a true and static description over time for each class which is hidden from the learner's view. Using the weather example, this could be exemplified as an additional feature (e.g. air pressure) which provides a consistent discrimination between classes, but is unavailable to the learner at the time of learning. Viewing the problem with the benefit of this (hidden) context would mitigate non-stationarity. Yet, the learner must cope with what information is available, and for this reason, the formalized definition of concept drift allows for a better understanding of the perspective from which we are viewing the information to be learned.

Having laid out the probabilistic framework of concept drift, we now discuss additional complexities of how drift occurs within an environment. Minku [7] and Kuncheva [8;9] also provide comprehensive summaries which organize the types of concept drift that are encountered. In general, drift may be characterized with respect to its *speed*, *randomness*, and *cyclical* nature.

Drift speed describes the displacement in the underlying class distributions $P_t(\omega|x)$ from one step in time to the distribution in the following time step $P_{t+1}(\omega|x)$. Larger displacement denotes fast drift and results in high classifier error; gradual drift appears with smaller displacement, and, although it may be difficult to detect, it will result in low classification error.

Drift randomness is an important descriptor in discerning between non-stationary data and noisy data, and can be best described as the variance of a distribution over a short period time. Randomness can be viewed in terms of its frequency and magnitude. High variance between two periods of time corresponds to a highly unstable environment which, as this level increases, approaches a state where the environment cannot be learned.

The cyclical nature of drift is a phenomenon which can be observed in many real-world data patterns (weather measurements, electricity demand). In such cases, class definitions change such that a previous environment recurs after some period of time. This recurrence can be periodic or random.

We now shift our focus from the "probabilistic" machine learning model to a "psychological" human learning model with a goal of making connections between the two.

2.2 SCHEMA THEORY FOR KNOWLEDGE ACQUISITION

Knowledge acquisition is a fundamental aspect of human cognition and learning; it follows that making a connection between computational intelligence and human learning could be beneficial in providing suggestions and guidelines for constructing a computational knowledge base. This connection is especially beneficial as we consider concept drift, for the brain is often confronted with new environments containing information which conflicts with its prior knowledge or experience. Consider the following example involving an individual's experience with a bookstore. In the first scenario, the individual visits a local bookstore and has an experience that is consistent with their prior experiences, and his definition of a bookstore remains unchanged; this experience could be said to occur within a stationary environment. In a second case, the person visits a new bookstore which has coffee bar and video rental department. There is a mild conflict between prior knowledge about bookstores and the individual's current experience, so the he must adapt his bookstore definition to include the new experience. The modification in definition can be equated with *concept drift* in computational learning. In the third case, the individual encounters an internet-based bookstore. This experience is so drastically different from his prior definition of a bookstore that he may need to completely redefine the term "bookstore" in his mind. In terms of machine learning, such an occurrence would be best described as a *concept change*. We see that in both machine learning and human cognition, there is an intrinsic need to build up a body of knowledge and interpret new experiences in consideration of prior knowledge.

Jean Piaget, a 20th century Swiss psychologist, revolutionized thinking in the area of human cognition as he developed a structure for human memory known as the *equilibration* model. Equilibrium is a term which describes a balance of state, and applies to many scientific processes in chemistry, biology, and physics. Piaget asserts that this also applies to human cognition, describing the learning process as a constant effort to maintain or achieve equilibrium between prior knowledge and new knowledge [10;11]. The model is broken down to form a foundational theory for both child and adult learning and has been extensively researched, specifically with regard to a subcategory of Piaget's theory known as *schema* [12].

The term *schemata* applies to a body of knowledge or descriptors (*schema*) that is built and adapted over time in the human brain. We also use the term "knowledge base." Schema theory is a psychological model formulated to describe the process of human knowledge acquisition and memory organization for future decision-making. This section explains the key concepts of schema theory which allow us to make connections to a computational intelligence model for learning in a non-stationary environment.

The properties of schema theory can be broken into two categories: *construction* (building and categorizing the knowledge base) and *activation* (utilization of schema to

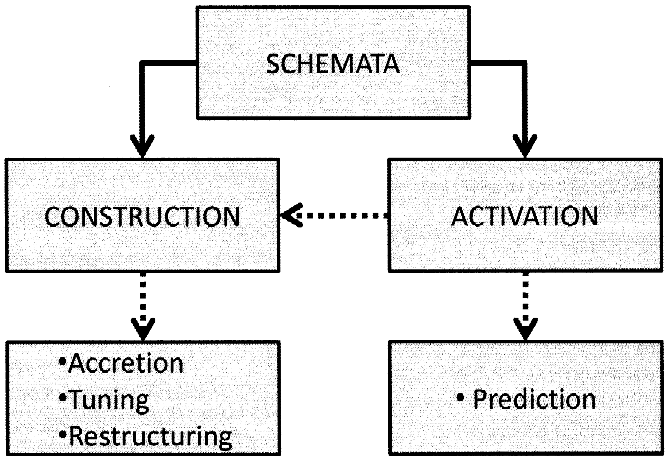


Figure 2.1: Breakdown of schema terminology

interpret unknown information) [13;14]. The subdivision of this terminology is illustrated in the block diagram in Figure 2.1 and described in the following sections.

2.2.1 Schemata Construction

Schemata **construction** describes the process of an adaptive knowledge base that can change and grow with new information. The human brain has a unique capability not only to collect new information, but also to identify important aspects of the information such that it can be summarized and categorized appropriately. Yet, not all information is accepted the same way; this is because not all new information is consistent with the brain's prior knowledge. These conflicts are the building blocks of human (and computer) learning. Piaget uses the terms *assimilation* and *accommodation* to describe the natural process of human data acquisition; these can be broken into three more descriptive terms: accretion, tuning, and restructuring.

Accretion describes the situation where information is remembered or interpreted in the context of existing schema. Accretion occurs when new information is agreeable with the current body of knowledge. In the bookstore example, accretion is described in the first scenario, where the individual enters a normal bookstore which is consistent with his previous bookstore experiences. Differences between incoming information and existing (prior) knowledge will often necessitate ***tuning*** of the schema. Here, the schemata evolve in order to accommodate the new information or experience when there is little conflict between the new and old, as described by the bookstore example when the individual must add the experience of a coffee bar and film section to his knowledge about bookstores. When new knowledge cannot be accommodated under the schema structure because of severe conflict, the result is a ***restructuring*** process in which new schemata are created to supplement or replace the prior knowledge base. Restructuring is exemplified in the final bookstore scenario where the individual encounters an internet-based bookstore which conflicts with all prior experiences.

Piaget stresses the importance of maintaining a balance among these modes of acquisition. This balance is known as *equilibration*, where the learner must be able to both focus on the current environment and give sufficient consideration to prior knowledge. We note that this issue of balance has also been addressed in machine learning under the name of the *stability-plasticity dilemma* [1].

The aforementioned schemata construction terminology defines an intensive process of constant memory organization which allows the brain to be extremely efficient in identifying, learning, and reinforcing important concepts. It is beyond our understanding how the brain is able to adaptively arrange information in such a concise and meaningful way, especially given the astronomical amount of data (sensory, auditory, visual, etc.) that is available at any given time. What is important to realize is that in order for such a level of categorization to take place, the brain must constantly be activating and evaluating its current knowledge base throughout the learning process.

2.2.2 Schemata Activation

Schemata **activation** occurs for two purposes. First, schemata are activated during the data acquisition process in order to determine which type of schemata construction (tuning, accretion, restructuring) should take place. Here, the current knowledge base must be compared to new knowledge in order to make connections and determine its adequacy to handle or understand the new information. Memory-searching enables constant memory evaluation to determine which knowledge is relevant within the current learning context [15]. It is difficult to know the extent to which memory is searched at any given time in the human brain, since we only seem to be conscious of the relevant memory.

Secondly, schemata are activated for the purpose of prediction and extrapolation. This type of activation enables the brain to interpret unknown data and predict situations; naturally, this extrapolation is based on current schemas which are relevant to a particular situation. Not only can the brain interpret novel information, but it can also hypothesize about missing material within its own knowledge base making the brain to be robust in the presence of structural damage which leads to memory loss.

2.3 Scaffolding Theory

Scaffolding is a tutoring theory developed to enhance human learning of complex data, and is primarily based on a theory developed by Russian psychologist Lev Vygotsky [16]. Scaffolding is essentially a supervised learning approach to building schemata in the most effective way possible by breaking up information such that it is learned in chunks (not learning everything at once), and by periodically intervening in order to evaluate performance within the scope of the most recent information. Considering the complexity of changes which the brain must accommodate in learning, the addition of a teacher/tutor in the learning process is especially beneficial. In expounding on some of the fundamental goals in scaffolding, we will see a close relation between scaffolding and schema theories. At the same time, we draw parallels to supervised computational learning in complex, non-stationary environments. Unless otherwise prompted, the supervisor simply provides the classifier(s) with feature-label training combinations as data become available, giving no insight as to the data examples that are being presented. Using scaffolding theory, we seek to enhance the teaching end of supervised learning.

The goal of scaffolding is to provide a learner with both *feedback* and *guidance*. The supervisor acts as a filter between the learner and the information to be learned. Vygotsky theorizes an ideal learning environment known as the *zone of proximal development* and asserts that providing a learner with tasks slightly harder than what it has already learned is optimal for accumulating knowledge [16]. This approach requires a balance in the flow of information in which new data is neither too difficult to learn nor redundant or time-wasting. Balance is achieved by *complexity reduction, problematizing,* and *fading*.

Complexity reduction is an important aspect of supervision, and requires the tutor to control the flow of information such that it is broken up into learnable pieces. This applies especially to complex concepts. One aspect of complexity reduction is to discern between important information and irrelevant (noisy) information. An additional role of the supervisor is to foster schema activation and construction by (1) making connections to prior knowledge, (2) drawing out conflicts between new information and the learner's knowledge, and (3) forcing the learner to make decisions and extrapolations, especially in areas where there are conflicts. The emphasis of conflicting information for training is known as **problematizing** [17], and, as was stated earlier, is the key to incremental learning and schema development.

The final aspect of effective scaffolding-based supervision is known as **fading** [17-19]. Not only is the supervisor responsible for filtering out irrelevant data, but it should also filter out superfluous data. At some point, the learner will acquire sufficient knowledge about some concept; it is then that the supervisor shall cease the teaching process. The amount of fading is proportional to or contingent upon the learner's competence. This presupposes that the supervisor is involved in periodic performance evaluation so that progress can be monitored.

CHAPTER 3: LEARNING IN NON-STATIONARY ENVIRONMENTS

Having properly defined the problem of concept drift, which traverses both human and computational learning, we now discuss general approaches for incrementally learning in a non-stationary environment. We begin with establishing specific goals and constraints based on the schema and scaffolding theories for human data acquisition as discussed in the previous chapter. We then provide a series of guidelines which translate the human learning model from theory to computer-based learning in terms of classification in non-stationary environments. The chapter concludes with a survey of specific computational approaches to learning with concept drift which stem from the proposed guidelines.

3.1 Combining Human and Computational Intelligence Learning Models

Piaget's and Vygotsky's philosophical advancements apply generally to human learning (regardless of age) and have been pointedly applied to childhood cognitive development. Now, we seek to draw parallels between the model for human cognition and a model for computer cognition and knowledge acquisition.

The schema and scaffolding theories described in Chapter 2 provide us with guidelines for setting up a computer-based model for classification in a changing environment. Formally, we describe data acquisition in terms of samples being obtained from a general data distribution or environment $\mathfrak{D}$ with class probabilities described by $P(\omega_j|x)$ with j classes described by representative feature vectors (instances) x. A stream of such instances of size m is made available to the learner at time t which is a sample or perception of the true distribution for the training data set, $\mathcal{D}_t$.

Using schema and scaffolding theory as well as some suggested rules provided by Kuncheva in [8] for incremental computational learning, we propose the following constraints and guidelines for learning in non-stationary environments:

- An instance of data $x_t(i)$ can only be seen once for the purpose of training, and therefore knowledge from each data instance must be generalized/summarized or stored in some way in the model parameters for future use.
- Knowledge should be categorized with respect to relevance to the current environment; knowledge should be dynamically updated when new training data is presented, since the most recent dataset is a representation of the current environment.
- The learner should be capable of identifying unlearned data within a training data set $\mathfrak{D}_t$ at time t.
- Knowledge should be incrementally and periodically stored so that it can be activated to produce the best possible hypothesis for an unknown piece of information at any particular time in the learning process.
- It is appropriate to have an exterior "supervisor" to assist the learner by monitoring both the incoming data and the learner's performance for the purpose of *complexity reduction*, *problematizing,* and/or *fading*.

The computer and cognitive models can be molded together, forming a practical model for computer learning that is guided by the aforementioned learning principles. Figure 3.1 illustrates our proposed combined model using both computational incremental learning and schema terminology. At each time t, we obtain training samples $\mathfrak{D}_t$ from an unknown distribution P_t which can appear as either training (labeled with class ω_j) or testing (unlabelled) data. Training data is used to construct the knowledge base (or schemata), and testing data is used for evaluation (activation) of the knowledge base. The same data can serve both purposes, where new data is first used to test the learner's performance, and then used to train the learner.

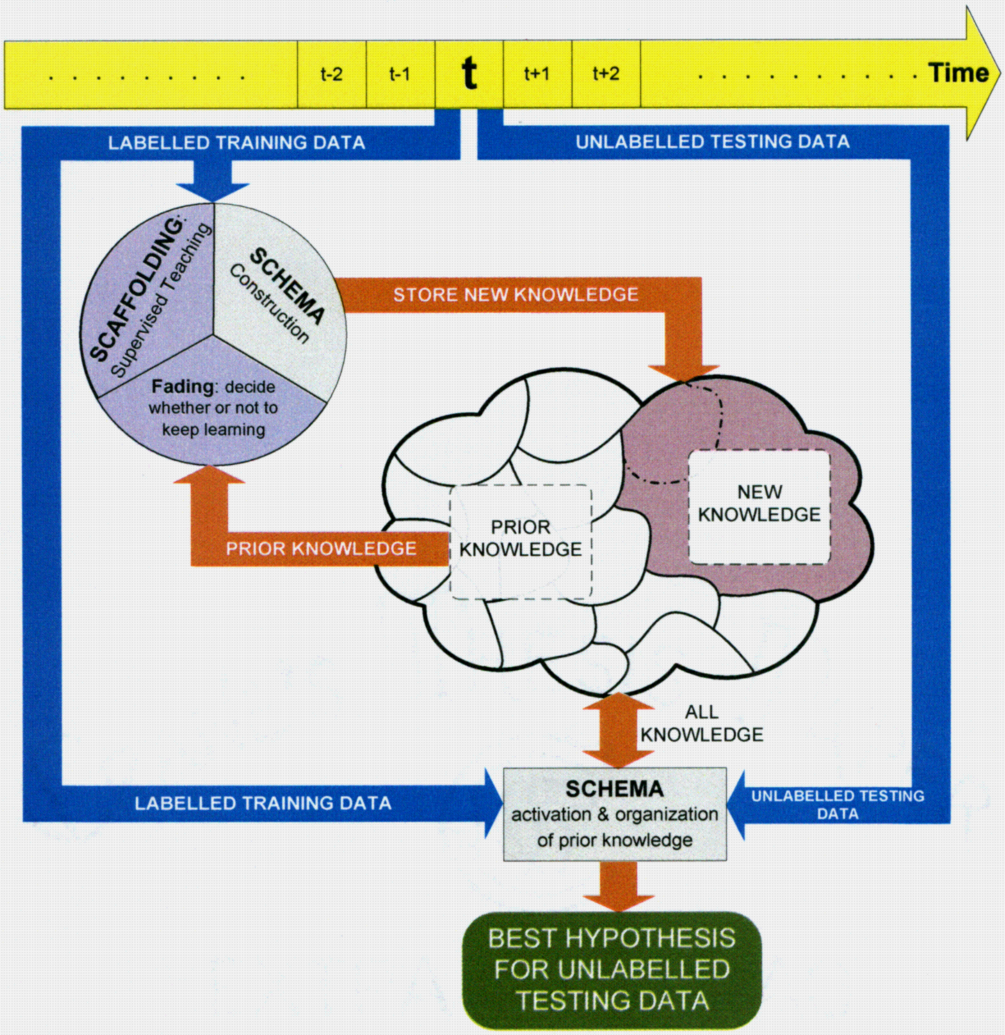


Figure 3.1: Computer learning model paralleling human learning theory

The knowledge base, or schemata, consists of representations of the data that have been seen during training. Knowledge is stored in the form of one or more classifiers, specifically the (adjustable) parameters of the classifiers which, in accordance with schema theory, provide a generalization of the input data by storing discriminatory information for separate classes.

In machine learning, various computer classifier models have been introduced for the purpose of representing/generalizing data. Classifier models are divided into two categories: generative or descriptive. *Generative* classification models seek to use the training examples to approximate the parameters of an assumed distribution. The most common example is the Naïve Bayes classifier, which uses the mean and variance of each class in the training data to characterize a Gaussian distribution from which probabilities are created. The weakness of generative models lies in the assumption of the distribution from which the data are drawn. *Descriptive* models, however, operate under no assumptions about the distribution $P(\omega|x)$. Instead, they create a decision boundary using a set of rules organized as a decision tree (CART, C45), map input features to output classes using back propagation algorithms (Multi-Layer Perceptron), creating a decision boundary that maximizes the margin between classes (Support Vector Machine), or some other classification algorithm. All of these methods are based solely on the data itself, and, although they require some tuning parameters, they are universal classifiers independent of the data distribution.

The computer learning model in Figure 3.1 depicts two ways in which the knowledge base (classifier) is activated. First, knowledge is activated in the context of schema construction. In order to determine which knowledge is relevant to the current context or environment, current classifiers are tested on incoming training data. The hypothesis $H_t(x)$ produced by the current knowledge is compared to the true class labels ω in order to identify information which has not been previously learned. If knowledge is stored in multiple classifiers, individual hypotheses can be used to identify which classifiers are most competent on the current environment. The second form of activation takes place with testing data, where labels are truly unknown at the time.

As we consider how to apply the proposed computer learning model in practical computer learning applications, we segment the discussion into the following topics: (1) methods of constructing the knowledge base in the form of one or more classifiers, (2) knowledge activation and organization, and (3) change detection techniques. The connection between the previously described learning theory and specific computer learning approaches to be discussed are overviewed in Figure 3.2.

Construction of the knowledge base is partially dependent on how data examples are obtained, either in an instance-by-instance (online) basis or batch-based approach. The flow of arriving data may be constrained by the source of data or may be controlled by a supervisor if scaffolding techniques such as dynamic windowing or instance selection are applied.

Organization of the knowledge base is often accomplished through a technique known as ensemble weighting, where classifiers are categorized according to their relevance at the current time. The basis for this approach is the realization that not all knowledge may be relevant at a given time. An alternative or complementary approach to organizing knowledge with weighting is known as ensemble pruning or controlled forgetting, where irrelevant knowledge is either permanently removed (forgotten) or temporarily ignored. Most organization techniques are related to classifier performance, and therefore necessitate knowledge activation.

Change detection methods are a means of determining the occurrence(s) of concept drift in the perceived class distributions. Scaffolding techniques such as fading are made possible through drift detection. Change may be monitored in various ways; the two

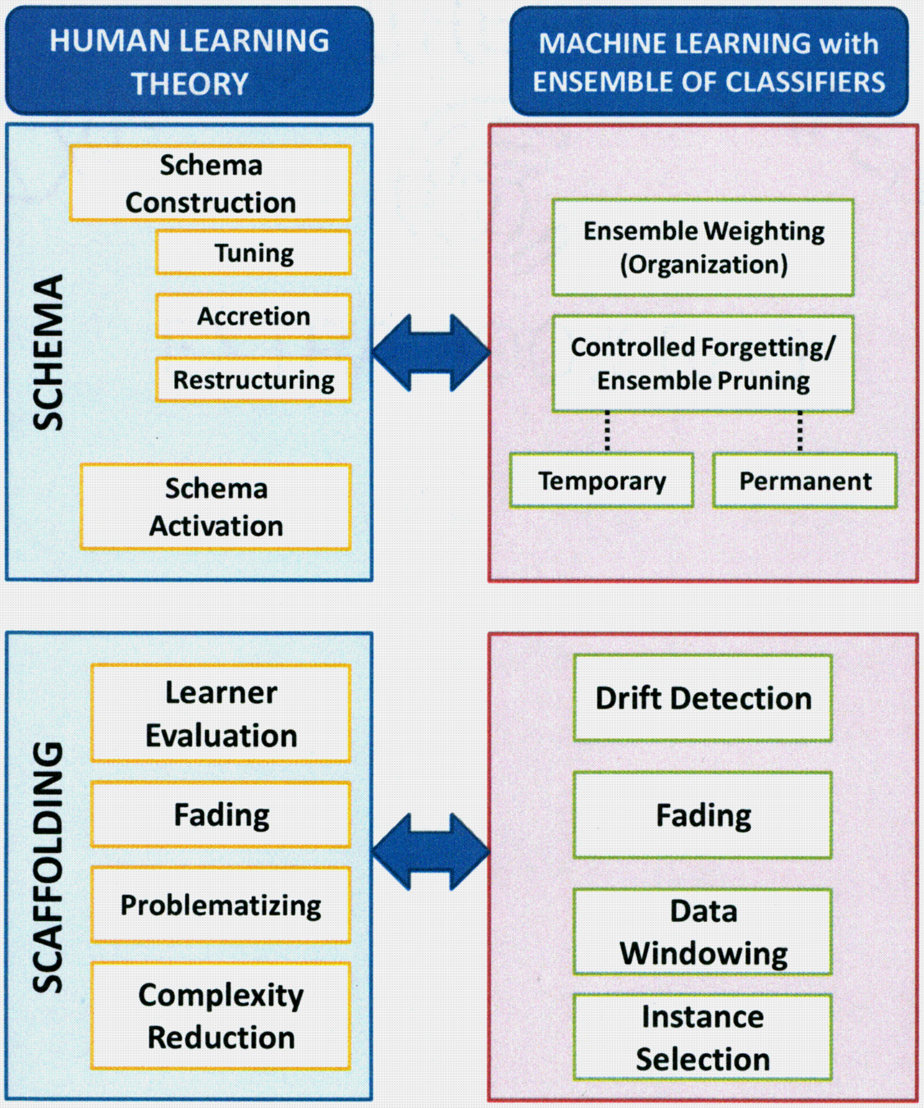


Figure 3.2: Correlation between schema and computational learning

most common "drift trackers" are data parameters and learner performance. Note that evaluation of learner performance requires activation.

3.2 ONLINE & BATCH LEARNING

Streaming data are typically learned either on an instance-by-instance (online) basis or in batches of instances. **Online** learning requires a classifier model that can be updated as new instances are presented one at a time. Examples of such models include the Naïve Bayes classifier, which can update the estimated distribution for each instance, or decision tree classifiers which update by adding nodes and levels to the tree structure. The common cost of instance-by-instance learning is an increase in computational complexity, especially with large quantities of data. The purpose of **batch** learners (such as the MLP or SVM) is to approximate the decision boundary between classes based on a set of multiple data instances. By nature, such classifiers benefit from the availability of increased amounts of data, and can be ineffective when the batch size is so small that they do not properly represent the overall distribution *or* when data from multiple environments is present in the same batch (we discuss *windowing* in a later section). These structures can only be created once for a given dataset and cannot be updated with new data unless retrained with the original instances; such retraining would not satisfy the incremental learning guidelines presented in Section 3.1 (p. 18). Note that while batch classifiers cannot be used for online learning, an online classifier can be adapted to accommodate batches of data. Whereas batch learning is dependent on the size of the incoming batch of data, online learning allows for some variability as to the frequency at which learning occurs; it may not be best to *learn* each and every incoming instance (e.g.

in the presence of noise). By its very nature, online learning is conceptually more readily suitable for incremental learning from a plasticity perspective, but they usually suffer from the stability perspective. Batch learners, typically have the opposite property.

3.3 Ensemble of Classifiers for Learning Concept Drift

While originally developed to improve the performance of a classifier, classifier ensemble systems can also be used – in fact, they are naturally suited – for learning concept drift. The notion of using an ensemble of classifiers was introduced by Desarthy & Sheela in [20] and further discussed by Hansen & Salamon in [21]. In its original development, the use of ensemble systems stems from a realization that, instead of attempting to store a large amount of knowledge within a single classifier model, it is often advantageous to partition the data into compartments, each learned on a different model. Instead of knowledge being stored in a single classifier, multiple classifiers are strategically trained on different portions of the data and then appropriately combined. Although each classifier may only be knowledgeable on a small portion of the data, the effectiveness of *combining* their respective knowledge has been shown to surpass the ability of storing all knowledge in a single classifier.

Using multiple *experts* or *opinions* is logically, theoretically, and empirically supported over a single classifier in many classification scenarios [22;23]. The inherent problem with single classifiers is obtaining the balance between *overfitting* training data (good performance on training data, but poor generalized performance on testing data) and over-generalizing. Single classifiers are especially outmatched in the following cases: 1) In the case of *small datasets*, where the available data may not be representative of the true class distributions, partitioning data into multiple classifiers can increase generalization, whereas a single model runs the severe risk of overfitting. 2) When data provided is extremely difficult (e.g. high dimensionality and/or overlapping), a single classifier may suffer because of the inability to best represent the decision boundary. An ensemble system trained on subsets of the data may effectively be able to break the complex problem into a set of simpler problems in a divide-and-conquer fashion. 3) Another case in which a single classifier is insufficient is when data arrive from a number of different heterogeneous sources (i.e. *data fusion*). Most importantly, as we consider the goals of incremental learning (especially in harsh environments) an ensemble systems approach presents itself as a viable option for constructing a knowledge base.

The chief goal in building an ensemble of classifiers is to reduce overall error; this can be attained by increasing *diversity* amongst the experts such their individual errors can be averaged out through a combination process. Diversity can be achieved by modifying the classifier architecture or parameters, varying the classifier type itself, or training each classifier with unique sub-samples of the data. These considerations apply to a problem which is incremental in nature, but only in a sense that the learner is *incrementally* learning more and more about a single, stationary environment.

In considering the acquisition of streaming incremental data under the guidelines proposed in Section 3.1, it follows that diversity in classifiers will be naturally obtained due to the novelty of the incoming data over time in a non-stationary environment. It follows that using new data as the basis for ensemble diversity will allow the knowledge base to be built such that it encompasses all the information represented in the training data.

3.3.1 Ensemble Organization & Evaluation (Weighting & Voting)

Since the conception of ensemble-based classification systems, much research has been conducted in order to determine the best way to combine the decisions of multiple classifiers for the best possible hypothesis. Comprehensive surveys of classifier combination rules can be found in [24] and [25] for general ensemble methods. Certain combination rules are dependent upon the classifier being used. For instance, the *product*, *sum*, and *median* rules can be used only for classifiers with soft class labels (the classifier provide continuous supports for classes) instead of a hard (absolute) decision. In such cases, we can take either the sum, product, or median of classifier supports for each class. For classifiers which return hard class labels (e.g. SVM), we are limited to a majority vote, where the hypothesized class is the one that receives the most classifier votes.

As we consider the problem of concept drift and the fact that classifiers are created using data from a changing distribution, we realize that a simple majority is not practical. Clearly, some classifiers will be more relevant to the current environment, and based on the schema-based model, it is necessary to measure this relevance and give certain classifiers a more powerful vote. The *weighted majority voting* technique was developed in order to accommodate variable competence of knowledge, as in certain ensemble systems such as Adaboost [26;27]. Instead of simply summing all votes for a particular class, each classifier's vote is multiplied with a weight factor. This factor is based on some evaluation of a classifier's performance on the newest training data, and can be calculated in many ways.

Many weighting methods are heuristic in nature. The first weighting method developed by Littlestone simply assigns a weight of 1 to a classifier when it was created decrements that weight by ½ for each future misclassification [27]. A slight variation of this method is used in [28] and [29], which create variable parameters for the weight initialization (not always 1) or the weight decrement factor β (not always ½). Wang introduces a more standard weighting measure derived from mean square error of classifier i (MSE_i) on the training data x [30]:

$$w_i = \overbrace{\sum_c P(c)\big(1 - P(c)\big)^2}^{MSE_r} - \overbrace{\frac{1}{|\mathfrak{D}_t|} \sum_{(x,c)\in\mathfrak{D}_t} \big(1 - f_c^i(x)\big)^2}^{MSE_i} \tag{3.1}$$

Here, the mean square error of the i^{th} classifier is calculated across all classes for training set $\mathfrak{D}_t$, and subtracted from MSE_r, the mean square error of a random classifier (based on class-dependent probability $P(c)$), to calculate the weight of a classifier. MSE_r is also useful for determining which classifiers perform worse than random guess, given a particular training set.

Fruend & Shapire introduce another dynamic weighting method in Adaboost, where weights are inversely proportional to classifier error ε_t on training data [31]. Weights are not recalculated with each incoming training set; rather, they are updated with the value $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ when the hypotheses $h_t(x_t)$ on data x_t are correct, and $\beta_t = 1$ otherwise (at time *t*). Within the Adaboost algorithm (see Chapter 4 on p.48), this value is bounded within the [32] interval, and is higher for classifiers with greater error.

Weights are determined by taking $\log(1/\beta_t)$, which maps the weights in the $\{0, \infty\}$ range.

Instead of generalizing error across an entire training dataset, Tsymbal relates classifier weight to performance as well as a proximity factor, giving higher weight to a classifier if its training data was in the same region as the testing example [2;33].

Comparisons are made among simple combination rules for incremental learning problems with minor concept drift in [34], as classes are added over the course of time; this represents a change in the priors $P(\omega)$, and is an example of *real* concept drift because the decision boundaries change to accommodate new classes. The comparison shows little difference in performance among sum, product, weighted, and unweighted ensemble combination rules. However, this comparison does not apply to other types of real and virtual drift (e.g. $P(x|\omega)$) which clearly indicate the necessity to dynamically evaluate and weight the ensemble or knowledge base.

Intuition tells us that it is best to weight classifiers, and to do so according to their knowledge on the current environment (not overall knowledge). This is also indicative of the research over the past decade. However, Gao points out an inherent problem with weighting according to current error in [5]. In streaming problems, one cannot always assume that the distributions of most recent training data and the incoming testing data are alike. This assertion leads them to the conclusion that a uniform ensemble should be used. However, this does not make practical sense with an ever-growing ensemble; therefore, it leads to the need for better weighting or organization techniques.

Ensemble weighting is not the only way to categorize knowledge. Next, we discuss a complementary method, which instead of assigning voting weights, completely removes classifiers from the ensemble. This approach is known as pruning.

3.3.2 CONTROLLED FORGETTING (ENSEMBLE PRUNING)

Having discussed that not all knowledge is equally relevant at a given time, and suggested various ways to measure a classifier's relevance, we now discuss the topic of *controlled forgetting*. Pruning is a process whereby knowledge is ignored or removed from the knowledge base, and it is used in a majority of ensemble-based learning algorithms, appearing in two forms: *temporary pruning* and *permanent pruning*. Within the following discussion, we address the following: (1) the motivation/purpose behind pruning, (2) the way that pruning is accomplished (approach) (3) the criterion for choosing what should be pruned, and (4) a comparison between permanent and temporary pruning.

Pruning Motivation. Fundamental to incremental learning is the fact that a knowledge base will grow over time as new classifiers are created and stored. Also, it is well-known that weighted majority voting is an effective technique to give relevant classifiers more voting power. However, this may not always negate the influence of irrelevant classifiers. Depending on the weighting method, it is quite possible that a large number of irrelevant classifiers with low weights will combine to offset the decision made by a small number of relevant classifiers. This phenomenon is the result of classifier *baggage*, and becomes more volatile as the knowledge base increases in size. The purpose of pruning is two-fold: to assure that only the most relevant knowledge is maintained and used for decisions (mitigate *baggage*) and, in the case of permanent forgetting, to conserve memory.

Pruning Approach. Pruning is carried out in one of two ways by instituting a threshold on either (1) classifier performance or (2) the size of the ensemble; in some cases, this threshold can be created to be dependent on the data (e.g. using a confidence interval to identify classifiers below a certain standard deviation from the top performer [35]). Other thresholds are chosen more arbitrarily, based on either the current or cumulative error score of a classifier.

The use of a performance threshold is empirically shown to be effective in reducing the size of the classifier ensemble in non-stationary environments [29]. This approach requires some performance metric to be calculated for all classifiers. This metric is based on current performance in the latest data environment or upon cumulative performance over the life of a classifier in previous environments. When a classifier's performance dips below the threshold, the classifier is pruned. The amount of ensemble-size reduction is not fixed, and is highly dependent on the threshold.

Setting a threshold on the ensemble size itself is a sure way of limiting memory usage. Here, the approach is very simple. When the size of the ensemble reaches a certain threshold, classifiers are removed to accommodate newly trained classifiers. The threshold can be imposed as either a fixed number or a percentage of classifiers to be pruned.

Pruning Criterion. We have already discussed the criterion for removing classifiers with a performance-based threshold. When using an ensemble-size threshold for pruning, three relevance measures can be used for the purpose of determining which knowledge is least competent at the current time: classifier age, current classifier error, overall classifier error [28;35]. Age-based pruning (a.k.a. *replace-the-oldest, oldest first*) makes the simple assumption that the oldest classifier carries the least knowledge about the current environment, and should therefore be ignored or deleted. Error-based pruning can be referred to as either *replace-the-weakest, replace-the-loser*, or *weakest-first* pruning. This type of pruning assumes that current classifier error focuses on tracking the current environment only, and removes/ignores the weakest classifier. Here, a classifier's strength is based on some measure of performance on the current training data, which is calculated using any of the weighting methods described in the previous section. Pruning based on overall error considers a classifier's performance over its entire lifespan. This approach may not always remove the classifier that is weakest at the current time, but it will maintain the best-performing ensemble. A heuristic performance threshold is a common way for evaluating overall performance [29;35], although a moving performance average could also be appropriate.

Permanent vs. Temporary Pruning. The aforementioned pruning criteria (age, current error, and overall error) can be used either for pruning on a temporary or permanent basis. The difference is that permanent pruning irrecoverably removes classifiers that lie outside a given threshold, whereas temporary pruning maintains classifiers so that they may be used in the future. Temporary pruning, in a sense, can be considered as a crisper version of ensemble weighting, where certain classifiers deemed to be irrelevant are simply ignored (given zero weight), and only classifiers deemed as experts are considered for classification [30;35]. Nonetheless, both weighting and temporary pruning can be used simultaneously, where the latter ensures that the effect of baggage is eliminated.

Because the purpose of temporary pruning is to pinpoint the experts in the ensemble, they commonly employ a performance threshold, as opposed to an ensemble-size threshold or age-based approach.

As we consider how pruning applies to the schema model, we see some apparent discrepancies which are inconsistent with human learning. The greatest of these is in the concept of permanent ensemble pruning. Permanent pruning is the severest approach to categorizing knowledge (schemata construction) according to the current environment. When implemented with a small ensemble size (limited knowledge base), permanent pruning is especially problematic because it practically cripples the ability of the knowledge base to learn or recognize environments which are similar to those which were previously seen. This concept of long-term recollection of previous concepts is a key ability of human cognition, however, and is desirable for computational cognition since cyclical and recurring environments are common in natural processes.

Considering this shortfall (known as catastrophic forgetting), long-term forgetting is perhaps is most appropriate and consistent with human cognition. It is reasonable to say that the human brain does not have unlimited memory and that some level of forgetting does occur; however, this is a long-term occurrence. The criterion for human forgetting is widely unknown, but could certainly be related to how often a knowledge base is used or reinforced. Thus, it is intuitive that knowledge that is sufficiently inconsistent with new environments may indeed be forgotten. Maintaining a large ensemble would provide some level of memory savings and baggage reduction while also increasing the possibility for knowledge from recurring environments to be maintained. In other words, increasing the ensemble size increases the stability of the knowledge base. Long-term forgetting has not been explicitly implemented or discussed in current literature, and is therefore investigated in this research.

3.4 Concept Drift Detection

We now discuss the *teaching* aspect of the supervised incremental learning problem, specifically with regard to the *scaffolding* theory. Recall that the main goal of the supervisor is to provide guidance and feedback for the learner by problematizing data (drawing out conflicts between incoming data and the current knowledge base), simplifying complex data, and fading (ceasing the learning process when an environment has been learned) when an environment has been learned. Each of these tasks requires some level of feedback about the incoming data and/or the learner's performance at a given time. This feedback is used to discern how the new information differs from previous data, and determine the learner's capability to grasp current concepts. In computer learning, this is known as **drift detection** and is used for the purpose of tracking changes (steady or gradual) in some representation of the data in order to determine whether or not concept drift is occurring.

In reviewing several approaches of drift detection, we first make the distinction between *active* and *passive* methods. In *passive* drift detection, the learner assumes that the environment may change at any time or is continuously changing, and therefore is continually learning from the environment by constructing and organizing the knowledge base. A passive approach includes aspects of scaffolding in that it may still draw out conflicts between prior knowledge and new data (problematizing). However, passive drift detection ignores the greater part of scaffolding theory that has been discussed, including complexity reduction and fading; both of these require some knowledge about how and when the environment is changing. Although a passive approach does not seek to benefit from these aspects of scaffolding theory, it is still a viable solution. Passive drift detection is a consistent (yet not guaranteed) way to attain plasticity, as all new information is stored in the knowledge base. Ultimately, plasticity is dependent on the ability to categorize (weight) knowledge. The potential risk of storing all new data is that that the new information may be irrelevant to some degree, containing noise or having already been learned. Continuing to learn such new information may be unnecessary at best, and detrimental at worst. *Active* drift detection methods, on the other hand, seek to pinpoint the time and severity of drift, and allow the supervisor to integrate the various scaffolding techniques to improve the learning process, thus fine-tuning the learner's plasticity. The downside of heavy reliance on drift detection is the risk of an imperfect detection mechanism which may yield false reports.

Concept drift can be detected by tracking changes in classifier performance, classifier structure, or data characteristics. First, let us recall the definition of *real* concept drift, that is, a change in distribution behind a particular class ω, that is, $P(\omega|x)$. This distribution is not explicit, as we are only presented with a sampling of the distribution for training. Thus, in many cases we can merely estimate whether true change has occurred, with no distinction between *real* and *virtual* drift.

Consider the following example which shows that an estimate of the true probability distribution is actually insufficient to consistently denote *real* concept drift. Figure 3.3 describes a case in which a 3-class decision boundary is created based on a set of initial training data provided. A subsequent snapshot of the data reveals class-specific data points that are much different than those seen before, so much so that they are inconsistent with the initial or perceived decision boundary that would be generated by a classifier. Although drift is perceived based on the conflict between new and prior knowledge, we eventually realize that these snapshots are consistent with the true decision boundaries (in red), and that we simply have not seen enough data to realize the full coverage of each class in the environment. Such an occurrence can best be described as *perceived* drift (also known as covariant shift in [6]). We reserve the terms *virtual* and *real* to classify the type of drift in retrospect since, in reality, concept drift may or may not have occurred and thus cannot truly be known at the time. Note in the case of virtual drift that although the true class definitions have not undergone change, this new data should still be learned, and not ignored. It is also important to recognize that real concept drift often renders prior classifiers to be irrelevant, but this is not the case for perceived or

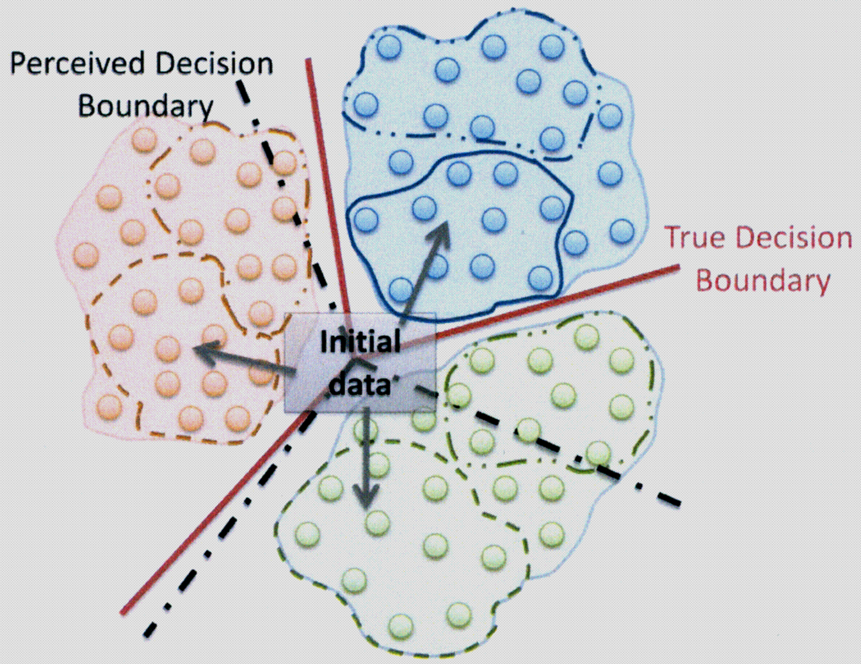


Figure 3.3: Virtual concept drift due to insufficient data

virtual concept drift. Here, it merely indicates that previous classifiers are incomplete in knowledge.

3.4.1 Drift Descriptors

In designing a mechanism for detecting concept drift, there are four main goals:

- Consistently detect *real* drift
- Provide feedback about the type and severity of drift
- Low dependence on outside parameters
- Low computational complexity

The three common types of active concept drift detection (classifier performance, classifier structure, and data characteristics) will be compared with respect to their ability to meet these goals.

Classifier structure. Here, the purpose is to extract useful information from a classifier as it is being updated with incoming data. Examples of structural information could include weight values for MLP node connections, number of branches and levels for a decision tree, distribution parameters for a Naïve Bayes classifier, or the margin location/width of an SVM classifier. Monitoring changes in classifier structure is seldom used, however, because it is difficult to draw out useful (i.e. drift-related) descriptors from a classifier structure for comparison over time. In fact, only the SVM has been used for this purpose, as in [36], by tracking changes in the weight vector *w*, which defines the direction of the hyperplane separating one class from another. Change in the weight vector can give a good approximation of drift because the margin between classes represents the optimal decision boundary by looking for maximum class separation.

Changes in the margin are directly related to the conditional probability $P(x|\omega)$ which may indicate *true* concept drift, as well as the balance of classes $P(\omega)$. However, the complication of tracking changes in the decision boundary increases with multiclass data. Just as the SVM must be modified for multi-class problems, so must a detection mechanism be modified to track the decision boundaries between classes, possibly using a *one-against-one* [37] or *one-against-all* [38] method.

Classifier performance. This is used in the vast majority of active drift detection methods [29;35;39-43], and operates on one of two intuitive assumptions: (1) after training from data $\mathfrak{D}_t$, the error of a classifier on subsequent data $\mathfrak{D}_{t+1}$ will remain relatively constant if the new distribution is the same, or (2) that the error of an ensemble of classifiers learning an environment will increase (and certainly *not* decrease) when the environment is stationary. When either assumption is violated, the inference is that the incoming data are changing in some way, which the learner is incapable of classifying, and therefore the knowledge base must be updated.

In a sense, performance-based change detection is simply a comparison between the perceived decision boundary and the decision boundary reflected by the conditional probability $P(\omega|x)$, since error reflects a learner's ability to correctly approximate the true decision boundary. Recalling that conditional probability $P(\omega|x)$ may or may not correspond to the true decision boundary, the learner may only perceive whether the true distributions are changing. Hence, the terms real drift (the distributions are changing) and virtual drift (the distributions are not changing) apply to this distinction. We see that although significant change in performance is an intuitive and accurate measure of the learner's ability, performance alone offers very little descriptive information regarding the various appearances of virtual drift and will tend to have a high false-positive rate in that respect. Classifier performance is sensitive to the balance of classes (which may vary greatly in consecutive training batches), feature relevance at a particular time, as well as random noise. In terms of the drift detection mechanism, the source of change in performance cannot be discriminated. Nevertheless, we realize that monitoring performance is consistent with the learning model and is an important asset to incremental learning.

Data parameterization is perhaps the most complex of drift detection methods. Here, we seek to represent the data itself (or a representation thereof) as an element of change. Data is most often represented as a distribution. However, since the true distribution is an unknown aspect of the environment, we are left to make an *approximation* of the distribution, make an *assumption* about the distribution, or find some other way to represent the data. Distribution approximation (density estimation) is a computationally intensive process that involves replicating/updating the perceived probability distribution function from the available data. Approximation can be as simple as a histogram-like binning of data, or more complex Parzen windowing. However, the computational cost of building such a model grows exponentially as the number of dimensions increases. A more common approach is to *assume* a distribution.

It is most often assumed that the data follow a Gaussian distribution; this is based on both its frequent occurrence in natural processes and measurements, as well as its simplicity, where the distribution can be represented by statistical moments such as mean, variance, kurtosis, skewness, etc. However, the assumption of normally distributed data is considered by many to be too strong [44;45]; thus there is a desire to represent data in some other way.

The Central Limit Theorem provides an alternative approach to distribution estimation. It states that if a population is sampled with sufficiently large batch sizes, the distribution of those sample averages will tend to be Gaussian. In other words, although a population or environment may not be Gaussian by nature, an increased number of averages taken from that population will tend to be Gaussian as the sample size increases. In [44], Alippi empirically shows that the distribution set forth by the Central Limit Theorem will not remain the same if the environment is non-stationary, and may therefore be used to detect drift regardless of the true distribution of the data. In addition, it must be realized that mean and variance is not sufficient to describe a non-normal distribution, and so further work in [46;47] suggests additional features by which data can be represented. For example, the Mann-Kendall [32] test statistics were developed specifically for representation of non-parametric data, that is, data with an unknown distribution. Trends developing between subsequent samples (x_j and x_{j+1}) are tracked as follows:

$$S_{MK}(t) = \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} sign(x_j - x_k) \tag{3.2}$$

Summing across all available instances *n* at time *t* will yields a statistic which indicates the overall similarity of examples. A large sum represents a steady dissimilarity between instances, and vice versa. The trend developing over time for the sum $S_{MK}(t)$ indicates how the data is changing between batches. Other similar measurements are noted in the following section.

In concluding this discussion on ways to quantify the statistical change in the data, we must recognize the importance of taking a correct probabilistic approach in consideration of the definition of concept drift. This is important because there are two feature-related probabilities that can be measured, or tracked. The simplest to estimate is the overall data distribution $P(x)$, that is, the distribution of all features independent of class. Tracking $P(x)$ allows for drift detection to occur not only on the training data (which requires class labels), but also on new unlabelled testing data, thus transcending the possible discrepancies between testing and training distributions. The downside is that tracking $P(x)$ only allows detection of overall shifts in the data (not changes in the class boundaries), and is therefore a very naïve approach that is highly subject to both false-negative and false-positive mistakes.

Two example environments are shown in Figure 3.4 to exemplify the potential false-positive detections while tracking changes in $P(x)$. In both examples, the overall distribution of the data remains stationary while the classes continually change. Example (A) shows three classes rotatating about a central point. The decision boundary between classes undergoes change, but the overall distribution of the data (independent of class) remains the same. Example (B) illustrates a similar situation, where a uniformly distributed class data are separated by a linear or planar boundary. A shift in this boundary would be a significant change in the concepts, yet it would not be detected using only $P(x)$.

It follows that a class-specific data parameterization is most appropriate for tracking concept drift. Waiting for class labels in streaming data allows for the probability $P(\omega|x)$ to be tracked, providing a more accurate representation of concept

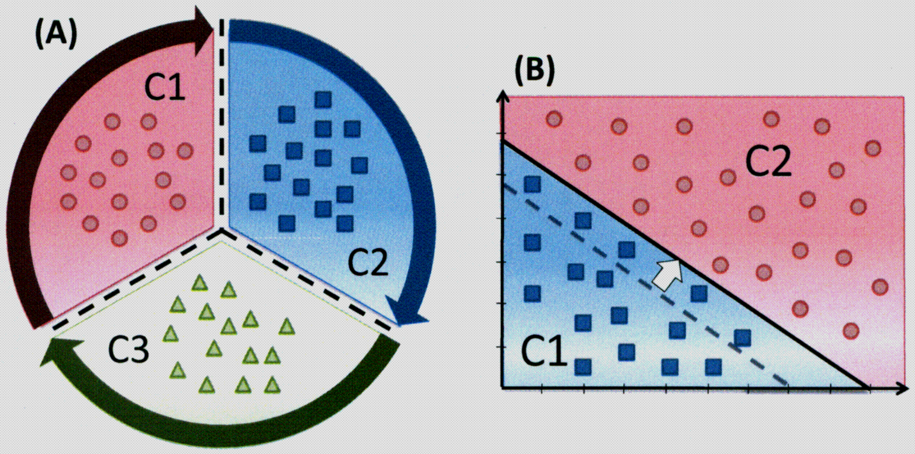


Figure 3.4: False negative drift detection using $P(x)$

drift. This approach is not without its downsides. First, the detection process is delayed since class labels are required before drift can be detected; yet this is not unique from most of the previously described methods. Second, the computational complexity is multiplied by the number of classes present. Finally data parameters are dependent on large data sets; many real-world examples suffer from class-imbalance. Unavailability of a particular class will significantly weaken the ability to correctly detect drift.

3.4.2 MEASURING DRIFT

The discussion on the defining possible descriptors for detecting concept drift leads to two important questions: How do we detect or track changes in the data descriptors (parameters or classifier performance)? And how much change in the descriptors is enough to denote *drift*? Here we discuss various approaches to change detection that have been borrowed from the field of quality control or designed specifically for the purpose of quantifying and detecting significant changes in drift descriptors (classifier structure, classifier performance, or data parameters).

Quality Control (QC) charts were developed prior to computer learning for the purpose of detecting changes in system processes; QC charts have yielded many popular methods which are now used for detecting concept drift. The purpose of QC charts is to indicate when a change in a process has occurred based on some control limit, which is either previously defined or dynamically updated according to the data [48]. In machine learning, dynamic updating is preferred (1) because of the dynamic nature of classification environments, and (2) to achieve the goal of reducing user-defined parameters.

The *Shewhart p-chart* is the original and perhaps the most common detection method [49], especially for tracking changes in performance. Using the central limit theorem, a comparison is created based on the binomial distribution (using mean and variance) to detect whether or not incoming data conforms to the current distribution. Control limits are defined as follows, where $\bar{\mu}$ is the mean of previously acquired sample averages, and n is the number of sample averages:

$$p = \bar{\mu} \pm 3\sqrt{\frac{\bar{\mu}(1-\bar{\mu})}{n}} \tag{3.3}$$

Statistic p represents the confidence interval encompassing three standard deviations on either side of the mean. A subsequent sample average (μ_t) lying outside interval p denotes a process change, or *drift*. The test is self-configured in that it is based on the mean and standard deviation of the data, however, the requirement for a change greater than three (constant) standard deviations diminishes the ability to track small or gradual shifts.

The *Exponentially Weighted Moving Average* (*EWMA*) Control Chart was developed by Roberts in 1959 [50], and is used specifically for concept drift detection in [42]. Similar to the p-chart, the goal is to track changes in a process mean. Instead of directly using the mean μ which only represents the current data, a separate *EWMA* statistic is introduced to weigh the data-under-test over time.

$$EWMA_t = \lambda\mu_t + (1 - \lambda)EWMA_{t-1} \tag{3.4}$$

The term λ defines the consideration given to historical data ($EWMA_{t-1}$), where a lower λ corresponds to more prior data consideration and λ=1 corresponds to only the current data (μ_t) being considered. $EWMA_{t=0}$ is the mean of prior data at the start of the test, and represents the target value. The configuration of control limits is identical to that of the p-chart, *except* that the number of standard deviations (set to 3 in the p-chart) is a user-defined parameter to allow for varied sensitivity.

Cumulative Sum (*CUSUM*) was introduced by Page in [51] and further developed by Lucas in [52-54], and is a well-researched and commonly-used method that is proven to be more accurate than the *p*-chart in many applications [44;55-60]. The greatest advantage of the CUSUM test is sensitivity to small and sustained shifts which cannot be accommodated by a constant 3-sigma control limit. Similar to EWMA, the observed quantity is the cumulative sum of the observations of the mean of a process.

$$S_t = \sum_{i=1}^{t}(\mu_i - \bar{\mu}_0) \tag{3.5}$$

Here, μ_i is an observed value, and $\bar{\mu}_0$ is the expected target value to which observations can be compared; $\bar{\mu}_0$ is often configured using prior data. Thus, the goal is not to detect changes in the mean, but to detect changes in the deviation *from* the expected mean. Barnard [61] (1959) introduced a visual approach to detecting change in CUSUM known as a *V-mask*, which is impractical for our purposes. A more common approach for non-stationary environments is described in Chapter 4.

A key advantage of the CUSUM test is that it requires no assumption about the distribution of the data. Many other distribution-free techniques have also been developed for comparing and quantifying differences between consecutive distributions. These include, Mann-Whitney U [62], a two-sample *t*-test for determining whether observations lie in one population (*null* hypothesis) or another (*alternative* hypothesis) based on a ranking statistic; Kruskal Wallis [63] , a one-way analysis of variance to determine whether or not two independent sample sets come from the same distribution; and Friedman [64] tests, which is a performance-based comparison to observe statistical changes in performance across a large number of bootstrapped sample sets. However, a study on many of these methods appears to indicate that no single change detection test is sufficient to accurately and consistently detect concept change [65].

3.4.3 PROBLEMATIZING (WINDOWING AND INSTANCE SELECTION)

In addition to observing changes in the concepts, the supervisor is responsible for controlling information flow by breaking data into learnable chunks or discerning among data with varying importance. The primary goal is to provide the learner with information which is most relevant to the current environment; this is often accomplished in tandem with a drift detection mechanism.

Data **windowing** is among the earliest methods for providing the learner with data that is deemed relevant to the current environment [41;66-68]. Incremental data can be visualized as a stream of instances which are broken into consecutive batches or windows through time. Windowing methods make the valid assumption that most current window or chunk of data is representative of the current environment (and *only* of the current environment) and use drift detection to determine an appropriate window size; this is known as *dynamic window adjustment* [67]. As new data are made available, old instances are deemed no longer relevant and are removed from the window. The weakness of many current implementations of windowing is that classifiers are trained using only the data in the current window, and often no prior knowledge is retained, thus crippling the learner's ability to recognize recurring environments. Additionally, dynamic window adjustment is only as good as the detection method used, and must be carefully chosen to minimize false drift reports.

The age of data is not the only way to determine what the learner should add to its knowledge. Windowing can be considered as a simple version of **instance selection,** where particular instances from the current data are selected. The purpose of instance selection is to ignore irrelevant (noisy or redundant) data. *Boosting* is an instance selection scheme developed and used originally for stationary incremental learning problems using an ensemble of classifiers [31;69;70]. By taking previous classifier performance into account, subsequent classifiers are more likely to be trained on data which were previously misclassified. Similarly, **instance weighting** gives preference to previously misclassified points. Both instance weighting and instance selection are a means to emphasize what has not been previously learned, and both methods have been adapted to the problem of handling concept drift [7;71;72].

CHAPTER 4: LITERATURE REVIEW

In this section, we highlight some particular approaches and algorithms which are foundational to research in non-stationary environments, specifically with respect to our work. These will be organized and evaluated with regard to the following criteria:

- Knowledge acquisition (batch vs. online)
- Drift detection (active vs. passive)
- Knowledge construction & organization
- Ability to handle recurring contexts
- Overall adherence to the proposed human-based computational learning model.

4.1 Ensemble-Based Methods

4.1.1 FLORA.

The *FLORA* family of algorithms (Widmer & Kubat, 2004) [67] is among the first to address the problem of non-stationary environments. The original FLORA is an online algorithm which stores and organizes descriptors (classifiers) based a window of incoming data instances. Descriptors are placed in one of three categories: (1) positive descriptors are most accurate according to the current data window, and are used for classification of unknown data. (2) Negative descriptors are least accurate in the current window, and are used to prevent over-generalization. (3) Potential descriptors include descriptors which were formerly positive or negative, but now have little relevance to the current data window. Each descriptor maintains a counter based on the number of data examples that they are able to correctly represent, and descriptors are pruned based on their relevance in the current data window. Several variations developed to accommodate particular challenges in non-stationary environments including drift speed, recurring contexts, and noise. These individual problems are handled using a window adjustment technique based on active drift detection (FLORA2), saving contexts descriptions for later use (FLORA3), and introducing a context interval around classification accuracy to decrease drift-detection sensitivity. The FLORA framework is consistent with the goals set forth for the computational learning model (Section 3.1, p.18) insofar as it is able to categorize knowledge pertaining to the current environment. However, FLORA is clearly unable to store old knowledge for future use because any descriptors that are irrelevant to the current data window will be removed permanently.

4.1.2 Streaming Ensemble Algorithm (SEA)

Streaming Ensemble Algorithm (*SEA*) [73] is among the first of batch-based ensemble of classifiers approaches to handling streaming data. The pseudocode in Figure 4.1 provides a formal description of the quality-update rule. In Step 1, a new classifier is trained on each new batch of data. As classifiers are trained on consecutive batches, a constant ensemble size is maintained (*k* experts), and experts are permanently pruned using a quality score (Step 7). Quality is defined as a classifier's ability to correctly classify data relative to the error of the ensemble. This ranking method, albeit heuristic, is a novel and accurate way to evaluate and update each classifier's competence in the current training environment for the purpose of pruning.

```
Algorithm SEA (Streaming Ensemble Algorithm)
Input:
  • Base classifier
  • Training dataset $\mathfrak{D}_t$ of size $N$, $t = 1, \ldots, T$
  • Maximum ensemble size $k$ (default $k = 25$)
Do for $t = 1,2, \ldots, T$
 1. Create new base classifier
 If $t > k$
   2. Evaluate ensemble on $\mathfrak{D}_t$ for composite hypothesis $H(n)$ using simple
      majority vote, $n = 1, \ldots, N$
   3. Train new classifier on data $\mathfrak{D}_t$ and obtain individual classifier hypothesis:
      $h_j(n)$, $j = 1, \ldots, k$ and $n = 1, \ldots, N$
   For $n = 1,2, \ldots, N$, Compute:
     $PC^*$: classification percentage of true class $c^*(n)$ for instance $n$
     $PC_c$: classification percentage of all classes $c$ for instance $n$
     $P1$: top classification percentage among classifiers for instance $n$
     $P2$: second highest classification percentage among classifiers for instance $n$
     For $j = 1,2, \ldots, k$
       If $h_j(n) = c^*(n)$
         If $H(n) = c^*(n)$
           4. Reward: $Q_j = Q_j + 1 - |P1 - P2|$
         Else
           5. Reward: $Q_j = Q_j + 1 - |P1 - PC^*|$
         Endif
       Else
         6. Penalty: $Q_j = Q_j - \left(1 - \left|PC^* - PC_{h_j(n)}\right|\right)$
       Endif
     Endfor
  If $Q_{k-1} > Q_j$ for $j = 1, \ldots, k-2$
    7. Prune classifier $j$
  Endif
 Endif
8. Compute composite hypothesis using simple majority
End
```

Figure 4.1: Streaming Ensemble Algorithm (SEA) pseudocode

Correct classifiers are given a reward proportional to the margin between the supports for the two highest-voted classes, $P1$ and $P2$; a high reward is indicative of contention in the ensemble decision (from Step 2) for a given example, and less reward is given when there is more unity among ensemble members (as in Step 4 or 5). Thus, the experts that perform well when the ensemble does not agree are especially rewarded. If a classifier is wrong, it is penalized proportional to margin between the ensemble's support for the true class PC^* and the support that the faulty classifier gives to its class of choice, $PC_{h_j(n)}$ (Step 6). SEA is shown to react to concept change and learn data from new environments in a concept-changing dataset. The blatant weakness of this and other permanent pruning methods is the inability to store long-term information about recurring environments. Also, we note that although weighted and unweighted ensemble voting techniques are empirically shown to be statistically similar in performance in [73], it seems that a weighted ensemble would be favorable for a larger knowledge base (classifier ensemble) for quick reaction to concept change.

4.1.3 Classifier Ensemble Approach for Mining Concept-Drifting Data

Wang [30] uses an ensemble classification approach with the expressed goal of accommodating large-scale drifting data streams. Wang contributes theoretical and empirical support for the use of a *weighted* ensemble of classifiers as opposed to single classifier approach. Weights are determined using the predictive accuracy of each classifier on the training data and is calculated using the mean square error method described in section 3.3.1, Equation (3.1). These weights are also the criterion for permanent ensemble pruning (similar to SEA). To some degree, it seems reasonable to incur some amount of knowledge forgetting for the sake of limiting computer memory; however, this is surprising considering that recurring environments cannot be realized over long periods of time.

4.1.4 Conceptual Clustering & Prediction

The *Conceptual Clustering & Prediction (CCP)* framework [74;75] provides an alternative to ensemble weighting that uses data clustering to associate classifiers to a particular environment. Clustering can be considered a type of drift detection, where data (or a representation thereof) is grouped into a predetermined number of batches (contained in G). Figure 4.2 contains the pseudocode for data clustering and building the ensemble of classifiers H. Incoming labeled training samples $x_L(i)$ in batches of size b are first mapped using a mapping function $M(\cdot)$. The mapping and clustering phase

```
Algorithm CCP (Conceptual Clustering & Prediction)
Input:
    • Data stream $\mathfrak{D}$ of instances $\{x_i, \omega_i\},\ i = 1,\dots,N$
    • Batch size $b$
    • Mapping function $M(\cdot)$
    • Incremental Clustering Algorithm $R(\cdot)$
For $n = 1,\dots,N$
  Initialize ensemble $H = 0$, cluster $G = 0, s = 1,\ j = 1$
  1. Add new classifier: $H = H \cup \{h_s\}$
  For $i = 1,\dots,N$
    2. Get prediction on new unlabelled data: $p_i = h_s(x_u(i))$
    If $i$ mod $b = 0$, then      // batch full //
      3. Get batch of labeled data $B_L(j) = \{x_L(i-b),\dots,x_L(i)\}$
      4. Map batch $B$ using function $M$ $Z_j = M(B_L(j))$
      5. Set cluster $s = R.cluster(Z_j)$
      If $\exists g_s \in G$ then      //match with existing cluster//
        6. Update: classifier $h_s$ with batch $B_L(j)$
                   cluster $Z_j = R.update(Z_j)$
      else
        7. Set new cluster $g_s = Z_j$,
        8. Update $G = G \cup \{g_s\},\ H = H \cup \{h_s\}$
        9. Update classifier $h_s$ with batch $B_L(j)$
      endIf
      $j = j + 1$
endAll
```

Figure 4.2: Conceptual Clustering & Prediction (CCP) pseudocode

determines if data belongs to an existing cluster or a new cluster and assumes each cluster is normally distributed, described mean and variance. Data are categorized into either an existing cluster or a new cluster based on a distance measurement (using existing cluster means and variances). Each instance is used to update (Step 6) or train a new classifier (Step 7) for the ensemble, where each classifier h_s is representative of the knowledge from a particular cluster (environment) g_s. New, unlabelled data instances are classified online in Step 1; the classifier associated with the cluster to which the unlabeled data belongs is used for classification. CCP offers a practical alternative to ensemble weighting; however, clustering algorithms are often limited in that they require the number of clusters (environments) to be determined before testing.

4.1.5 DYNAMIC WEIGHTED MAJORITY (DWM)

Dynamic Weighted Majority (*DWM*) is an online approach to building a weighted ensemble of classifiers (a.k.a. experts). Weights are determined heuristically using an update factor (similar to the original Weighted Majority by Littlestone in [26]) which decrements a classifier's weight with each misclassification. Weighted majority vote is used to evaluate the testing data at any time, and retraining occurs for each new instance as soon as the class label is available. Figure 4.3 shows the pseudocode for the ensemble construction and organization. For each new instance x_n, each expert e is evaluated (Step 1), and weights w_j for a classifier j which misclassify the current data are decremented using the update factor β (Step 2), and then all weights are normalized (Step 3). The knowledge base is reconstructed at a user-defined period p, at which time the algorithm removes classifiers with weights below the pruning threshold θ (Step 4), and

```
Algorithm DWM (Dynamic Weighted Majority)
Input:
    • Dataset $\mathfrak{D} = \{x_n, \omega_n\},\ n = 1, \dots, N$
    • Update period $p$ (default $p = 1$)
    • Weighting factor $\beta$ (default $\beta = 0.5$)
    • Pruning weight threshold $\theta$ (default $\theta = 0.01$)
Initialize ensemble size $k$=1
For $n = 1, \dots, N$
    If $n = 1$, Train new ensemble expert $e_k$, Initialize weight $w_k = 1$
    1. Evaluate all experts on example $x_n$ $h_j : x_n \to \omega_n$
    2. Update weights:
       $w_j = \begin{cases} \beta w_j & if\ hypothesis\ h_j(x_n) = \omega_n \\ w_j & otherwise \end{cases} \quad for\ j = 1, \dots, k$
    Endif
    If $mod(n, p) = 0$
        3. Normalize weights $w_j = w_j / \sum_j w_j \quad for\ j = 1, \dots, k$
        4. If $w_j < \theta$, Remove expert $e_j$, decrement $k = k - 1 \quad for\ \ j = 1, \dots, k$
        5. Obtain weighted ensemble hypothesis:
           $H(x_n) = \arg\max_y \sum_{j:h_j(x_n)=y} w_j$
        If $H(x_n) \neq \omega_n$
            Increment $k = k + 1$,
            Train new ensemble expert $e_k$
            Initialize weight $w_k = 1$
        Endif
    6. Re-train ensemble on sample $\{x_n, \omega_n\}$
EndAll
```

Figure 4.3: Pseudocode for Dynamic Weighted Majority (DWM) Ensemble Construction

adds a new classifier if the ensemble has misclassified the current data instance (from Step 5).

DWM is shown to perform well in the presence of concept drift on both synthetic and real-world datasets involving harsh concept drift and concept change. DWM is able to store relevant information a relatively low amount of online classifiers and achieve high plasticity, since the pruning criterion effectively limits the size of the ensemble.

Because the ensemble size is not capped and the pruning weight measurement is cumulative, DWM *can* be capable of maintaining knowledge for future environments. However, this is improbable because of the threshold parameter. Decreasing the weight threshold (allowing more classifiers) will increase the number of classifiers to be maintained, but it could also potentially increase baggage from irrelevant classifiers. Also, the weights cannot be increased at a later time, and so a long period of irrelevance will render that knowledge virtually useless at a later time.

4.1.6 Knowledge-Based Sampling (KBS)-Stream Algorithm

The *Knowledge-Based Sampling (KBS)-Stream* algorithm is a batch based ensemble of classifiers approach to actively adjusting to concept drift (algorithm pseudocode in Figure 4.4). KBS-Stream is active in that uses competing classifier ensembles (current ensemble M and prior ensemble M^*) to determine whether the environment has changed (Step 1-4). The alternative ensemble is simply a clone of the current ensemble minus the most recent classifier (as seen in Step 9). Thus, the comparison of the two ensembles reveals whether or not the addition of a new classifier increases performance in the current environment.

In addition to dynamically adjusting the structure of the ensemble, KBS-Stream employs both a windowing and instance weighting method for evaluating classifiers. Windowing is closely associated with the ensemble comparison for drift detection, operating under the assumption that an increased performance of new ensemble M over old ensemble M^* indicates that the environment has changed, since the new classifier has added relevant knowledge. In such a case, the alternative ensemble is discarded, as well as the training data associated with it. Instance weighting is accomplished by constant evaluation the classifier ensemble on the most recent training data to identifies instances which are unknown; the ability of an individual classifier to correctly classify instances in a batch of data represented by the LIFT of an instance x in training set $\mathfrak{D}$:

$$\mathrm{LIFT}_D^{h(\mathfrak{D})}(\omega^* - \omega') = \frac{P_{x,\omega}[h(x) = \omega^*, \omega = \omega']}{P_{x,\omega}[h(x) = \omega^*] \cdot P_{x,\omega}[\omega = \omega']} \tag{4.1}$$

The LIFT quotient measures the correlation between the predicted class label ω^* from classifier hypothesis $h(x)$ and true class label ω', where a positive correlation is represented by LIFT greater than 1. The distribution D, yielded by the LIFT of each instance in $\mathfrak{D}$, is iteratively updated for each classifier (Step 5). The purpose of this distribution is two-fold: first in training a new classifier model (Step 6), where previously misclassified instances are boosted (given a higher probability to be sampled for training); second, the LIFT distribution is used to evaluate classifiers on unlabeled data, acting as a weight for each classifier. To this end, the vote of each classifier is weighted by the product of LIFT values for that classifier to obtain composite value β for instance x (Equation 4-2); this value is used to compute the final probability estimate for an unknown point x belonging to class ω (Equation 4-3).

$$\beta(x) = \frac{P(\omega = +1)}{P(\omega = -1)} \prod_{t=1}^{T} \frac{\mathrm{LIFT}_{D_i}^{h}(h_i(x) \to \omega = +1)}{\mathrm{LIFT}_{D_i}^{h}(h_i(x) \to \omega = -1)} \tag{4.2}$$

$$H = \frac{\beta(x)}{1 - \beta(x)} \tag{4.3}$$

The KBS-Stream algorithm is shown to be effective in adapting to slow drift situations, and operates similar to other boosting algorithms such as Adaboost (see description Section 4.3.1). One particular downfall of the KBS-Stream algorithm is the loss of prior knowledge when drift is detected. Along with discarding old training data,

```
Algorithm Knowledge-Based Sampling Stream (KBS-Stream)
Input:
    • Data batches $\mathfrak{D}_t$ of instances $\{x_t, \omega_t\}$ of size $m$, $t = 1,\dots,T$
Initialize current ensemble $M = [\,]$, previous ensemble $M^* = [\,]$
For $t = 1,\dots,T$
    1. Evaluate new data using ensemble: $H_M =$
    If ensemble $M^*$ exists
      2. Evaluate data $\mathfrak{D}_t$ with $M$ and $M^*$, get error $\varepsilon_M$, $\varepsilon_{M^*}$
      if $\varepsilon_M < \varepsilon_{M^*}$
        3. Discard $M^*$, $\mathfrak{D}^* = \mathfrak{D}_{t-1}$
      endIf
    4. Initialize distribution $D_1$ uniformly over $\mathfrak{D}_t$
    5. For $i = 1,\dots,M$
        - Obtain hypothesis $h_i(\mathfrak{D}_t)$ for classifier $i$
        - Recompute $\mathrm{LIFT}_{D_i}^{h_i(\mathfrak{D}_t)}(\omega^* - \omega')$
        - Update LIFTs of $h_i(\mathfrak{D}_t)$ in $M$:
          $D_{i+1}(x_j, \omega_j) = D_i(x_j, \omega_j) \cdot \left(\mathrm{LIFT}_{D_i}^{h_i(x_j)}(h(x_j) - \omega_j)\right)^{-1}, j = 1,\dots,m$
    endFor
    6. Train classifier model $h_{|M|+1}$ on $\mathfrak{D}_t$ with $D_{|M|+1}$
    7. Compute $\mathrm{LIFT}_{D_{|M|+1}}^{h_{|M|+1}(\mathfrak{D}_t)}(\omega^* - \omega')$
    8. Add model $h_{|M|+1}$ and LIFTs to ensemble M
    9. If $t = 1$, $\mathfrak{D}^* = \mathfrak{D}_t$
       Else
         Extend previous batch $\mathfrak{D}^* = \mathfrak{D}^* \cup \mathfrak{D}_t$
         $M^* = clone(M)$
         Discard last base model of $M^*$
         Repeat Steps (4-8) using $\mathfrak{D}^*$ and $M^*$
      endIf
```

Figure 4.4: Knowledge-Based Sampling (KBS) Stream Algorithm Pseudocode

old knowledge is also pruned, rendering the ensemble unable to recognize recurring environments.

4.1.7 Adaptive Classifiers-Ensemble (ACE)

The *Adaptive Classifiers-Ensemble* (*ACE*),introduced by Nishida in [35;76], is a hybrid method for detecting and responding to non-stationary environments. The term "hybrid" alludes to the many approaches that are combined to handle concept drift. Pseudocode for the ACE algorithm is provided in Figure 4.5. ACE uses an online *and* batch-based learning strategy to maintain both plasticity and stability. An online learner (classifier j_0) is employed for learning the most recent data, and the output of a drift detector is used to train a batch classifier ($j_1, \ldots, J$) on the window of data (B_l) that represents the most recent environment. The recent environment is represented by the data instances spanning from N_0 to t(Step 6), and a batch classifier is created with these instances only when drift is detected *or* when the current data window exceeds some chunk size threshold S_c. Once the classifier is trained, the online classifier hypothesis H_0, data window B_l for batch learning, and time pointer N_0 are re-initialized. Weighted majority vote is used to combine classifier predictions (used in Step 1, computed in Step 2). Log-based weights, v_j, (as in Adaboost, see Section 3.3.1) are calculated based on classifier performance ($A_{j,t}$ for classifier j at time t) on the most recent training datapoints (size S_a). Temporary pruning is employed (Step 2) in addition to weighting in order to ignore irrelevant classifiers at a given time by giving a non-zero weight only to classifiers with performance that lies within a given confidence interval (upper bound $A_{j,t}^u$ and lower bound $A_{j,t}^l$ based on confidence level $100(1-\alpha)\%$). Permanent pruning decreases memory usage by removing classifiers which have performed poorly over an extended period of time (error-based pruning). The criterion for permanent pruning is a separate

```
**Algorithm ACE (Adaptive Classifiers Ensemble)**
**Input:**
  • Dataset $\mathfrak{D} = \{x_t, \omega_t\},\ t = 1,\dots,N$
  • Short term memory size $S_a$, chunk size $S_c (> S_a)$
  • Maximum ensemble size $\theta$
  • Drift detection confidence level $100(1-\alpha)\%$,
  • Ensemble weighting confidence level $100(1-\beta)\%$
**Initialize** $J = 0, B_l = \{\}, N_0 = 0$
**For** each $(x_t, \omega_t)$
1. **Output** final hypothesis $H_f(x_t) = \arg\max_y \sum_{j=0}^{J} v_j [\![H_j(x_t) = \omega]\!]$
2. **For** each $j = 0, \dots, J$
     $CR_{j,t} = [\![H_j(x_t) = \omega_t]\!]$
     $A_{j,t} = \begin{cases} \sum_{s=\max[t-S_a+1, N_0+1]}^{t} CR_{j,s} / \min[t - N_0 + 1, S_a] & if\ j = 0 \\ \sum_{s=t-S_a+1}^{t} CR_{j,s}/S_a, & otherwise \end{cases}$
     **Compute** $100(1-\alpha)\%$ confidence intervals for proportions $[A^l_{j,t}, A^u_{j,t}]$
     **Compute** classifier voting weight:
     $v_j = \begin{cases} \log\left(\frac{A_{j,t}}{1-A_{j,t}}\right), if\ A_{j,t} > \max_k A^{l,\beta}_{k,t} \\ 0, \quad otherwise \end{cases}$
     **Compute** classifier pruning weight:
     $j_{max} = \arg\max_{j=0,\dots,J} A_{j,t}$
     $w_j = \begin{cases} w_j + 1 \text{ if } j = j_{max}\ and\ w_j \ge 0 \\ 0 \quad \text{if } j = j_{max}\ and\ w_j < 0 \\ w - 1,\ otherwise \end{cases},\ j = 1, \dots, J$
  **endFor**
3. Call **Online Learner** $(x_t\ \omega_t)$; Update hypothesis $H_0: X \to \Omega$
   **Add** $(x_t, \omega_t)$ to $B_l$,  **Set** $AcqFlag = false$, **Set** $j' = \arg\max_{j:1,\dots,J} \sum_{s=t-S_a+1}^{t} A_{j,s}$
4. **If** $\{A_{j',t} < A^l_{j',t-S_a}$ or $A_{j',t} > A^u_{j',t-S_a}\}$ and $t - N_0 \ge S_a$ and $t - N_{j'} \ge 2S_a$
     **Set** $AcqFlag = true$
   **endIf**
5. **if** $t - N_0 \ge S_c$ , **Set** $AcqFlag = true$ **endIf**
   **if** $AcqFlag = true$,
     6. Call **BatchLearn** $(B_l)$; **Get** hypothesis $H_{J+1}: X \to \Omega$ with error:
       $E_{J+1} = \frac{\sum_{(x_1,\omega_1)\in B_l} [\![H_{J+1}(x_i) \ne \omega_i]\!]}{|B_l|}$
       **If** $E_{J+1} \ge 1/|\Omega|$, Discard $H_{J+1}$
       **Else for** each $m = (t - S_a + 1), \dots, t$
         7. **Set** $CR_{J+1,m} = CR_{0,m}$
         8. **Set** $A_{J+1,m} = A_{0,m}, A^l_{J+1,m} = A^l_{0,m}, A^u_{J+1,m} = A^u_{0,m}$
       **endFor**
       9. **Set** $N_{J+1} = t,\ J = J + 1$
       **endIf**
     10. **Initialize** $H_0$; Set $B_l = \{\}, N_0 = t$
   **endIf**
11. **If** $ensembleSize > \theta$, **Remove** classifier $k$ where $k = \arg\min_{j=1,\dots,J} w_j$    **endIf**
**endFor**
```

Figure 4.5: Pseudocode for Adaptive Classifiers Ensemble (ACE) algorithm

weight (w_j) which increments or decrements based on classifier performance $A_{j,t}$ (updated in Step 2). Pruning occurs when the size of the ensemble exceeds a threshold θ (Step 11). ACE uses active drift detection by tracking performance of the batch classifier decision. In Step 4, drift is triggered when ensemble performance exceeds a confidence interval configured on an initial sequence (size S_a) of classification performances. The output of the drift detector indicates when a new batch classifier should be trained on the most recent window of data that represents the current environment (Step 6).

ACE appears to be a very robust algorithm in that it utilizes nearly every available approach to handling concept drift that has been discussed in the previous chapter, although it is not tested extensively on datasets involving various types of drift. Possible downsides of this approach are a high number of user-defined parameters that may need to be fine-tuned for any given experiment. ACE boasts the ability to handle recurring environments using a batch-learning approach; however, permanent pruning removes all guarantee that knowledge (classifiers) will be maintained until an environment re-occurs.

4.2 Drift Detection Methods

4.2.1 Early Drift Detection Method

The *Early Drift Detection Method* (*EDDM*) [39] is a performance-based drift detection mechanism which uses a heuristic for sensing significant change in ensemble performance. Instead of directly using the binomial distribution on percentage of error (as in [43]), which is reasoned to be effective only with concept *change*, EDDM configures the drift detection test using the average distance between error (p_n') and standard deviation (s_n') for each sample n within an initial period of 30 ensemble errors.

The maximum average distance between errors in the initial (configuration) period is represented by $\max(p_n' + 2s_n')$, and marks the point where the ensemble performed best. Drift is detected using two threshold levels: *warning* and *drift*.

$$Warning\ level: \frac{p_n' + 2s_n'}{p_{max}' + 2s_{max}'} > \alpha \tag{4.4}$$

$$Drift\ level: \frac{p_n' + 2s_n'}{p_{maz}' + 2s_{max}'} > \beta \tag{4.5}$$

Drift is perceived when the warning level is reached, but there is no reaction until the drift level is passed. It is here that the test is reconfigured by recalculating the parameters within a new period of 30 ensemble errors, and a new classifier is trained on the data that was seen after the warning level; data following a warning is assumed to represent the new environment. EDDM appears to be an effective way to track gradual drift, and is partially self-configuring, requiring only two bounded parameters (warning and drift thresholds) to be manually configured.

4.2.2 Computational Intelligence-CUSUM (CI-CUSUM)

Alippi describes a parameter-based drift detection mechanism in [44;46;47;60] which combines distribution-free CUSUM change detection with a log-likelihood measurement based primarily on the central limit theorem. *Computational Intelligence CUSUM (CI-CUSUM)* first builds a feature vector from consecutive batches of incoming data, as seen in the pseudocode in Figure 4.6. In Step 1, each concurrent batch of data $\mathfrak{D}_t$ is represented by a feature vector $\varphi(t)$ containing the mean, variance, skewness, kurtosis, Mann-Kendall features, and CUSUM features as described in [47].

A large feature base decreases any dependence on a particular distribution; however, it also increases computational complexity, especially as each feature set is calculated for each dimension. For this reason, principal component analysis (PCA) is used (Step 2) to find the feature combinations that carry the most information. The

```
Inputs:
 • Configuration size $CS$
 • Training size $M$
 • Data $\mathfrak{D}_t,\ t = 1, \dots, T$
For $t = 1, \dots, T$
  Initialize $t_{drift} = 0$
    1. Compute feature vector $\boldsymbol{\varphi}(t)$ from dataset $\mathfrak{D}_t$
    2. Apply Principal Component Analysis to obtain $\boldsymbol{\varphi}_{PCA}(t)$
    Elseif $t - t_{drift} = CS$
      3. Compute mean $\hat{M}_0$ and covariance matrix $\hat{C}_0$ for null hypothesis
         $\hat{M}_0 = \frac{1}{CS}\sum_{t=t_{drift}}^{t_{drift}+CS} \boldsymbol{\varphi}(t),\quad \hat{C}_0 = \frac{1}{CS}[(\boldsymbol{\varphi}(t) - \hat{M}_0)(\boldsymbol{\varphi}(t) - \hat{M}_0)']$
         $H_0 : \theta^0 = \{\hat{M}_0, \hat{C}_0\}$
      4. Compute confidence interval $M_{1,max}, M_{1,min}, C_{1,max}, C_{1,min}$
         Select alternative hypothesis $\theta^1 = \{M_1, C_1\}$
      5. Configuration Parameters
        - Log-likelihood
          $R(\tau) = \sum_{i=t_{drift}}^{\tau} \ln \frac{P_{\theta^1}(\boldsymbol{\varphi}(i))}{P_{\theta^0}(\boldsymbol{\varphi}(i))},\quad for\ \tau = t_{drift} \dots t$
        - Minimum: $m(\tau) = \min_\tau R(\tau),\quad for\ \tau = t_{drift} \dots t$
        - CUSUM parameter: $g(\tau) = R(\tau) - m(\tau),\quad for\ \tau = t_{drift} \dots t$
        - Threshold: $h(\tau) = \max_\tau g(\tau),\quad for\ \tau = t_{drift} \dots t$
  Else $(t - t_{drift} > CS)$
    6. $R(t) = \sum_{i=t_{drift}}^{t} \ln \frac{P_{\theta^1}(\boldsymbol{\varphi}(t))}{P_{\theta^0}(\boldsymbol{\varphi}(t))}$
    7. Compute $m(t),\ g(t)$
    If $g(t) > h(t)$
      $drift(t) = 1$
      $t_{drift} = t$
    Else  $drift(t) = 0$    Endif
  Endif
End
```

Figure 4.6: Data parameter log-likelihood CUSUM test for drift detection

detection method uses a configuration sequence (size *CS*), beginning at time t_{drift}, to configure the parameters of the test, making it robust in the presence of various types of drift.

The metric for change is the result of a CUSUM log-likelihood function $R(t)$, which compares a feature's probability of belonging to either the null (θ^0) or alternative (θ^1) distributions computed in Step 4. The null distribution is defined by assuming the central limit theorem over an *initial* sequence of feature vectors (configuration sequence) of size *CS*, and the alternate distributions are defined as the upper and lower bounds of the confidence interval around θ^0.

Drift is detected using a threshold defined completely by the data; this threshold is calculated using the configuration parameters in Step 5. Following the log-likelihood calculation, the minimum of the sequence $m(t) = \min_{\tau=1:t} R(\tau)$ is computed as the base value for the CUSUM test. The difference between the current likelihood and the minimum, that is $g(t) = R(t) - m(t)$, can be considered as the true metric for the original CUSUM test, where $R(t)$ is the current sample measurement, and $m(t)$ is the desired value from which measurements will deviate. The threshold is the maximum deviation from the minimum that is observed within the configuration sequence: $h(t) = \max_{t=1:k} g(t)$. After configuration, these parameters are calculated for each incoming batch (Step 6-7), and compared to the threshold h; when this threshold is surpassed, drift is detected, and the test is reconfigured beginning at the updated time t_{drift}.

There are two main weaknesses with this approach. The primary weakness is in the *application* of the method (not the method itself), as it is only used to detect change in probability $P(x)$ and therefore reveals no information about class drift. The second weakness is that the knowledge construction aspect is quite primitive using a KNN classifier that learns incrementally from the point at which drift is detected.

4.3 Methods Related to Our Work

As we begin to describe the specific contributions of our work, let us first introduce two important algorithms which lay the foundation. Algorithm Adaboost, although not designed for incremental learning, introduces the groundwork for the classifier-ensemble model and weighting scheme. Algorithm Learn^{++} extends Adaboost to accommodate incremental learning, yet not specifically to learn from non-stationary environments.

4.3.1 Adaboost

Adaboost (*Adaptive Boosting*) [31] is a learning algorithm that builds a strong classifier (generalized knowledge base) by creating an ensemble of *weak* classifiers, that is, classifiers with performance slightly better than random guess. Fruend & Shapire give theoretical and empirical evidence that such an ensemble will exceed in performance over a single classifier trained on the same data. Weak classifiers are created using an instance selection process known as *boosting* [23], which ensures ensemble diversity by seeking to train new weak classifiers on previously misclassified data. Classifiers are combined using weighted majority voting, as described in section 3.3.1.

The pseudocode for the Adaboost.M1 algorithm is provided in Figure 4.7. For a maximum of T iterations, subsamples of data are selected for training a new classifier; samples are selected according to an instance-weighting probability distribution D_t. The

```
**Algorithm Adaboost.M1**
Input: For a dataset $\mathfrak{D} = \left\{x_i^t \in X; \omega_i^t \in \Omega = \{1,\dots,c\}\right\}, i = 1,\cdots,N$
    • Error distribution $D$ over all $N$ instances
    • Weak learning algorithm **BaseClassifier**
    • Integer $T$, specifying the number of classifiers to generate
**Do** for $t = 1,2,\dots,T$
  **If** $t = 1$, initialize weight vector $w_1(i) = D(i)$ for $i = 1,\dots,N$
  1. Set $\boldsymbol{D}_t = \boldsymbol{w}_t / \sum_{i=1}^{m} w_t(i)$ so that $D_t$ is a distribution    (4.1)
  2. Randomly choose training $TR_t$ and testing $TE_t$ subsets from $\boldsymbol{D}_t$
  3. Call **WeakLearn** with training data $TR_t$
  4. Receive hypothesis $h_t : X \to \Omega$, and calculate error of $h_t$:
       $\varepsilon_t = \sum_{i:h_t(x_i) \neq \omega_i} \boldsymbol{D}_t(i)$    (4.2)
     **If** $\varepsilon_t > \frac{1}{2}$, set $T = t - 1$, and abort loop.
  5. Otherwise, compute normalized error $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$    (4.3)
  6. Set the new weight vector
       $w_{t+1} = w_t(i) \times \begin{cases} B_t, & if\ H_t(x_i) = \omega_i \\ 1, & otherwise \end{cases}$    (4.4)
Call **weighted majority** to combine hypothesis $H_t$ and output the final hypothesis
       $H_{final} = \arg\max_{\omega \in \Omega} \sum_{k=1}^{K} \sum_{t:H_t(x)=\omega} \log\left(\frac{1}{B_t}\right)$    (4.5)
```

Figure 4.7: Adaboost.M1 Algorithm

distribution, being related to classification error, gives higher probability to instances (to be selected for the next classifier training data) which were previously misclassified.

Once trained, the t^{th} classifier (hypothesis) h_t is evaluated over all training data $\mathfrak{D}_t$, and its error ε_t is computed by summing the distribution D_t over all misclassified instances. Classifier weights are assigned in Equation 4.4. The entire process is repeated until T classifiers are created *or* the error ε_t over misclassified instances exceeds ½.

4.3.2 LEARN^{++}

Learn^{++} [70;77] is an incremental learning algorithm is inspired by Adaboost.M1 and is designed specifically for learning from a stationary system from which data are incrementally acquired in batches. Learn^{++} was designed essentially to handle data with virtual drift; that is, where information about a single distribution is not readily available and *appears* to be changing when, in fact, it is not. This can include small changes in class balance and even a *perceived* drift in the class-conditional probabilities when, in fact, the true distribution is unchanging.

Different from the Adaboost framework which learns from a single frame of data, Learn^{++} receives consecutive windows of training data $\mathfrak{D}_k$, as seen in the pseudocode in Figure 4.8. From each of these datasets, an ensemble of T_k classifiers is trained as in Adaboost.M1 (Steps 1-6). As in Adaboost, Learn^{++} utilizes a distribution over the data instances to increase the likelihood of selecting previously misclassified instances for training the next classifier. The Learn^{++} approach differs from Adaboost in that the distribution is updated based on the entire ensemble decision, rather than the decision of the most recent classifier. Learn^{++} uses the ensemble decision to maximize learning of previously unknown data; this instance selection approach is especially applicable when new classes are presented. When a new dataset arrives, the distribution is re-initialized by evaluating the entire ensemble and initializing the distribution (Steps 5-6), and the boosting process repeats. Weighted majority combines classifiers from each sub-ensemble to provide a final hypothesis (Equation 4.13).

Considering the definition of true concept drift, we realize that this architecture is inappropriate in some ways. First, Learn^{++} demands a sufficiently large window so that k

classifiers can be constructed on different data within that window. Assuming a large training window size (e.g. 200) increases the propensity for training to occur on data from multiple environments. A small window decreases the boosting effect of the ensemble, as either the number of classifiers *k* or the size of training set TR_t must be limited.

```
**Algorithm Learn++**
Input: For each dataset $\mathfrak{D}_k$  $k = 1,2 \dots K$
    • Training data $\{x_i^t \in X; \omega_i^t \in \Omega = \{1, \dots, c\}\}, i = 1, \cdots, m^t$.
    • Weak learning algorithm **BaseClassifier**
    • Integer $T_k$, specifying the number training iterations
**Do** for $k = 1,2, \dots, K$
    **If** $k \neq 1$, Set $t = 0$ and Go to Step 5 to adjust the weights
    **Do** for $t = 1,2, \dots, T_k$
      1. Set $D_t = w_t / \sum_{i=1}^{m} w_t(i)$ so that $D_t$ is a distribution        (4.6)
      2. Randomly choose training $TR_t$ and testing $TE_t$ subsets from $D_t$
      3. Call **WeakLearn** with training data $TR_t$
      4. Receive hypothesis $h_t: X \to \Omega$, and calculate error of $h_t$:
         $\varepsilon_t = \sum_{i:h_t(x_i) \neq \omega_i} D_t(i)$        (4.7)
         **If** $\varepsilon_t > \frac{1}{2}$, set $t = t - 1$, discard $h_t$ and return to Step 2.
         Otherwise, compute normalized error $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$        (4.8)
      5. Call weighted majority to obtain composite hypothesis
         $H_t = \arg\max_{\omega \in \Omega} \sum_{t:h_t(x)=\omega} \log\left(\frac{1}{\beta_t}\right)$        (4.9)
         And compute composite error
         $E_t = \sum_{i:H_t(x_i) \neq \omega} D_t(i)$        (4.10)
         **If** $E_t > \frac{1}{2}$, set $t = t - 1$, discard $H_t$ and go to step 2.
      6. Set $B_t = \frac{E_t}{1} - E_t$ to normalize error; update instance weights        (4.11)
         $w_{t+1} = w_t(i) \times \begin{cases} B_t, & if\ H_t(x_i) = \omega_i \\ 1, & otherwise \end{cases}$        (4.12)
Call **weighted majority** to combine hypothesis $H_t$ and output the final hypothesis
    $H_{final} = \arg\max_{\omega \in \Omega} \sum_{k=1}^{K} \sum_{t:H_t(x)=\omega} \log\left(\frac{1}{B_t}\right)$        (4.13)
```

Figure 4.8: Learn^{++} Algorithm Pseudocode

Second, the classifier voting weights B_t are assigned once (when the classifier is trained) and never updated. Static weighting is problematic in concept drifting data because classifiers probably will not have the same relevance in new environments as when they were first trained. Thus, for non-stationary environments, it is desirable to periodically update classifier weights.

In summarizing much of the prior research that has been accomplished in the area of learning in non-stationary environments, there are a number of weaknesses that arise: first, a majority of learning methods trades the ability to use prior knowledge (stability) in recurring contexts for a smaller ensemble (plasticity). Second, the idea of long-term forgetting (tradeoff between limiting memory and handling recurring environments) is not discussed. Finally, the foremost ensemble approaches seldom use *active* drift detection techniques; those which *do* actively detect drift tend to be implemented using a weak classification method.

CHAPTER 5: THE LEARN^{++}.NSE ALGORITHM

Here we present Learn^{++}.NSE (Non-Stationary Environment), an ensemble-based classification algorithm, as a new framework for building a knowledge base in a changing environment. The algorithm was created in order to handle some of the key issues in non-stationary environments that its predecessor Learn^{++} and many other current methods do not consider, especially regarding knowledge categorization and retention for handling recurring environments.

We also compare this new framework to the approaches that have been proposed in the literature and introduce new methods which are either original or adaptations of prior work. We do so not only for comparison purposes, but also to improve the algorithm while conforming to goals set forth by the learning model (see Section 3.1).

The general framework for Learn^{++}.NSE is as follows (see Figure 5.1 for algorithm pseudocode): the knowledge base is initialized by creating a classifier on the first available batch of data. Once prior knowledge (an existing ensemble) is created, this knowledge base is evaluated on newly arriving training data (Step 1). Step 2 effectively problematizes (Section 2.3) the data by identifying examples which are not recognized by the current knowledge. Step 3 adds to the current knowledge base by training a new classifier on the current training data. Step 4 is the first step in categorizing knowledge with respect to the current environment. Each classifier in the knowledge base (the existing ensemble) is evaluated on the current training data. Because previously unknown data have been identified in step 2, the penalty for misclassifying such instances is reduced in the error calculation. In other words, more credit is given to a classifier that is capable of classifying previously unknown instances while classifiers which misclassify data (especially previously *known* data) are penalized. In Step 5, classifier error is weighted with respect to time so that recent competence (error rate) is considered more heavily for categorizing knowledge. Step 6 computes the voting weights using only a classifier's classification error (relevance), regardless of the classifier's age. Ignoring classifier age in computing classifier voting weights can be considered an advantage for the computer model over human learning in that human recollection can be hindered by memory age-related forgetfulness. Step 7 completes the model, showing that the knowledge base (classifier ensemble) can be tested at any point to provide the best possible decision. These steps are described in more detail in the following section.

5.1 Algorithm Description

As the current training data $\mathfrak{D}_t$ become available, Learn^{++}.NSE is presented with training data x_t consisting of m instances with corresponding class labels ω_t (note: unless specified otherwise, the subscript refers to the primary time index, and the secondary index is indicated by parenthesis as in $\varepsilon_t(k)$, $D_t(i)$ etc.). As in the original Learn^{++}, we begin with initializing the first classifier weight and a *penalty distribution* over all m_t instances during the first time step such that all instances have equal penalty.

$$D_1(i) = 1/m_1 \qquad (5.1)$$

We use the term penalty distribution because it differs from distribution created in the predecessor Learn^{++}. Here, the distribution is used to assign error (in Step 4) and *not* for subset sampling (all data $\mathfrak{D}_t$ is used for training). Based on the assumption that the environment may change at any time and that the data are received in small chunks, the algorithm begins by training a classifier using the entire training dataset $\mathfrak{D}_t$ in Step 3.

Once the knowledge base has been constructed (at least one classifier has been created) at time $t > 1$, subsequent iterations will re-initialize the penalty distribution D_t over all instances based on ensemble predictions on the most recent training data $x_t(i)$. Ensemble error E_t is obtained by summing across all misclassified instances (where the composite hypotheses of the ensemble H_{t-1} on training data x_t do not match the correct class labels ω_t), and divided by the total number of instances m_t.

$$E_t = \sum_{i=1}^{m_t} \left(\frac{1}{m_t}\right) \cdot [\![H_{t-1}(x_t(i)) \neq \omega_t(i)]\!] \tag{5.2}$$

In Step 2, we introduce an instance weighting method, where correctly classified instances are given lower weight (proportional to ensemble error) and misclassified instances receive a maximum weight of $1/m_t$.

$$w_t(i) = \frac{1}{m_t} \cdot \begin{cases} E_t, H_{t-1}(x_t(i) = \omega_t(i)) \\ 1, \quad otherwise \end{cases}, \quad i = 1, \cdots, m_t \tag{5.3}$$

The penalty distribution D_t is represented by a normalization of these weights (dividing each weight by the sum of weights) such that the area under the distribution will equal 1:

$$D_t = \frac{w_t}{\sum_{i=1}^{m_t} w_t(i)} \tag{5.4}$$

Instances which were previously misclassified are always given a higher penalty weight than the penalty for instances that were correctly classified by the ensemble. This ensures that in Step 4, classifiers will be judged most severely on what the ensemble does not already know, and instances which the ensemble *does* recognize will receive less consideration. The relativity of penalties is based on the overall error of the ensemble. When the ensemble does well, misclassified points get higher relative penalty weight, and when the ensemble performs poorly on the new data, misclassified data receives a penalty weight more similar to that of the correctly classified examples. For instance, a high ensemble error of $E_t = 0.75$ corresponds to a weight of $w_t(i) = \frac{E_t}{m_t} = 0.075$ for a misclassified instance (batch size $m_t = 10$), and all correctly classified instances receive a weight of $w_t(i) = \frac{1}{m_t} = 0.1$. If ensemble error is low (e.g. $E_t = 0.1$), correctly classified instances still receive a weight of $w_t(i) = 0.1$, but misclassified instances receive a weight of $w_t(i) = 0.01$. The penalty weight of misclassified instances with high ensemble error ($w_t(i) = 0.75$) is more similar to the weight of correctly classified instances ($w_t(i) = 0.1$) than the penalty weight of misclassified instances with low ensemble error ($w_t(i) = 0.01$).

In Step 4, the error ε_t of each classifier is evaluated on the training data from the current environment, $\mathfrak{D}_t$:

$$\varepsilon_t(k) = \sum_{i=1}^{m_t} D_t(i) \cdot [\![h_k(x_t(i)) \neq \omega_t(i)]\!], \quad k = 1, \ldots, t \tag{5.5}$$

Instead of assigning the same error to each misclassified instance, the penalty weight distribution D_t is used; for each misclassified instance i (hypothesis $h_k(x_t(i))$ does not match true class label $\omega_t(i)$), the associated penalty weight is summed to obtain the error of the classifier on the training data at time t. Using the penalty weight distribution effectively gives a classifier more credit for correctly classifying data that the ensemble did not know; in other words, classifiers which perform well on novel data are deemed more relevant than others.

```
Input: For each dataset $\mathfrak{D}_t$  $t = 1,2,\ldots$
  - Training data $\{x_t(i) \in X; \omega_t(i) \in \Omega = \{1,\ldots,c\}\}$,   $i = 1,\cdots,m_t$.
  - Supervised learning algorithm BaseClassifier
  - Ensemble size $s$
  - Sigmoid Parameters $a, b$
Do for $t = 1,2,\ldots$
    If $t = 1$, Initialize $D_1(i) = w_1(i) = 1/m_1$, $\forall i$, Go to step 3. Endif      (5.1)
    1. Compute error of the existing ensemble on new data
       $E_t = \sum_{i=1}^{m_t} \left(\frac{1}{m_t}\right) \cdot [\![ H_{t-1}(x_t(i)) \neq \omega_t(i) ]\!]$      (5.2)
    2. Update and normalize instance weights
       $w_t(i) = \frac{1}{m_t} \cdot \begin{cases} E_t, & H_{t-1}(x_t(i)) = \omega(i) \\ 1, & otherwise \end{cases}$,   $i = 1,\cdots,m_t$      (5.3)
       Set $D_t = \frac{w_t}{\sum_{i=1}^{m_t} w_t(i)} \Rightarrow D_t$ is a $penalty$ distribution      (5.4)
    3. Call BaseClassifier with $\mathfrak{D}_t$, obtain $h_t: X \to \Omega$
    4. Evaluate all existing classifiers on new data $\mathfrak{D}_t$
       $\varepsilon_t(k) = \sum_{i=1}^{m_t} D_t(i) \cdot [\![ h_k(x_t(i)) \neq \omega_t(i) ]\!]$,   $k = 1,\ldots,t$      (5.5)
       If $\varepsilon_t(k = 1) > 1/2$, generate a new $h_t$.
       If $\varepsilon_t(k > t) > 1/2$, set $\varepsilon_t(k) = 1/2$,
       $\beta_t(k) = \frac{\varepsilon_t(k)}{(1 - \varepsilon_t(k))}$,   $k = 1,\ldots,t$      (5.6)
    5. Compute the time-weighted average of all normalized errors
       for $k^{th}$ classifier $h_k$ For sigmoid parameters $a, b \in \mathbb{R}$
       $\sigma_t(k) = 1/\left(1 + e^{-a(t-k-b)}\right)$, $\sigma_t(k) = \sigma_t(k) / \sum_{j=0}^{t-k} \sigma_{t-j}(k)$      (5.7)
       $\bar{\beta}_t(k) = \sum_{j=0}^{t-k} \sigma_{t-j}(k) \beta_{t-j}(k)$,   $k = 1,\ldots,t$      (5.8)
    6. Calculate classifier voting weights
       $W_t(k) = \log\left(\frac{1}{\bar{\beta}_t(k)}\right)$,   $k = 1,\ldots,t$      (5.9)
    7. Obtain the final hypothesis
  → $H_t(x_t) = \arg\max_{\omega} \sum_k W_t(k) \cdot [\![ h_k(x_t) = \omega ]\!]$      (5.10)
```

Figure 5.1: Learn^{++}.NSE Algorithm Pseudocode

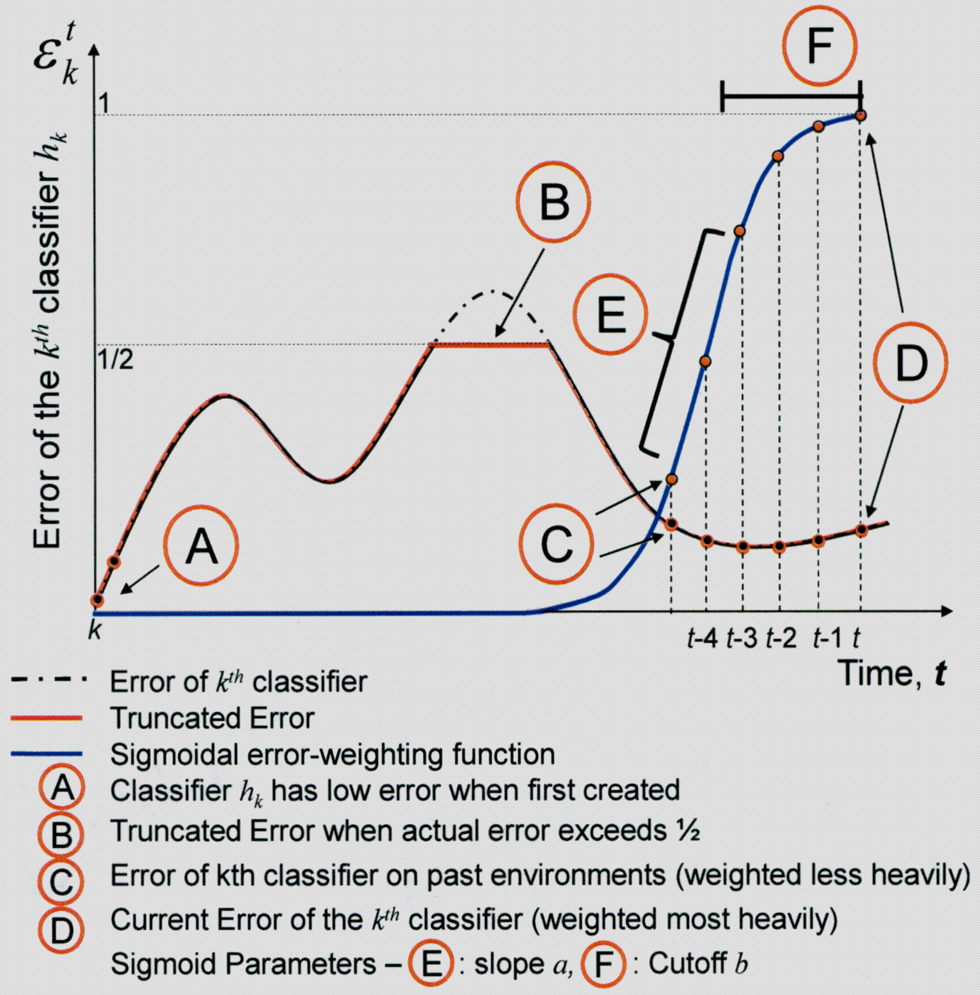
If the newest classifier is unable to perform better than ½, it is discarded since it is not likely to have a positive contribution to the ensemble, and a new classifier is trained in its place. Any other classifier error greater than ½ is saturated at ½ so that all normalized error β in (Equation 5.6) will be mapped from 0 to 1, where 0 represents perfect classification, and 1 represents worst-case (½) classification, where classifier with especially high error will be given zero weight (yet only at that time step). Note that in a two-class problem, an error of ½ corresponds to 50% classification (random guess). With additional classes, the classification percentage of random guess decreases (e.g. 25% for four-class), making the goal of 50% more difficult to attain for any classifier. The effect of truncating error is illustrated in Figure 5.2. The purpose in setting the threshold at ½ is that it prevents negative weights from being calculated later in Step 6.

$$\beta_t(k) = \frac{\varepsilon_t(k)}{\left(1 - \varepsilon_t(k)\right)}, \quad k = 1, \dots, t \tag{5.6}$$

Step 5 introduces an error-weighting sigmoid function for computing the time-based weighted error $\bar{\beta}$. As discussed earlier, it is imprudent to determine classifier weights based on all error over time because classifiers should be organized/weighted according to their relevance at the current time. For this reason, using a sigmoid function over a classifier's error through time considers competence only in recent time step(s).

$$\sigma_t(k) = \frac{1}{\left(1 + e^{-a(t-k-b)}\right)} \rightarrow \sigma_t(k) = \sigma_t(k) / \sum_{j=0}^{t-k} \sigma_{t-j}(k) \tag{5.7}$$

Parameter a defines the slope of the sigmoid cutoff, and b refers to the number of prior errors to be considered before the cutoff. The effect of the sigmoid on classifier error can be seen in the following two figures. Figure 5.2 illustrates the effect of the sigmoid on the error $\varepsilon_{t,k}$ over time for a single classifier k in the ensemble. The sigmoid parameters (a and b) are also depicted, showing how a classifiers prior error is given less consideration than current error when multiplied by the sigmoid. From this picture, we can also infer the effect of changing the sigmoid parameters (cutoff and slope) to give more or less weight to a classifiers prior error. Figure 5.3 shows how the same sigmoid parameters are applied across the error matrix $\varepsilon_{t,k}$ of all k classifiers in the ensemble at time step t. Note that the sigmoid cutoff lies the same distance from the most recent error, regardless of the age of the classifier. If a classifier is created recently ($t > b$), the error will not even be cut off because all error is considered relevant in computing weights for recent classifiers.



ε_k^t
Error of the k^{th} classifier h_k
1
1/2
k
t-4 t-3 t-2 t-1 t
Time, t
A
B
C
D
E
F
Error of k^{th} classifier
Truncated Error
Sigmoidal error-weighting function
Classifier h_k has low error when first created
Truncated Error when actual error exceeds ½
Error of kth classifier on past environments (weighted less heavily)
Current Error of the k^{th} classifier (weighted most heavily)
Sigmoid Parameters – E : slope a, F : Cutoff b

Figure 5.2: The effect of sigmoid error weight for a single classifier over time [78]

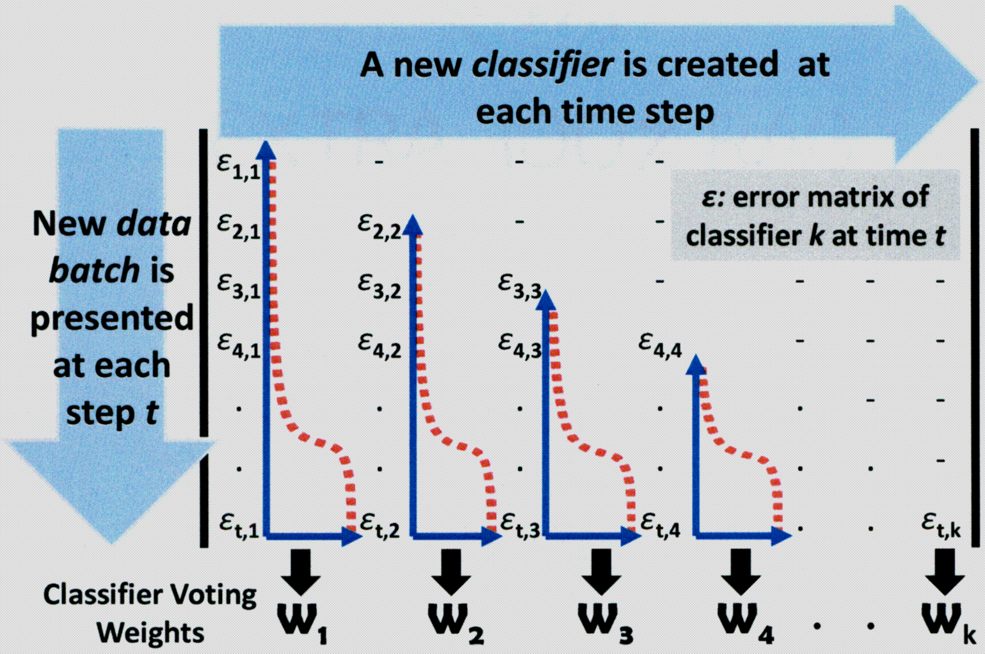


Figure 5.3: Time-based sigmoid error weighting

While many other algorithms use only a classifier's current error, here a sigmoid-based weight is applied across all error, for it is desirable to consider a classifier's error in recent time steps as there may be knowledge pertinent to the current environment. The sigmoid is applied individually to each classifier's error over time so that some (not all) prior error is considered in calculating classifier weights. Under this strategy, any classifier containing relevant knowledge, *regardless of the classifier's age*, can receive a significant voting weight. Classifier age itself has no direct effect on voting weight.

Once weighted appropriately with the sigmoid function $w_t(k)$, the classifier error is then normalized as a weighted sum, with weights obtained from the sigmoid function as shown in Equation 5.8.

$$\bar{\beta}_t(k) = \sum_{j=0}^{t-k} \sigma_{t-j}(k)\,\beta_{t-j}(k), \quad k = 1, \dots, t \tag{5.8}$$

The last phase of knowledge categorization occurs in Step 6, where the age-adjusted classifier weights are assigned. Log-based weighting [31] is assigned using the age-weighted error for each classifier in the ensemble (Equation 5-9).

$$W_t(k) = \log\left(\frac{1}{\bar{\beta}_t(k)}\right), \quad k = 1, \dots, t \tag{5.9}$$

Step 7 shows the formulation of the final ensembles weighted decision/hypothesis on unlabelled data, where the classifier weights are summed with respect to their respective class selection (Equation 5.10); the predicate $[\![\cdot]\!]$ will yield a 1 for classification of class y from classifier k. Classifiers with the high voting power will yield the most support for the class they choose. Testing can be performed any time following the learning phase and will provide the best possible decision on the current environment.

$$H_t(x_t) = \arg\max_{\omega} \sum\nolimits_k W_t(k) \cdot [\![h_k(x_t) = \omega]\!] \tag{5.10}$$

5.2 Controlled Forgetting

5.2.1 Permanent Pruning

The learning model described in Section 5.1 can be adjusted to accommodate any controlled forgetting mechanism. Here, we introduce three possible permanent pruning methods, all of which use the ensemble size threshold approach to limiting the ensemble size. The first option is to remove the oldest classifier in the ensemble to make room for knowledge carried by a new classifier. The second uses a classifier's most recent error β_k^t as the basis for removing knowledge. The third approach, using *averaged* error of a classifier over time, is a strategy which is original to our work. A possible reason for neglect in the literature is the small ensemble size that is commonly used. Small ensembles are used explicitly for achieving plasticity, and therefore a classifier's prior performance (from previous environments) is irrelevant. The criterion for removing classifiers is the error *on the most recent data,* and to consider averaged error from previous time steps would be misleading. Also note that some methods which use a small ensemble size do not use a weighting method to categorize or organize classifiers. Rather, they assume that the ensemble contains knowledge which is completely and equally pertinent to the current environment.

Within the Learn^{++}.NSE algorithm, the task of (permanent) pruning (pseudocode in Figure 5.4) – if used - occurs immediately following the calculation of the voting weights. Steps 7-8 in the pruning pseudocode below are intended as replacement for Step 7 in the original Learn^{++}.NSE pseudocode in Figure 5.1.

```
< Refer to Learn++.NSE pseudocode for Steps 1-6 >
7. Permanent Ensemble Pruning:
 If $t > s$
  A. Age-based: Remove $h_{t-s}$ from the ensemble
  B. Current error-based: Remove $h_{k^*}$ where
       $\beta_t(k^*) = \max \beta_t(k), \quad k = 1 \dots t$                    (5.11)
  C. Average error-based: Remove $h_{k^*}$ where
       $\beta_t(k^*) = \max\left(average(\beta_t(k))\right), \quad for\ k = 1 \dots t$   (5.12)
 Endif
8. Obtain the final hypothesis
   $H_t(x_i) = \arg\max_{\omega} \sum_s W_t(s) \cdot [\![h_s(x_i) = \omega]\!]$           (5.13)
```

Figure 5.4: Permanent Controlled Forgetting

In scenario A (*age-based pruning*), the oldest classifier in the ensemble (created $t-s$ time steps ago) is removed. In scenario B (*current error-based pruning*), the index $\beta_{k^*}^t$ of the classifier with highest error (β_k^tover k classifiers) is found using the max(·) function, and the corresponding classifier is removed:

$$\beta_{k^*}^t = \max \beta_k^t, \quad k = 1 \ldots t \tag{5.11}$$

In scenario C (*average error-based pruning*), the average error over time for each classifier is calculated, and is then used for the pruning criterion as in scenario B:

$$\beta_{k^*}^t = \max\left(average(\beta_k^t)\right), \quad for\ k = 1 \ldots t \tag{5.12}$$

The final hypothesis calculated in Step 8 (Equation 5.13) is identical to that of the original Learn^{++}.NSE.

In using averaged error-based pruning, we reason that a larger ensemble size can and should be used, as it more closely represents the vast knowledge base in human memory. Increasing the ensemble size adds a minor complication, since it also increases the chance of harboring irrelevant knowledge (baggage); thus, it is necessary to employ a weighting method such as the one in the Learn^{++}.NSE algorithm. The overall purpose of average-error pruning with a large ensemble size is *not* to keep only relevant knowledge, for this task is delegated to the weighting method. Instead, average error-based pruning removes knowledge that is *generally* less useful over the course of time. It is intended that pruning with average error will also have implications with regard to recurring environments as it should retain useful classifiers.

5.2.2 TEMPORARY PRUNING

Temporary pruning (expert selection) is primarily employed for baggage reduction; baggage takes the form of old classifiers which are no longer relevant to the current environment. Although the Learn^{++}.NSE framework combats the influence of irrelevant classifiers by its dynamic weighting technique, baggage is practically inevitable when the knowledge base is constantly expanding. Two methods are proposed (Figure 5.5) which can be substituted directly into Step 6 in the original Learn^{++}.NSE framework. The first method, derived from the Adaptive Classifiers Ensemble (ACE) algorithm [35], selects only the classifiers which lie within a confidence interval around the highest performing classifier on the current training data. The confidence interval is based on classification error $A_{k,t}$ of all k classifiers on the most recent training data at time t. Classification error is calculated using Laplace's rule of succession, which [35] suggests is more accurate for small sample sizes (m_t).

$$A_t(k) = \frac{\left(\sum_{i=1}^{m_t} [\![h_k(x_i) \neq \omega_i]\!]\right) + 1}{s + 2} \tag{5.14}$$

The lower bound $A_t^{l,\alpha}(k)$ for classifier performance is computed using the *score confidence interval* [79] with confidence of $100(1-\alpha)\%$.

$$A_t^{l,\alpha}(k) = \frac{m_t}{m_t + z_{\alpha/2}^2}\left(\left(1 - \varepsilon_t(k)\right) + \frac{z_{\alpha/2}^2}{2m_t} + z_{\alpha/2}^2\sqrt{\frac{\varepsilon_t(k)\left(1 - \varepsilon_t(k)\right)}{m_t} + \frac{z_{\alpha/2}^2}{4{m_t}^2}}\right) \tag{5.15}$$

Once the confidence boundary has been attained, the expert selection process is simple. For any classifier k, a weight is assigned only if the classifier's performance $A_t(k)$ surpasses the lower (l) confidence bound $A_t^{l,\alpha}$.

$$W_t(k) = \begin{cases} \log\left(\frac{1}{\bar{\beta}_t(k)}\right) & if \;\; A_t(k) > \max_k A_t^{l,\alpha}(k) \\ 0, & otherwise \end{cases} \tag{5.16}$$

The confidence interval criterion produces a variable percentage of classifiers selected at any given time-step, with sensitivity related to the confidence interval parameter α.

The second temporary pruning method ensures a minimum sub-ensemble size by selecting a constant n percent of the top performing classifiers according to the classifier

```
< Refer to Learn++.NSE pseudocode for Steps 1-5 >
 6. Temporary Ensemble Pruning:
  If $t > s$
   1. Confidence Interval Criterion:
     Calculate performance of classifier $k$:
     $A_t(k) = \frac{\left(\sum_{i=1}^{m_t} [\![ h_k(x_i) \neq \omega_i ]\!]\right) + 1}{m_t + 2}$     (5.14)
     Compute lower confidence $100(1-\alpha)\%$ over $k$ performances:
     $A_t^{l,\alpha}(k) = \frac{m_t}{m_t + z_{\alpha/2}^2}\left(\left(1 - \varepsilon_t(k)\right) + \frac{z_{\alpha/2}^2}{2m_t} + z_{\alpha/2}^2\sqrt{\frac{\varepsilon_t(k)\left(1-\varepsilon_t(k)\right)}{m_t} + \frac{z_{\alpha/2}^2}{4{m_t}^2}}\right)$     (5.15)
     Compute classifier weights:
     $W_t(k) = \begin{cases} \log\left(\frac{1}{\bar{\beta}_t(k)}\right) & if \;\; A_t(k) > \max_k A_t^{l,\alpha}(k) \\ 0, & otherwise \end{cases}$     (5.16)
   2. Top Percentage Criterion:
     Compute classifier weights
     $W_t(k) = \log\left(\frac{1}{\bar{\beta}_t(k)}\right) \;\; for\; k = 1, \dots, t$     (5.17)
     $W_{sort} = sort(W_t(k)), \;\; W_{n\%} = W_{sort,\; round\,(n*t)}$     (5.18)
     $W_t(k) = \begin{cases} W_t(k) \;\; if\; W_t(k) \geq W_{n\%} \\ 0, \;\; otherwise \end{cases}$     (5.19)
  Endif
< Refer to Learn++.NSE pseudocode for Step 7 >
```

Figure 5.5: Temporary Controlled Forgetting

weight $W_t(k)$. The process begins by computing the weights $W_t(k)$ from time-weighted error as in the original algorithm (Equation 5.17). Weights are then sorted from highest to lowest (W_{sort}), and the weight of classifier in the n^{th} percentile, $W_{n\%}$ is attained.

$$W_{sort} = sort\big(W_t(k)\big), \quad W_{n\%} = W_{sort,\ round\,(n*t)} \tag{5.18}$$

The final step is to zero out all weights corresponding to classifiers which fall below the $W_{n\%}$ threshold.

5.3 Drift Detection

We now discuss the role of the *supervisor* in the learning process Recall that he supervisor-based approach is known as *scaffolding* (see Section 2.3) and provides us important guidelines for improving the learning process including:

- *Problematizing* – emphasizing conflicts between new data and the learners current knowledge
- Monitoring both the flow of incoming data (to be learned) and the learner's performance for change
- Controlling the flow of incoming data such that it is broken into learnable chunks
- *Fading* – ceasing the training process when an environment is unchanging to mitigate redundant knowledge

The task of *problematizing* is inherent within the Learn^{++}.NSE algorithm, and is the explicit goal of using the penalty weight distribution D_t for evaluating classifier error. The aspect of monitoring data and ensemble performance is less explicit, as the Learn^{++}.NSE algorithm is *passive* by nature. That is, we assume that all new data comes from a new environment that may (or may not) be changing continuously. Therefore, Learn^{++}.NSE constantly augments and organizes the knowledge base according to that environment. This approach appears to be consistent with the human-based learning system. Yet, we also seek to improve upon the learning model by experimenting with some aspects of *active* drift detection; this will allow us to investigate the aspect of scaffolding known as *fading*.

The survey of literature regarding active drift detection methods leads to a number of considerations with regard to complexity and sensitivity. An algorithm should add complexity *only* to the degree that it (1) will provide accurate results, and (2) will yield relevant information that can be acted upon. For this reason, we seek to *reduce* complexity, desiring to know only if significant change (in data distribution) has occurred, and not needing to know what type of change has occurred (this is irrelevant because it would not affect our approach).

A performance metric appears to be sufficient for measuring the source of changes in the data (i.e. which probability is changing). Although it does not explicitly reveal the type of drift that is occurring, performance-based drift detection is a clear indication of change in a learner's ability (or lack thereof) to track the new environment. The primary assumption behind the approach is that a learner's ability to correctly classify data in a stationary environment will fall within some probability distribution, and performance will shift from that performance distribution when drift occurs. This assumption translates to a likelihood approach, where the learner's performance will tend toward a null distribution (no drift), or some alternative distribution (drift).

Note that a performance-based approach is insensitive to the virtual drift problem; that is, an apparent change in class-conditional probability $P(x|\omega)$, posterior probability

$P(x)$, or priors $P(\omega)$ in the training data may alter performance although there is no change in the *true* data distribution. However, from a learning aspect, data that is virtually drifting should be learned as well, since it is relevant (and unknown) knowledge about the environment.

The advantage of a likelihood approach in performance-based drift detection is that a normal distribution can be assumed. Therefore, we propose a detection method similar to that described in [44] for CUSUM-based drift detection of the *data* distribution. What appears to be a rather complex method can be significantly simplified when tracking performance since (1) only one dimension (classification accuracy) is considered, and (2) mean and variance are sufficient descriptors of the (Gaussian) distribution.

The drift detection method (pseudocode in Figure 5.6, pictured in Figure 5.7) is a batch-based approach where data arrives in consecutive frames or windows (as in Learn^{++}.NSE). The drift detection test begins with a *training* phase. In Step 1, a drift-detection classifier (with corresponding hypothesis h_{dd}) is trained with an initial subset of training data of size M (this subset may include instances from consecutive data batches). The drift-detection classifier is a separate compartment of the knowledge base and is not included in the decision ensemble. Using a separate single classifier isolates the performance metric from the effects of ensemble learning, which is often characterized by an *increase* in performance as knowledge is acquired. For instance, in a stationary environment, performance is expected to increase as knowledge (classifiers) is added to the ensemble, whereas a single classifier will have a more consistent level of performance. Conversely, in a changing environment, an ensemble may be able to adapt

```
Input:
  • Configuration size $CS$ for drift detection
  • Training size $M$ for drift detection classifier
  • Training data $\left\{x_i^t \in X; \omega_i^t \in \Omega = \{1, \dots, c\}\right\}, i = 1, \cdots, m_t$
Initialize $t_{drift} = 0$
For $t = 1,2 \dots$
If $t - t_{drift} < CS$
  1. Train drift detection classifier $h_{DD}$ on $M$ examples
  2. Configuration accuracy $a_{DD}(t - t_{drift}) = \sum_{i=1}^{m_t} 1 - \left(\frac{1}{m_t}\right) \cdot [\![h_{DD}(x_i) \neq \omega_i]\!]$      (5.20)
Elseif $t - t_{drift} = CS$
  3. Null hypothesis from mean & variance over $a_{DD}$: $\mu_0, \sigma_0^2$
  4. Upper and lower bounds: $\mu_1 = \mu_0 \pm \gamma z_{\alpha/2}\sqrt{\sigma_0^2}$      (5.21)
  5. Configuration Parameters
     - Log-likelihood
         $R(\tau) = \sum_{i=t_{drift}}^{\tau} \ln \frac{P_{\theta^1}(a_{DD}(i))}{P_{\theta^0}(a_{DD}(i))}$,   $for\ \tau = t_{drift} \dots t$      (5.22)
     - Minimum: $\eta(\tau) = \min_\tau R(\tau)$,   $for\ \tau = t_{drift} \dots t$      (5.23)
     - CUSUM parameter: $g(\tau) = R(\tau) - \eta(\tau)$,   $for\ \tau = t_{drift} \dots t$      (5.24)
     - Threshold: $\lambda = \max_\tau g(\tau)$,   $for\ \tau = t_{drift} \dots t$      (5.25)
Else $(t - t_{drift} > CS)$
  6. Compute accuracy $a_{DD}(t) = \sum_1^{m_t} 1 - \left(\frac{1}{m_t}\right) \cdot [\![h_{DD}(x_i) \neq \omega_i]\!]$
  7. $R(t) = \sum_{i=t_{drift}}^{t} \ln \frac{P_{\theta^1}(a_{DD}(i))}{P_{\theta^0}(a_{DD}(i))}$      (5.26)
  8. Compute $\eta(t)$, $g(t)$ using Eq. 5.23-5.24
  If $g(t) > \lambda$
    $drift(t) = 1$
    $t_{drift} = t$
  Else   $drift(t) = 0$   Endif
Endif
```

Figure 5.6: Performance-based CUSUM drift detection using log-likelihood

to a gradual change in the data because of its diverse/generalized knowledge. Therefore, the ensemble may be slow to degrade in performance due to the change in the environment. The performance of a single classifier will more quickly begin to degrade if the new environment differs from that on which the classifier was trained.

The next phase of the drift detection test is referred to as *configuration.* The configuration period is defined as number of time steps (or batches) controlled by configuration size parameter *CS*. Once the drift detection classifier is trained, it is then evaluated on a sequence of all CS training batches, yielding configuration accuracy a_{DD} in Step 2:

$$a_{DD}(t - t_{drift}) = \sum_{i=1}^{m_t} 1 - \left(\frac{1}{m_t}\right) \cdot [\![h_{DD}(x_i) \neq \omega_i]\!] \tag{5.20}$$

Accuracy a_{DD} is simply one minus the error, which is calculated as the sum of all instances where hypothesis h_{DD} on instance x_i does not match the true class label ω_i. Parameter configuration occurs once the classification accuracy has been attained over the configuration period (while $t - t_{drift} < CS$), where t is the current time step, and t_{drift} represents the beginning of the test (where the last drift was detected). Once $t - t_{drift} = CS$ is true, the null distribution θ^0 is formulated over the classifier's performance by calculating mean and variance of performance a_{DD} within the configuration period (Step 3). In Step 4, upper and lower bounds on mean and variance of null hypothesis θ^0 are used to create competing alternative hypotheses θ^1, θ^2, etc. An alternative hypothesis essentially represents a new environment for performance. The threshold for sufficient deviation in performance from the null hypothesis into an alternate/competing hypothesis is determined in Step 5. Here, the CUSUM parameters are calculated, beginning with the log-likelihood R (Equation 5.22). If the probability of performance lying within the null hypothesis $P_{\theta^0}(a_{DD}(i))$ is greater than the probability of lying within the alternative hypothesis $P_{\theta^1}(a_{DD}(i))$, the value for that particular performance will be negative. If the instance is more likely to lie within the alternative hypothesis, the value will be positive. Since the values are summed from the beginning of the experiment to time *t*, the trend of *R* is expected to be downward when accuracy a_{DD} is most likely in the null hypothesis, and vice versa.

$$R(\tau) = \sum_{i=t_{drift}}^{\tau} \ln \frac{P_{\theta}1(a_{DD}(i))}{P_{\theta}0(a_{DD}(i))}, \; for\; \tau = t_{drift} \; ... t \tag{5.22}$$

Next, the minimum of log-likelihood is calculated. The minimum value of the log-likelihood is the point at which there is greatest cumulative likelihood that the null hypothesis includes a_{DD}, and can be considered as the expected value or goal of the CUSUM test.

$$\eta(\tau) = \min_{\tau} R(\tau), for\; \tau = t_{drift} \; ... t \tag{5.23}$$

The CUSUM drift equation computes the difference between the current log-likelihood and the minimum:

$$g(\tau) = R(\tau) - \eta(\tau), \qquad for\; \tau = t_{drift} \; ... t \tag{5.24}$$

The deviation of log-likelihood $R(\tau)$ from $\eta(\tau)$ indicates a trend toward an alternative hypothesis (i.e. drift). Thus, the test is configured by setting a threshold λ is which represents the maximum observed deviation from the expected value η within the configuration sequence.

$$\lambda = \max_{\tau} g(\tau), \quad for\; \tau = t_{drift} \; ... t \tag{5.25}$$

The drift detection phase (Step 6-7) is similar to configuration in that the accuracy is continually calculated on incoming data batches, and the CUSUM parameters are recomputed at each time step. If the deviation $g(t)$ of likelihood $R(t)$ from the minimum $\eta(t)$ surpasses the maximum deviation observed within the configuration sequence (threshold λ), drift is detected by setting a flag for parameter $drift$ at time t. The drift time t_{drift} is set to the current time, and the drift test is reinitialized.

Such an approach is highly advantageous by nature of the CUSUM test's ability to track gradual drift. One tradeoff that can be expected is between over-generalization of the configuration distribution and subjectivity to noise. However, such is the case with most any threshold-based change detection system. The advantage in this case is the self-configuration of the threshold.

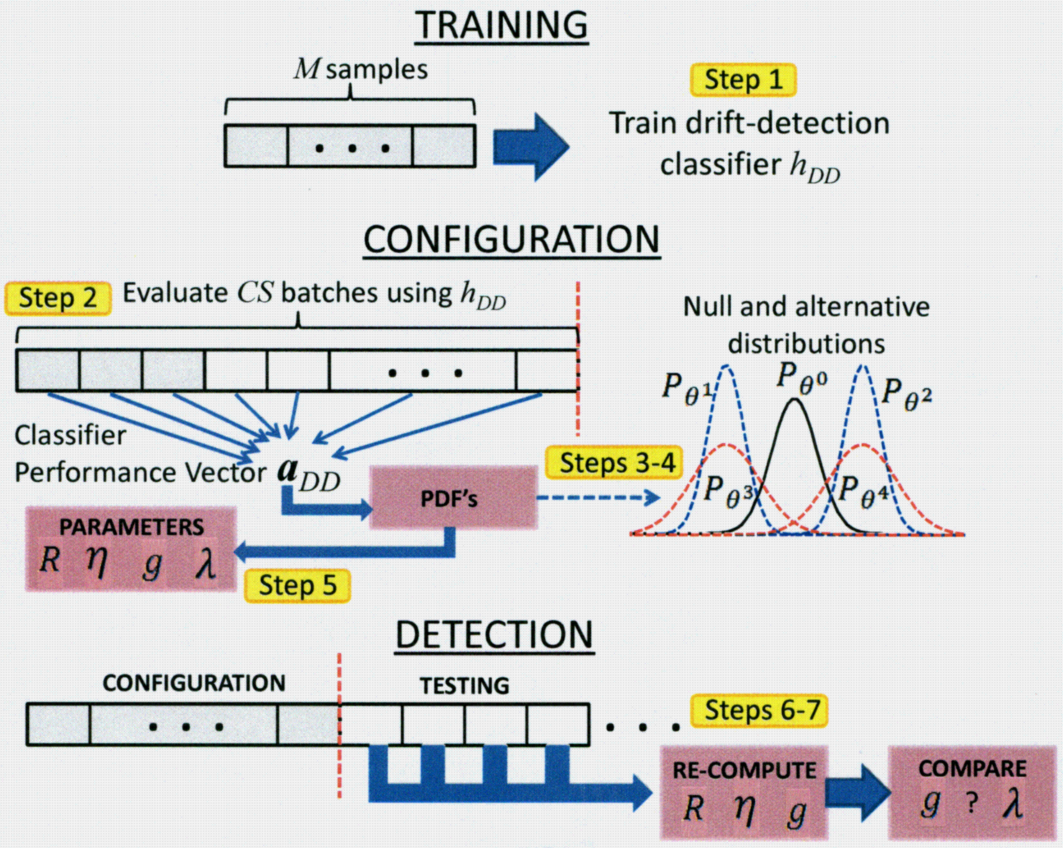


Figure 5.7: Drift detection procedure

5.3.1 FADING

Fading is the process whereby the learner is prompted to add to its knowledge base (i.e. train a additional classifier) *only* when a change or drift in the data is detected. The process works in tandem with the *active* drift detection mechanism and can be integrated into the framework of the Learn^{++}.NSE model with ease. Using the drift time indicator t_{drift} as the cue for learning, the knowledge base is augmented during the configuration phase of the drift detection method, in which the null performance hypothesis θ^0 is calculated. Provided that the configuration phase (CS batches) is long enough to acquire an accurate representation of the probability distribution over the classifier's performance (in drift detection), it is reasonable to assume that the same data contained in the interval $\{t_{drift}, t_{drift} + CS\}$ will also representative of the new environment that should be learned by the ensemble. Figure 5.8 shows the simple alteration in Step 3 of the Learn^{++}.NSE algorithm to employ fading. While the drift detection mechanism is being configured, new classifiers are trained and added to the knowledge base.

```
< Refer to Learn++.NSE pseudocode for Steps 1-2 >
 3. Fading
   If $t - t_{drift} < CS$
     Call BaseClassifier with $\mathfrak{D}^t$, obtain $h_t: X \rightarrow \Omega$
   Endif
< Refer to Learn++.NSE pseudocode for Steps 4-7 >
```

Figure 5.8: Learn^{++}.NSE Fading pseudocode

5.3.2 DYNAMIC SIGMOID ADJUSTMENT

The time-weighted sigmoid is designed to mitigate ensemble baggage by allowing only recent error to be considered when calculating classifier weights. Both intuition and characterization of these sigmoid parameters indicate that they can be adjusted, depending on the *type* of drift that is occurring, in order to attain optimal classifier weights. Specifically, three drift scenarios can be considered: no drift, concept change, and steady drift. These scenarios can be easily detected using the drift detection mechanism. Once the testing phase begins, a counter (t_{stable}) is instantiated, and will increment for every time step when no drift is detected. The duration of stability allows us to quantify how frequent drift is occurring, and categorize the following drift scenarios:

1) *No drift*: defined by an extended period during which drift is not detected (stable time t_{stable} exceeds stability threshold γ). When calculating weights, more prior error can be considered since the environment has remained unchanged. The sigmoid cutoff can be slowly increased until some saturation point (b_s) which is suitable for a stationary environment. This increase toward the saturation point is depicted in Figure 5.9.
2) *Concept change*: defined for our purposes as a drift detected after a period of stability. In this case, weights should be determined by classifiers most recent performance, and the cutoff parameter should be drastically reduced (b_d) to accommodate the new environment. After this period, the cutoff can be incrementally increased by a factor ϕ_d in the direction of the optimal parameter for steady drift as seen in Figure 5.9.
3) *Steady drift*: drift is continuously being detected, and the sigmoid cutoff should incrementally increase by a factor ϕ_s after a period of *concept change* and should converge to some optimal condition b_o.

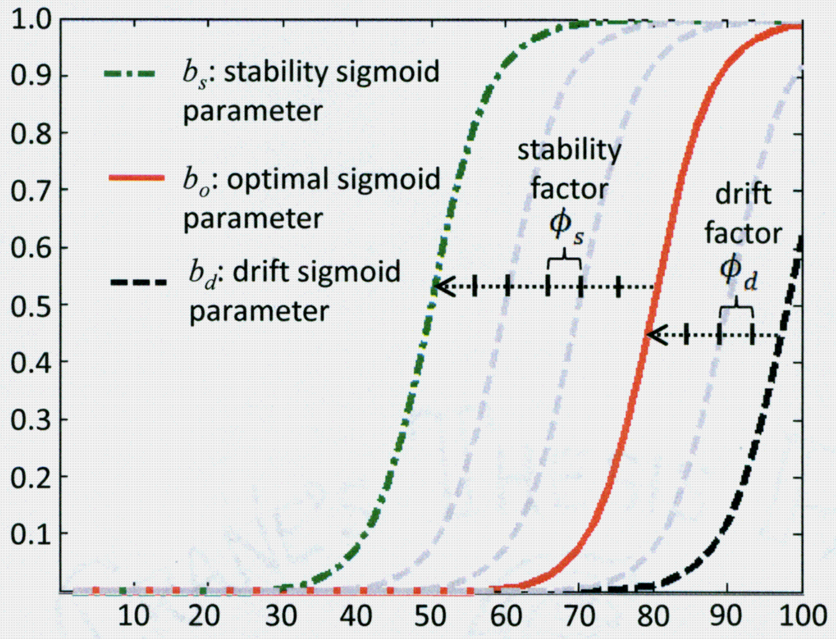


Figure 5.9: Illustration of dynamic sigmoid cutoff parameter b

The heuristic approach in Figure 5.10 relates the regularity of drift detections to the parameter b which controls the amount of error to be included when computing classifier weights. The justification for this approach is seen in a characterization of the cutoff parameter b (see results in section 6.11.1). Alternate parameters can be inserted directly into the sigmoid calculation. Note that only the cutoff parameter b is considered for adjustment; characterization of the slope parameter a (see Appendix F) reveals that it has little or no relation to the drift scenario.

When drift is detected (that is, when current time t equals time of drift t_{drift}), we check for stability (if t_{stable} exceeds threshold γ) and assign the either the drift factor b_d if the environment was previously stable, or assign optimality constant b_o when the environment was previously drifting.

$$b = \begin{cases} b_d, & t_{stable} > \gamma \\ b_o, & otherwise \end{cases} \tag{5.27}$$

If drift is *not* detected at time t, we check for three situations by comparing stability time t_{stable} with threshold γ.

$$b = \begin{cases} b + \phi_d, & t_{stable} < \gamma \; and \; b < b_o \\ b + \phi_s, & t_{stable} > \gamma \; and \; b < b_s \\ b_o & otherwise \end{cases} \tag{5.28}$$

First, if concept change has recently been detected, b must be incremented by drift factor ϕ_d until it reaches the optimality parameter b_o. Second if the environment is stable, b is

```
< Refer to Learn++.NSE pseudocode for Steps 1-4 >
Input:
  • Optimality, stability, and drift constants $b_o, b_s, b_d$
  • Drift and stability factors $\phi_d, \phi_s$
  • Stability threshold, $\gamma$
  • Time of most recent drift $t_{drift}$
  5. Dynamic Sigmoid Adjustment for computing the time-weighted average
       of all normalized errors for the $k^{th}$ classifier $h_k$
     If $t = t_{drift}$
        $b = \begin{cases} b_d, & t_{stable} > \gamma \\ b_o, & otherwise \end{cases}$      (5.27)
        $t_{stable} = 0$
     Else
        $b = \begin{cases} b + \phi_d, & t_{stable} < \gamma \; and \; b < b_o \\ b + \phi_s, & t_{stable} > \gamma \; and \; b < b_s \\ b_o & otherwise \end{cases}$      (5.28)
        $t_{stable} = t_{stable} + 1$
     Endif

     For $a \in \mathbb{R}$
        $\sigma_k^t = \frac{1}{(1 + e^{-a(t-k-b)})}, \quad \sigma_k^t = \frac{\sigma_k^t}{\sum_{j=0}^{t-k} \sigma_k^{t-j}}$      (5.29)

        $\bar{\beta}_k^t = \sum_{j=0}^{t-k} \sigma_k^{t-j} \beta_k^{t-j}, \quad k = 1, \dots, t$      (5.30)
< Refer to Learn++.NSE pseudocode for Steps 6-7>
```

Figure 5.10: Dynamic sigmoid adjustment pseudocode

incremented by the stability factor ϕ_S until it reaches the stability parameter b_S. Otherwise, b is given the optimality parameter b_O for steady drift. Parameter b is used in the same way to calculate a classifier's time-weighted error as in the original method in Learn^{++}.NSE (Equations 5.29-5.30). The pseudocode in Figure 5.10 may be directly substituted in Step 5 of the Learn^{++}.NSE algorithm.

CHAPTER 6: EXPERIMENTATION AND RESULTS

6.1 MOTIVATION & ORGANIZATION

In this chapter, we provide an empirical analysis of Learn^{++}.NSE, along with its variations, improvements, and comparisons to its competitors. In doing so, we seek to answer the following questions:

- Is the proposed framework able to learn from a variety of non-stationary environments?
- Is an ensemble of classifiers better than a single classifier? (confirm Wang's assertion in [30])
- Is the Learn^{++}.NSE weighting method better than an alternative weighting technique or an unweighted ensemble? (contradicting Gao in [5])
- Which is preferable, online learning (e.g. DWM [29]) or the Learn^{++}.NSE batch-based approach?
- Is any type of controlled forgetting appropriate for any ensemble size?
    - Compare age vs. error-based vs. average error-based vs. Streaming Ensemble Algorithm (SEA), and temporary forgetting
- Is active drift detection worthwhile?
    - Should the ensemble *always* be learning?

- Can the alteration of the Learn^{++}.NSE in various drift situations successfully maximize the knowledge base?

Each of these topics is discussed based on learning from a variety of datasets derived synthetically or from real-world phenomenon. Because of the high dimensionality of most real-world datasets, it is difficult to visualize or quantify the drift that is occurring in any of the features. Statistical analysis can merely *suggest* whether real concept drift is occurring within the data, and consequently it is difficult to surmise *when* the environment actually changes. Thus, we introduce a number of synthetic datasets which have been infused with specific drift scenarios.

Two training/testing scenarios can be simulated in our experiments for training and testing on batches of the available data $\mathfrak{D}_t$, as depicted in Figure 6.1 - Figure 6.2. A batch of data can be imagined as a *window* which slides to include new data at each concurrent time step. Classification describes the case where training (x_{TR}) and testing (x_{TS}) data (size *m* and *n*, respectively) are selected randomly from the same environment or window. The percentage *b* determines how many examples of data $\mathfrak{D}_t$ are selected for training.

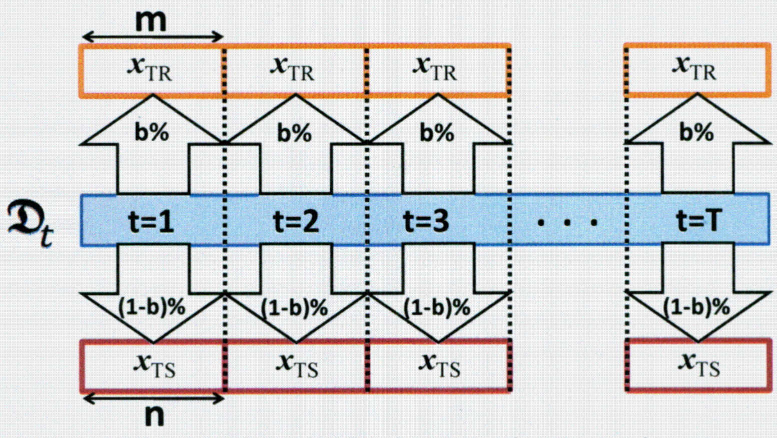


Figure 6.1: Classification Learning Scenario

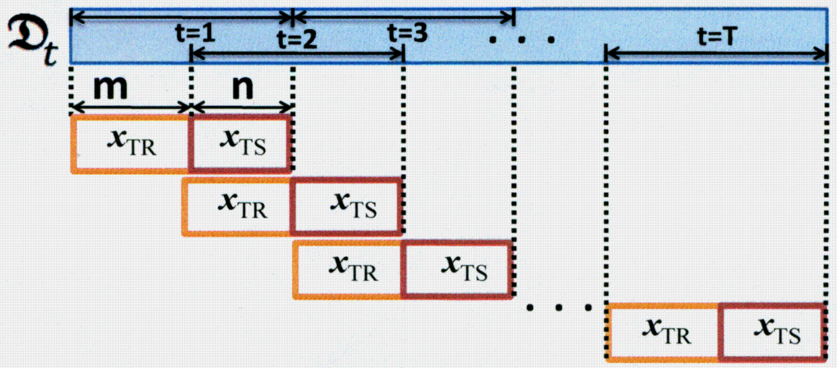


Figure 6.2: Prediction Learning Scenario

Prediction is a more difficult (and realistic) task, where testing data x_{TR} comes from a window directly following the training data x_{TS}. At each time step *t*, the windows shift such that the training window begins where the previous training data ended in the previous time step. All new data is first used as unlabelled testing data at time *t* and made available as training data at time $t + 1$.

6.2 Experimental Procedure & Performance Evaluation

As we seek to evaluate and compare multiple learning algorithms to determine which (if any) is superior, we will consider two approaches. We first calculate and compare the average performance measures over time for each method, along with statistical analysis to determine whether or not there is sufficient improvement of one method over another. Multiple comparison of means using one-way ANOVA (Analysis Of Variance) over repeated experiments determines significant difference among overall performances. The Tukey-Kramer method test statistic with 95% confidence is used to make the comparison.

The second and most important performance measure is the time-based plot of classification accuracy of hypothesis H_t on testing data (x_i, y_i):

$$A_t = 1 - \left(\frac{1}{m_t}\right) \cdot [\![H_t(x_i) \neq y_i]\!] \tag{6.1}$$

Observing performance (correctly classified instances divided by total instances m_t) over time enables analysis of the knowledge base throughout the learning process, specifically under various types of drift introduced throughout the course of an experiment. Each performance point represents the average accuracy across all testing examples at that time step (for the sake of visualization, further averaging may be necessary to create a smooth plot). Each performance curve is accompanied by a moving confidence interval based on at least 50 independent trials.

6.3 SYNTHETIC DATA

The use of synthetic datasets is necessary to provide accurate insight into the aforementioned algorithm comparisons. First, it allows the isolation and simulation of particular drift situations for observing an algorithm's behavior and performance. Since the *type*, *time*, and *rate* of drift is defined beforehand, we can observe the learning process before, during, and after drift with increased precision. Also, synthetic data can often be more easily visualized by reducing feature dimensionality to two or three. Finally, a Gaussian distribution is used for certain datasets, allowing us to calculate the Bayes classification error and measure an algorithm's performance based on that standard.

6.3.1 RANDOM GAUSSIAN DRIFT

Changing class distributions can be easily modeled as a set of Gaussian distributions with changing parameters (mean and/or variance) over time. A two-dimensional representation of the first dataset, a four-class drifting environment, can be seen in Figure 6.3; these distributions are governed by the parametric equations in Table 6.1. The rate of drift for a particular class is dependent on the difference in mean and variance at the beginning ($t = 0$) and end ($t = 1$) of a normalized time interval. This experiment consists of 200 time steps between $t = 0$ and $t = 1$, and each training window is a snapshot of 20 total points with equal prior probability. The knowledge base is tested at each time step using a uniformly spaced grid of 1,024 (32 by 32) points.

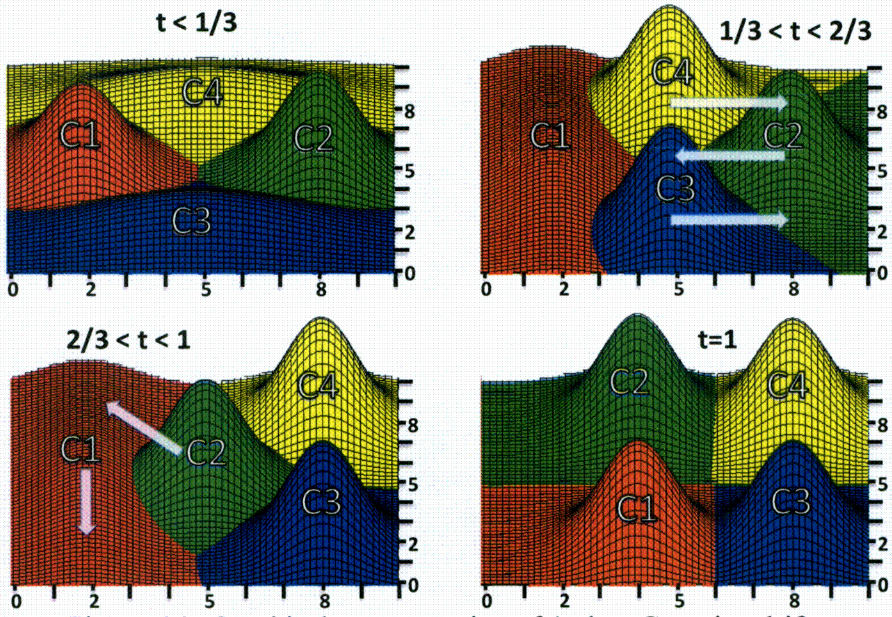


Figure 6.3: Graphical representation of 4-class Gaussian drift

Table 6.1:Parametric equations for drifting Gaussian data

	0<t<1/3				1/3<t<2/3			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_1	2	5	1	1+6t	2	5	1	3
C_2	8	5	1	1	8-9(t-1/3)	5	1	1
C_3	5	2	3-6t	1	5+9(t-1/3)	2	1	1
C_4	5	8	3-6t	1	5+9(t-1/3)	8	1	1

	2/3<t<1			
	μ_x	μ_y	σ_x	σ_y
C_1	2+6(t-2/3)	5-9(t-2/3)	1	3-6(t-2/3)
C_2	5-3(t-2/3)	5+9(t-2/3)	1	1
C_3	8	2	1	1
C_4	8	8	1	1

6.3.2 Triangular Gaussian Drift

A second model using a Gaussian distribution introduces the presence of cyclical drift. The two dimensional representation in Figure 6.4 illustrates a single rotation of three classes which are drifting in a triangular fashion according to the parametric table in Table 6.2. The experiment consists of two rotations at a constant drift rate, and the environment drifts between periods of high and low class separability. The duration of the experiment is 200 time steps, and data arrives in batches of size 20.

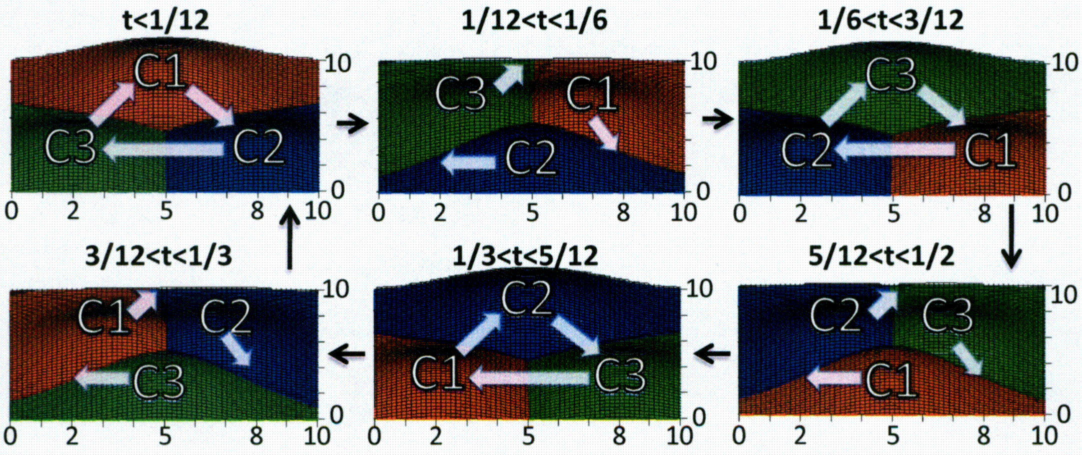


Figure 6.4: Graphical representation of 3-class triangular drift (single rotation)

Table 6.2: Parametric equations for triangular Gaussian drift data

	0<t<1/6 , 1/2<t<2/3				1/6<t<2/6 , 2/3<t<5/6			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_1	5+18t	8-36t	2	2	8-36t	2	2	2
C_2	2+18t	2+36t	2	2	5+18t	8-36t	2	2
C_3	8-36t	2	2	2	2+18t	2+36t	2	2

	2/6<t<1/2 , 5/6<t<1			
	μ_x	μ_y	σ_x	σ_y
C_1	2+18t	2+36t	2	2
C_2	8-36t	2	2	2
C_3	5+18t	8-36t	2	2

6.3.3 Random Gaussian Drift with Class Addition & Removal

This experiment contains four classes governed by a changing Gaussian distribution (mean & variance). In addition to shifting joint probabilities for each class, we also introduce severe change in class balance, where classes are permanently added or subtracted. Figure 6.5 graphically represents constant shift in distribution, where each class is governed by the respective parametric equations in Table 6.3. This experiment uses the same training/testing scenario as the other Gaussian data sets, and lasts 300 time steps.

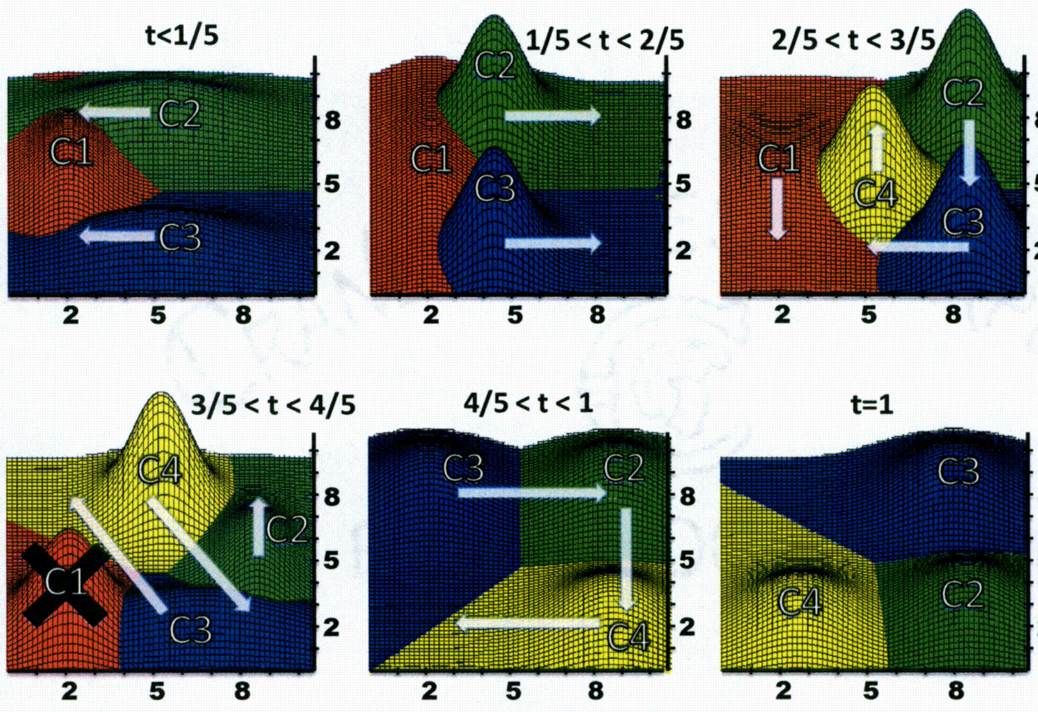


Figure 6.5: Graphical representation of 4-class Gaussian drift with class addition/removal

Table 6.3: Parametric equations for 4-class Gaussian drift with class addition/removal

	$0 < t < 1/5$				$1/5 < t < 2/5$			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C1	2	5	1	2+5t	2	5	1+5t	3-5t
C2	5-5t	8	3-10t	1	4+20t	8	1	1
C3	5-5t	2	3-10t	1	4+20t	2	1	1
C4	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	$2/5 < t < 3/5$				$3/5 < t < 4/5$			
C1	2	5-15t	2-5t	2-5t	N/A	N/A	N/A	N/A
C2	8	8-20t	1	1+5t	8	4+20t	1+2.5t	2-2.5t
C3	8-10t	2	1+10t	1	6-20t	2+30t	3-7.5t	1+2.5t
C4	5	5+15t	1	1	5+15t	8-30t	1+2.5t	1+2.5t

	$4/5 < t < 1$			
	μ_x	μ_y	σ_x	σ_y
C1	N/A	N/A	N/A	N/A
C2	8	8-30t	1.5	1.5
C3	2+30t	2	1.5	1.5
C4	8-30t	2	1.5	1.5

6.3.4 NON-GAUSSIAN DRIFT (CHECKERBOARD DATASET)

A non-Gaussian data set is derived from the canonical XOR problem, which resembles a rotating checkerboard. As shown in Figure 6.6, the rotation makes this deceptively simple-looking problem particularly challenging, as the angle and location of the decision boundaries change rather drastically at each time step. The images show half of an entire rotation ($\alpha = 0$ to π), indexed to the parameter α, where the axis of rotation is the lower left corner of the sampling window. Note that after half a rotation, data are drawn from a recurring environment, as the $[\pi\ 2\pi]$ interval will create an identical distribution drift to that of the $[0\ \pi]$ interval. In order to prevent training on identical snapshots of data and increase complexity, random noise (10%) is introduced. Each training dataset is kept particularly small, consisting of a mere 25 samples (total from both classes) drawn from the sampling window. Providing the learner with minimal training data is an additional challenge in this and other datasets.

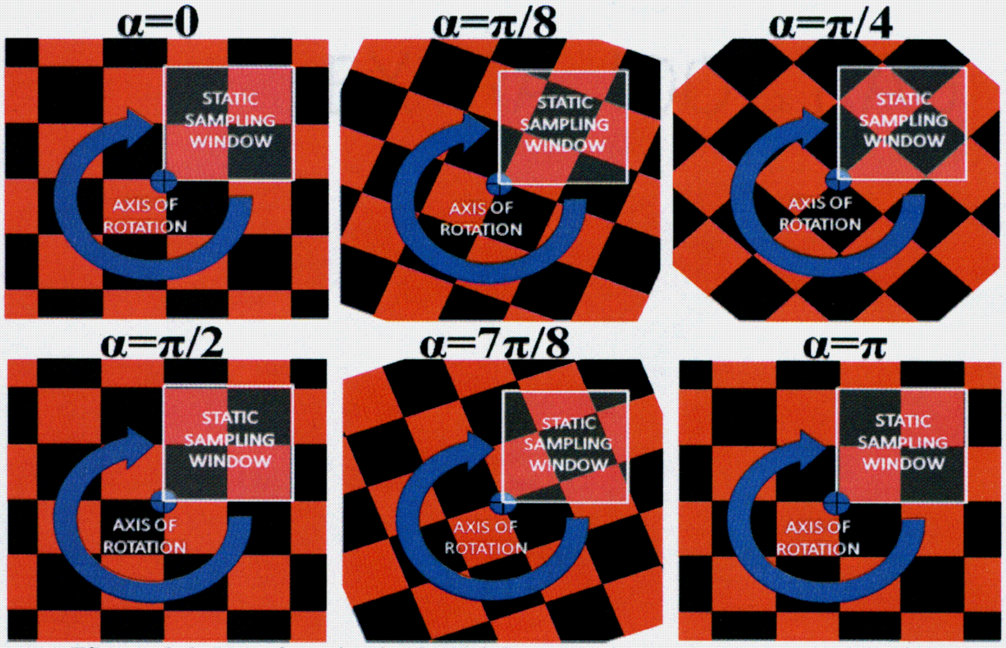


Figure 6.6: Rotating checkerboard dataset (single rotation, $\alpha = 0\ to\ \pi$)

All testing data is composed of 1,024 data points which are presented to the learner at each time step. This resolution (32 by 32) is sufficient to evaluate the learner's ability to approximate the sharp angles of the true decision boundary.

We introduce four variations of this dataset to observe the learner's resilience in the presence of harsher environments with varying drift rate, that is, where the rate of change in the distribution is not constant. This is accomplished by applying positive or negative acceleration to the α parameter as it increases from 0 to 2π. A constant drift rate is tested along with an exponentially increasing, pulsing, or sinusoidally fluctuating drift rate. These varying drift rates are depicted in Figure 6.7.

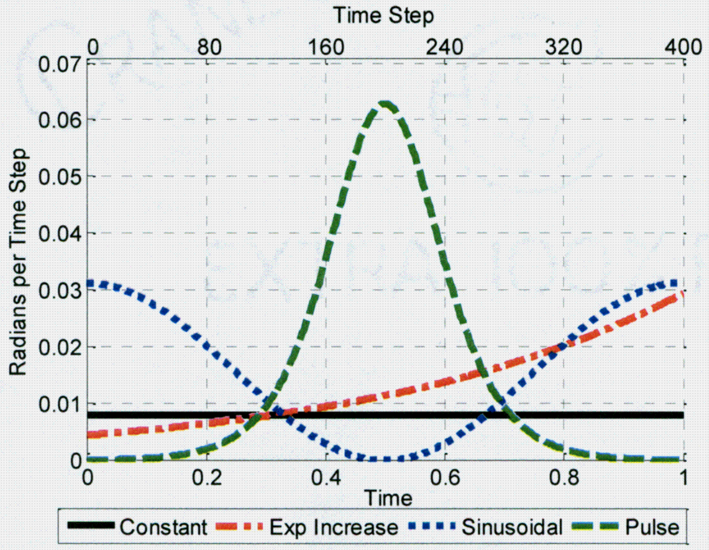


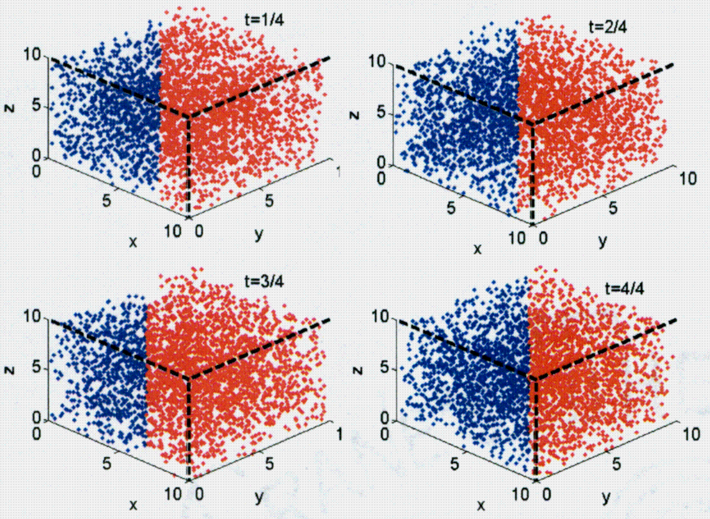
Figure 6.7: Variable α parameter for rotating checkerboard dataset

6.3.5 Concept Change (Streaming Ensemble Algorithm Dataset)

The SEA Concepts are a benchmark dataset developed by Street in [73], which has been adopted by many algorithms as a standard test for concept change. The dataset is characterized by an environment which undergoes extended periods without drift as well as occasional sharp changes in the class boundary (*concept change*) rather than concept *drift*. The environment consists of two classes of three features, although only two of the features are considered relevant, and the third feature is simply noise. Class labels are assigned based on the sum of the relevant features of the data, and are differentiated by comparing this sum to a threshold which effectively separates classes by a two-dimensional hyper-plane. For any data instance *n*, the sum of features $(f_1 + f_2)$ which fall below the threshold θ are assigned to class 1, and the remaining data points belong to class 2.

$$\omega_i = \begin{cases} 1 \; if \; f_{n,1} + f_{n,2} \leq \theta \\ 2 \;\; otherwise \end{cases} \tag{6.2}$$

At regular intervals of time, the threshold is changed, creating an abrupt shift in the class boundary. Data is uniformly distributed between 0 and 10, and the threshold θ_t is changed three times throughout the experiment with increasing severity.



$$\theta_t = \begin{cases} 8, & 0 < t \leq 0.25 \\ 9, & 0.25 < t \leq 0.5 \\ 7, & 0.5 < t \leq 0.75 \\ 9.5, & 0.75 < t \leq 1 \end{cases} \tag{6.3}$$

Figure 6.8: SEA Concepts (labeled testing data)

Figure 6.8 depicts actual snapshots of all the testing data from each environment. Training procedures are also followed from [73], where a total of 50,000 total points are introduced as training data (25,000 points per class), and 250 points are introduced at each time step, corresponding to a total of 200 time steps in the experiment from *t*=0 to *t*=1. In addition, 10% noise is added to training data as in [73]. A separate set of 50,000 total data points (no noise) are used for testing.

6.4 REAL-WORLD DATA

Real-world data, although ambiguous with respect to the presence concept drift, is a vital standard for ensemble comparison. Performance with large-scale drifting data will not only show whether or not an algorithm is consistent with a cognitive model, but will also reveal an algorithm's practicality in real data environments.

Here, we introduce a series of datasets from a variety of sources, measuring natural phenomena and large scale data trends. To some degree, it can only be assumed that concept drift is occurring; however, we can also obtain some more certain knowledge concerning class drift by using statistical analysis in an attempt to monitor trends in the data.

For observing changes in either the class-conditional probability $P(x|\omega)$ or prior probability $P(x)$, the Hellinger distance can be useful to compute the difference in distribution between consecutive batches or windows of data in order to ascertain whether or not the distributions are in fact changing. Note that this comparison requires no assumption of the distribution of the data. The two distributions under comparison are represented by data windows G and Q of a pre-determined size. Data is discretized into B evenly spaced bins in a histogram-like fashion to approximate a distribution, forming an approximation of the distributions, $\hat{G}$ and $\hat{Q}$ for the data windows G and Q. As in equation (6.4) for computing Hellinger distance, each bin (denoted by j) is normalized across all instances, and the summed square root of the distance between normalized bins is averaged across all features F (denoted by i) for the final measurement.

$$H^2(G,Q) = \frac{1}{F}\sum_{i=1}^{F}\sqrt{\sum\nolimits_{j=1}^{B}\left(\frac{\sqrt{G_{i,j}}}{\sum_j G_i} - \frac{\sqrt{Q_{i,j}}}{\sum_j Q_i}\right)^2} \tag{6.4}$$

The advantage of this method over density estimators is that it simplifies the observation by considering all features. Yet, simplification comes at a potential cost of over-generalization. Appendix A provides a characterization of Hellinger distance for the synthetic data sets in order to show that it is a viable tool for measuring relative changes in an environment. Each example figure also provides a plot of the measured drift rate, which is simply the derivative of Hellinger Distanced over time. Examples such as the checkerboard datasets show the relative changes in Hellinger distance, as well as changes in the drift rate among the four drift scenarios (constant, pulsing, exponential, and sinusoidal). Yet, the implementation of Hellinger distance supports the reasoning in Section 3.4.2 (p.42) that data distribution tracking is unreliable for drift detection. In order to see meaningful results, the amount of data per window was increased significantly for the synthetic datasets; otherwise, the output for Hellinger distance would appear mostly as noise. Thus large amounts of data are required to properly approximate and compare distributions.

We also provide a description of class balance for each real-world dataset. For monitoring class balance, we use a simple approach which counts the number of class occurrences in consecutive windows of data, and calculates the percentage of each class within that window. Two window sizes are used in the plots used; larger window size gives a more general description, and a smaller window size is an example of the actual size used in a given experiment.

6.4.1 New South Wales Electricity Dataset

The electricity pricing domain is a sequence of data relating various aspects of time and power transfer to fluctuations in the price of electricity in New South Wales, Australia. This dataset is becoming one of many benchmark non-stationary problems in the field [29;80]. The data consists of features acquired twice per hour, and covers a span from

May 7, 1996 to December 5, 1998, where class labels correspond to either an *increase* or *decrease* in the price of electricity. A total of six features are used to represent the time (time of day, and day of week), electricity demand (New South Wales and neighboring Victorian state), and the scheduled transfer of electricity between the two states. Classes are balanced relatively evenly (58% "up", 42% "down") among the 27,549 instances, and there is a consistent presence of both classes over the course of time, as seen in the graphical representation (Figure 6.9) of class balance over time, where percentages are based on the population of the "up" class within consecutive windows of the data.

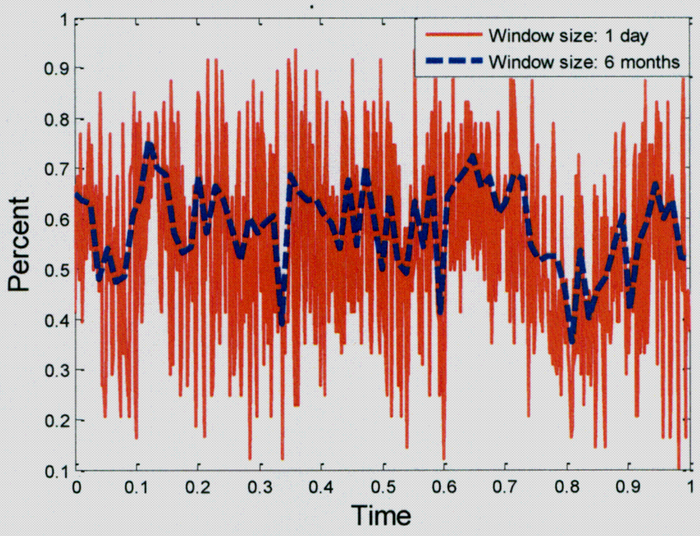


Figure 6.9: Percent class ("up") instances per window

Two plots are provided for Hellinger distance. The class-dependent Hellinger distance (Figure 6.10) estimates change and drift rate of each class over time, whereas class-dependent Hellinger distance (Figure 6.11) represents general changes in the data, regardless of class. The plots indicate some gradual drift in the class-dependent distribution as well as some more severe levels of drift in the class-independent distribution ($t = 0.2, t = 0.35, t = 0.6$).

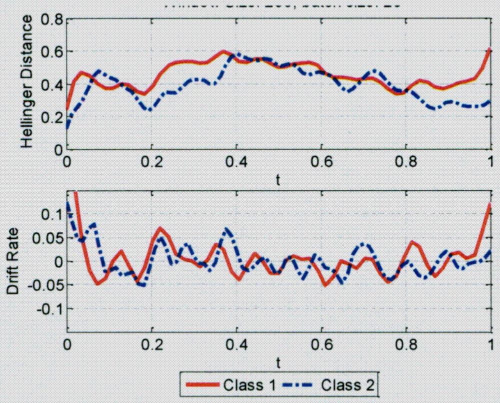


Figure 6.10: Class-dependent Hellinger distance and drift rate with window size of 200 for electricity pricing data

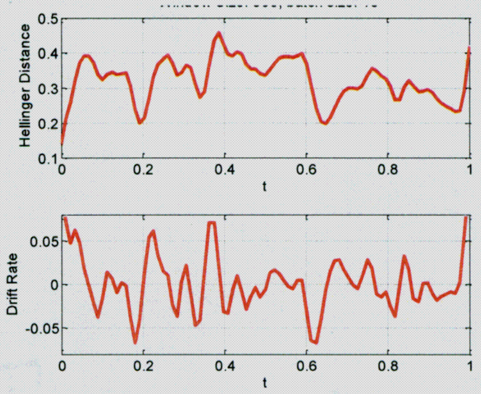


Figure 6.11: Class-independent Hellinger distance and drift rate with window size of 200 for electricity pricing data

Table 6.4: Learning scenarios for electricity pricing data

Scenario	**Classification**		**Prediction**	
Experiment	**A**	**B**	**D**	**E**
Training (days)	24	48	15	24
Testing (days)	24	48	15	24

Two learning scenarios (*prediction* and *classification*) are based on a small window size spanning either a half or a whole day's worth of readings (24 and 48, respectively) as shown in Table 6.4. Recall that in classification, both training and testing data are drawn randomly from a single batch, and in prediction, testing data comes from a batch *after* the training batch.

6.4.2 NEBRASKA WEATHER DATASET

The National Oceanic and Atmospheric Administration (NOAA), part of the United States Department of Commerce (USDC), has compiled a database of weather measurements from over 7,000 weather stations worldwide. Records date back to the mid-1900's providing a wide scope of weather trends. Daily measurements (Table 6.5) include a variety of features (temperature, pressure, wind speed, etc.) as well as a series of indicators for precipitation and other weather-related events. The Offutt Air Force Base in Bellevue, Nebraska was selected for experimentation based on its extensive range of over 50 years (1949-1999) as well as its full feature set. Also, the geographic location is known to undergo diverse weather patterns, making it a viable classification or prediction problem where the measurements represent the features for a given weather-related event (rain, fog, snow, etc.).

Table 6.5: Weather data measurements and event indicators

DAILY MEASUREMENTS	
Temperature	**Dew Point**
Sea Level Pressure	**Station Pressure***
Visibility	**Average Wind Speed**
Max. Sustained Wind Speed	**Maximum Wind Gust***
Maximum Temperature	**Minimum Temperature**
INDICATORS ("yes" or "no")	
Fog	**Rain**
Snow	**Hail**
Thunder	**Tornado**

*removed because number of missing features>20%

Eight features are select from the list above, based on their availability; a missing feature rate above 15% was deemed insufficient for use. For the selected feature set, missing values were synthetically generated beforehand using an average of the instances before and after the missing one. Class labels are determined based on the binary indicator(s) provided for each daily reading. Using rain as the class label yields the most balanced dataset consisting of 18,159 daily readings, 5,698 (31%) of which are positive ("rain") while the remaining 12,461 (69%) are negative ("no rain") as depicted in Figure 6.12.

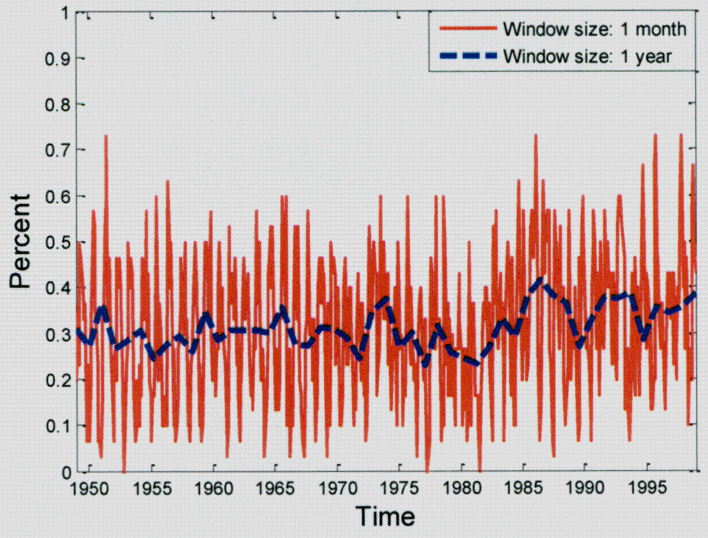


Figure 6.12: Seasonal observation of class balance (120 days per window)

With respect to drift, the Hellinger distance particularly indicates periodic drift and drift rate in both the class-conditional (Figure 6.13) and class-independent (Figure 6.14) measurements, which would be expected on a year-round basis as weather patterns change in a cyclical fashion. The presence of periodic drift is also supported in the empirical results presented later, revealing a clear cyclical drift in the class descriptions over the course of each year.

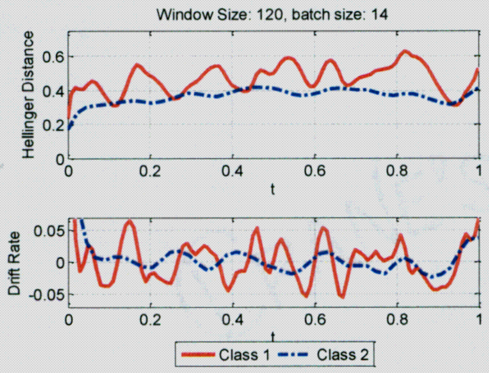


Figure 6.13: Class-dependent Hellinger distance and drift rate with window size of 120 for weather data

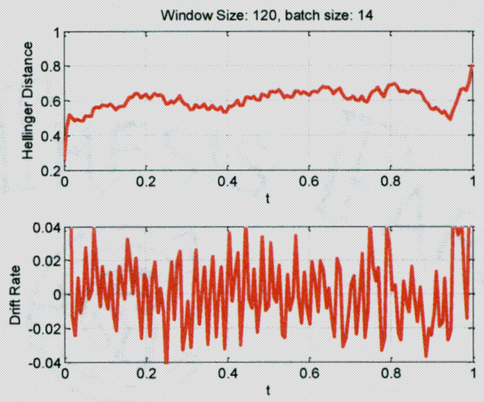


Figure 6.14: Class-independent Hellinger distance and drift rate with window size of 120 for weather data

Both classification and prediction are simulated as learning scenarios using this dataset. Training/testing combinations are grouped by weeks, months, or entire seasons (Table 6.6). These variations will indicate whether it is best to learn an environment (season) altogether or in smaller consecutive segments. The particular risk of small a small batch size is class imbalance.

Table 6.6: Learning scenarios for weather data

Scenario	**Classification**		**Prediction**	
Experiment	**A**	**B**	**D**	**E**
Training (days)	15	30	14	30
Testing (days)	15	30	14	30

6.4.3 Spam Dataset

The Spam Data set is a text mining problem introduced in [75]. Each instance represents an email transaction, described by a 499-bit binary stream, where each bit represents whether or not a particular word occurs in the text. The dataset includes a total of 9,500 messages, each of which are labeled as "spam" or "legitimate." The balance of classes favors "legitimate" emails (73%) over spam emails (27%) as shown in Figure 6.15, while also showing periods of severe imbalance ($t = 0.5, t = 0.3, t = 0.9$) where only one class is available in the data. Two learning scenarios are used (classification and prediction), each consisting of a window size of 20. Because the features are binary representations of words, the Hellinger distance does not yield a useful description of this environment.

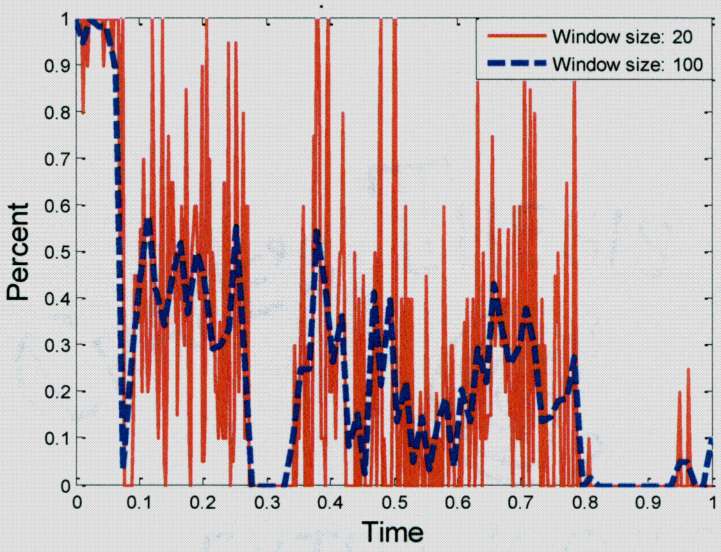


Figure 6.15: Percent of positive ("spam") class instances per window

6.5 General Performance and Base Classifier Analysis

A key advantage of the Learn^{++}.NSE algorithm is its functionality across all types of generative and descriptive base classifier models. In our experiments, the *Naïve Bayes* (NB) classifier is use as a representative generative model, characterized by its assumption of a normal Gaussian distribution with class-conditionally independent features. Naïve Bayes associates each class as a Gaussian distribution (mean and variance) to represent class-conditional likelihood $P(x|\omega)$ for data x in class ω. Classification on unlabelled testing data is accomplished by choosing the class with the highest probability according to Bayes rule. The Naïve Bayes classifier is expected to perform rather well in cases where the data is indeed of a normal distribution, and to suffer to some degree when this is not the case. The *Multi-Layer Perceptron* (MLP) neural network is a descriptive model which maps the appropriate output classes to input training examples using a network of nodes with weighted connections. Connection weights are adjusted in the training process using a back-propagation algorithm in order to provide an appropriate fit to the training data. Because the algorithm is designed to *approach* optimality, it may yield unique (though similar) solutions when trained on identical datasets. The greatest disadvantage of the MLP networks in comparison with other methods in this study is time required for training and testing, especially as data features increase. The *Support Vector Machine* (SVM) is an optimization approach to finding a margin between classes that maximizes distance while also accounting for noise and overlap in the data. A key advantage of the SVM is that it will yield an optimal result; additionally, the SVM boasts a significant improvement in training and evaluation time over the MLP.

Before discussing the comparison of base classifiers, let first us discuss some notable characteristics of the learner on each dataset. These characteristics are displayed in two ways: a comparative performance plot over time, and an ANOVA comparison plot for averaged performance. Each individual performance curve is color-coded and enclosed with similarly colored background shading, denoting the 95% confidence interval over at least 50 independent trials. The axis includes both the normalized time $0 \leq t \leq 1$ as well as the number of time steps. The legend includes the name of the corresponding method or algorithm, averaged performance over time, and the 95% confidence interval across all trials. Note that some performance plots are smoothed using an average of previous performance points, allowing for better visual comparison. Smoothing also accounts for the initial performance of 0% in some plots in which there is an insufficient number of points available to compute the average.

The first performance plot shown is the random Gaussian drift dataset (Figure 6.16); this dataset is a relatively separable problem that undergoes steady gradual drift, and is therefore learned rather easily by the ensemble.

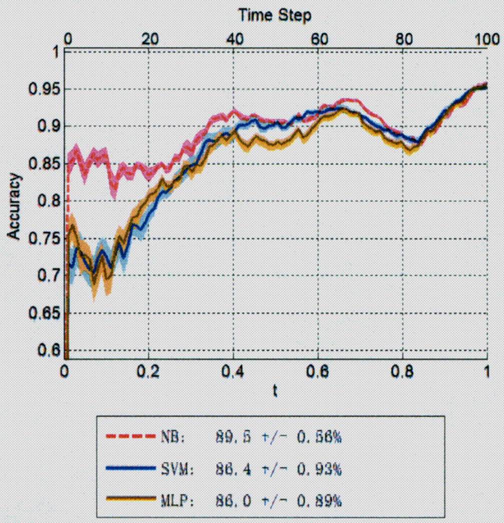


Figure 6.16: Base classifier comparison for random Gaussian drift data

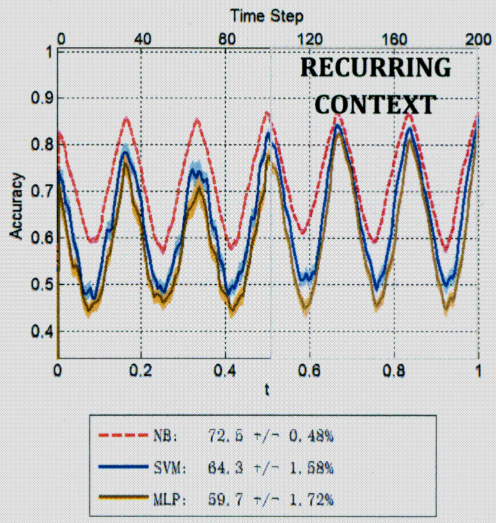


Figure 6.17: Base classifier comparison for triangular Gaussian drift data

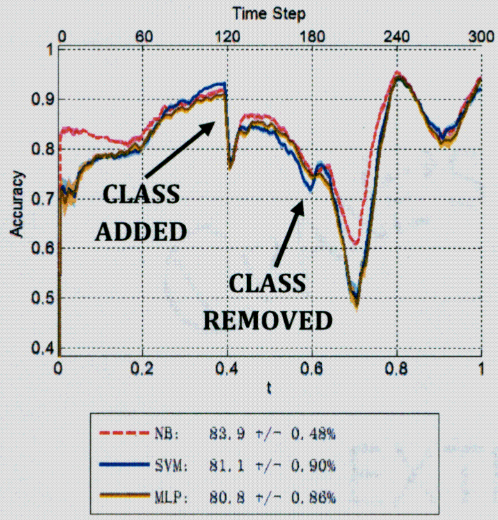


Figure 6.18: Base classifier comparison for random Gaussian drift data with class addition/removal

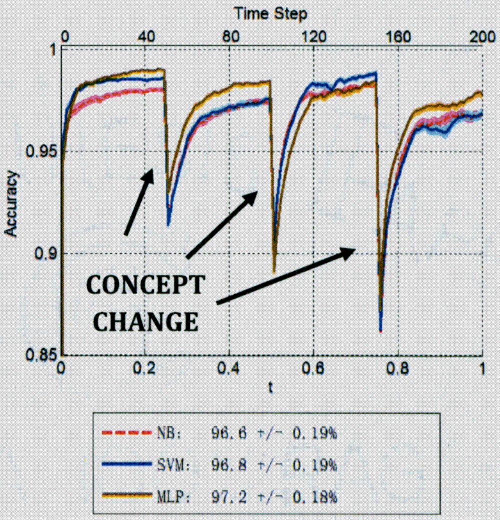


Figure 6.19: Base classifier comparison for SEA concept data

The triangular Gaussian drift dataset (Figure 6.17) is slightly more difficult because of the significant amount of class overlap as classes drift from one point of the triangle to another. This change in the separability of the data accounts for the periodicity of the performance curves.

In the class addition/removal dataset (Figure 6.18), we make three observations: (1) the drop in performance following the addition of a fourth class ($t = 0.4$), (2) the increase in performance when class 1 is removed ($t = 0.6$), and (3) the significant changes in performance from $t = 0.6$ to $t = 1$ as a result of low and high class separability.

The notable characteristics of the SEA dataset (Figure 6.19) are the convergence of performance during periods of no drift, and the ability of the learner to react to concept change that is introduced at quarterly intervals between $t = 0$ and $t = 1$. This experiment is a good test for both the stability and plasticity of an algorithm.

The prominent attributes of performance curves on the checkerboard data (Figure 6.20-Figure 6.23) are the four periods during the experiment at which the data appears in perfect right angles (α in multiples of $\pi/2$); this is the most easily learned data environment and is manifested in the spikes in performance (especially the MLP and SVM). Performance increases over the latter half ($\alpha = \pi\ to\ 2\pi$) are indicative of the learner's ability to recall the environments which were learned in the former half ($\alpha = 0\ to\ \pi$).

Performance curves on the weather prediction data (Figure 6.24) appear to also indicate some level of annual periodicity, relative to the cyclical seasonal change in the data.

The electricity data (Figure 6.25) appears to undergo little or no cyclical change, by nature of the randomness of performance.

Finally, with regard to the spam data (Figure 6.26), we note the periods from $t = 0$ to $t = 0.1$, $t = 0.3$ to $t = 0.4$ and $t = 0.8$ to $t = 1$ yield high performance results because only one class is present.

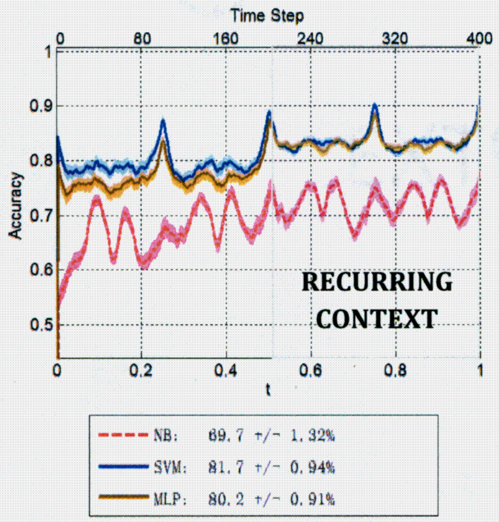


Figure 6.20: Base classifier comparison for checkerboard data (constant drift)

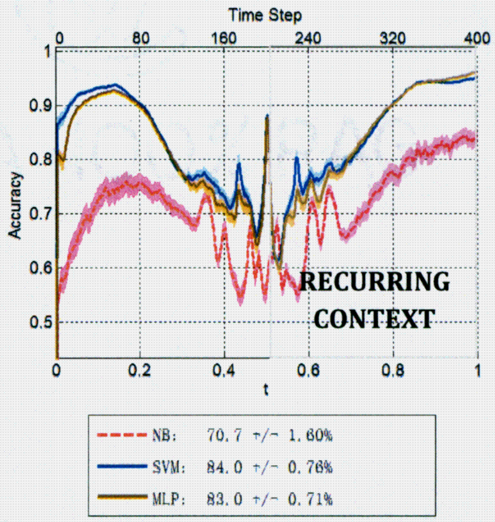


Figure 6.21: Base classifier comparison for checkerboard data (pulsing drift)

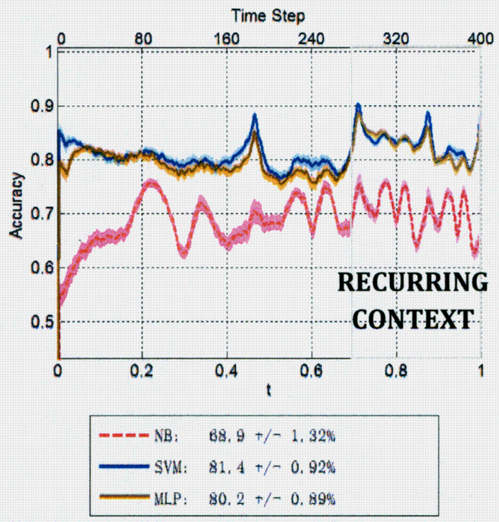


Figure 6.22: Base classifier comparison for checkerboard data (exponential drift)

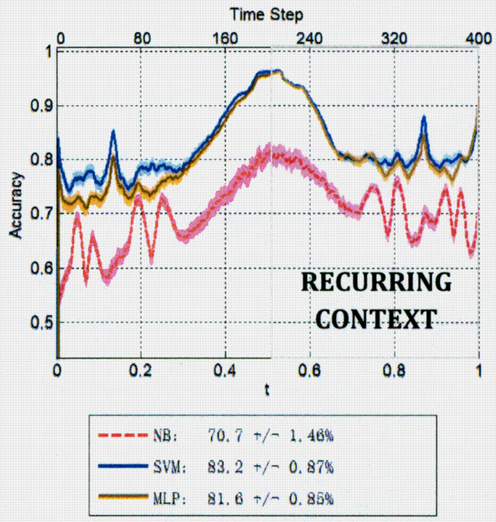


Figure 6.23: Base classifier comparison for checkerboard data (sinusoidal drift)

The comparison of average performance using ANOVA is provided at the conclusion of this section in Figure 6.27 for all results that were previously discussed, as well as some additional learning scenarios ("pred" for *prediction,* and "clas" for *classification*) for real-world datasets. This plot serves as an additional support to the general observations to be made in the plots of performance averages over time. Each ANOVA plot displays the average performance of a particular method, accompanied by a confidence interval. Any overlap among intervals denotes a lack of statistical significance between any two methods over the entire experiment.

For the Naïve Bayes classifier, results in the synthetic experiments support the reasoning that the classifier will learn and accurately predict data from a Gaussian distribution (Figure 6.16-Figure 6.17), even surpassing generative models in some case. However, performance is significantly diminished when data is non-Gaussian, as seen in the checkerboard dataset (Figure 6.20-Figure 6.23) and spam dataset (Figure 6.26). Poor performance on the spam dataset may also be related to the binary nature of the input data, although this is mostly speculation.

The MLP and SVM architectures, consisting of a small number of free parameters, can be set to accommodate the provided data sets with little or no tuning across experiments. MLP neural networks are trained with an error goal of 0.01 and 25 hidden layer nodes. The SVM classifier is structured with a polynomial kernel of order 6 for synthetic data and a more generalized order of 2 for real-world data. The purpose of this work is *not* the optimization of these parameters; rather, it is the comparison of methods and algorithms under identical parameters – it is quite possible that the provided MLP and SVM parameters can be tuned for an even better performance in a given experiment. The most important consideration in parameter selection is the prevention of overfitting to ensure that the classifiers are learning a general distribution, and not "memorizing" snapshots of training data.

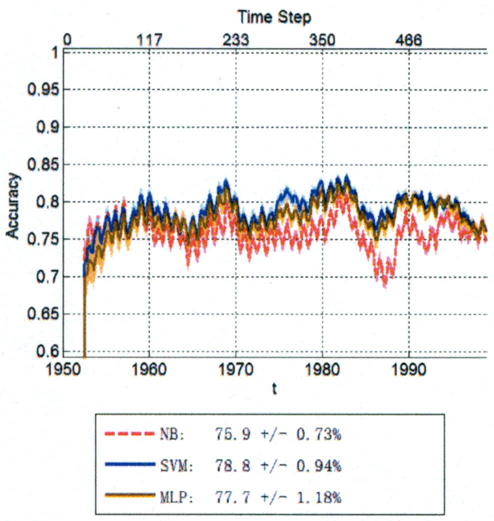


Figure 6.24: Base classifier comparison for weather prediction data, m=30

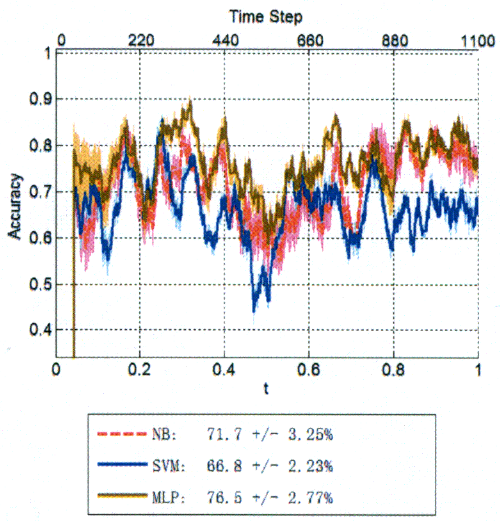


Figure 6.25: Base classifier comparison for electricity pricing prediction data, m=24

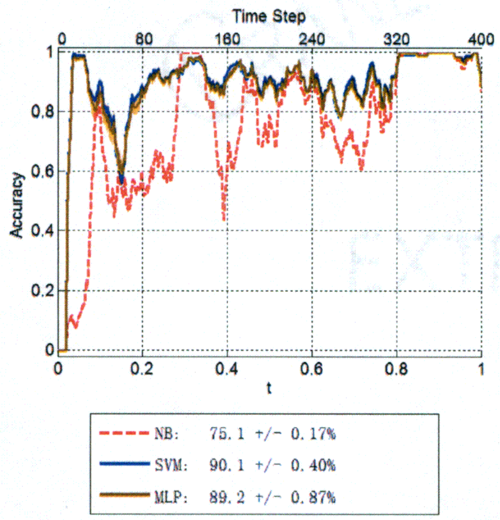


Figure 6.26: Base classifier comparison for spam prediction data, m=20

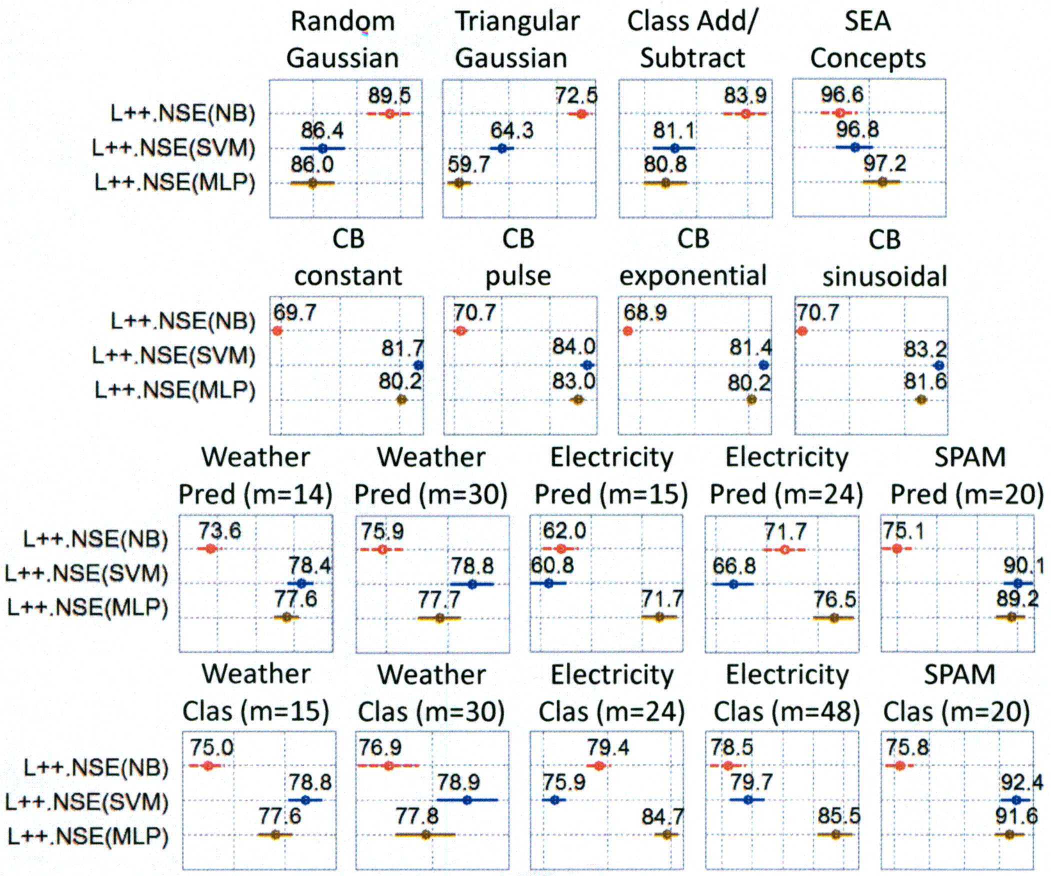


Figure 6.27: ANOVA comparison for base classifier analysis

In general, all memory characteristics are consistent across each base classifier models, including the following:

Data-related performance peaks – some environments are simpler to learn than others. One example can be seen in the triangular drift data (Figure 6.17), where performance increases during periods when distributions are most separate, lying on the three points of the triangle. Another example can be seen in the checkerboard dataset (Figure 6.20), where the environment at every multiple of $\alpha = \pi$ is simple to learn; this is least noticeable in the Naïve Bayes performance curve because of the classifier's difficulty in representing the complex, non-Gaussian dataset.

Memory recall is apparent in all experiments infused with recurring environments, such as the triangular Gaussian drift (Figure 6.17) and checkerboard datasets (Figure 6.20-Figure 6.23). Learn^{++}.NSE is clearly able to utilize prior knowledge in order to boost performance, regardless of the base classifier.

Reaction to different types of drift – specifically, this refers to the trends during various drift rates (degraded performance when drift increases, and vice versa) depicted the checkerboard data (Figure 6.20-Figure 6.23) as well as concept change (sharp drop in performance followed by recovery period) in the SEA Concepts (Figure 6.19) and class addition/removal (decrease in performance when classes are added, and vice versa) in Figure 6.18.

6.6 CLASSIFIER ENSEMBLE VS. SINGLE CLASSIFIER

A key outcome observed throughout all experiments is between the performance of the ensemble of base classifiers and that of the single most recent classifier. This observation reveals, first and foremost, whether or not it is even worthwhile to maintain old knowledge. We observe in nearly every experiment the overall superiority of the ensemble over a single classifier. Momentary dips in the Learn^{++}.NSE ensemble are most often explained by periods of sharp drift (SEA Concepts, checkerboard with pulsing drift rate, electricity,), during which prior knowledge serves little or *no* purpose and therefore has a negative (baggage) effect on the ensemble decision. These are but momentary lapses before the ensemble learner is able to catch up. The superiority of the Learn^{++}.NSE ensemble is *not* a trivial benchmark, for it proves that prior knowledge/classifiers do indeed carry significant information relevant to the current environment. If this were not the case, then the entire effort to learn and store classifiers would be in vain.

6.7 Ensemble Weighting Comparison

This section discusses computational intelligence approaches to knowledge base organization techniques using ensemble weighting. What makes the Learn^{++}.NSE weighting method unique compared to others is the use of the penalty distribution for computing error, as well as the sigmoidal, time-based error weighting for computing the classifier voting weight. Here we show three comparisons which display the specific effects of taking these measures. First we observe a comparison with an un-weighted ensemble, where all classifiers receive equal voting power. Second is the Adaboost weighting equation from which Learn^{++}.NSE weighting is derived. It is important to note that henceforth, we refer to Adaboost as an implementation of its weighting method, and *not* an implementation of the Adaboost algorithm. Adaboost weighting differs from Learn^{++}.NSE weighting in that (1) error is represented by the percent misclassification, and (2) weights are calculated based on a classifier's current error only. The final comparison is the weighting method proposed in [35] for the ACE classifier, which combines Adaboost-type weighting with a temporary pruning method.

Learn^{++}.NSE appears to be the most consistent performer throughout all experiments, boasting its ability to *best* organize/weight classifiers, especially when environments are recurring (triangular drift and especially the checkerboard experiments). Learn^{++}.NSE strikes a good balance between stability and plasticity with its superior ability (1) to react quickly to periods of severe drift (Figure 6.33-Figure 6.34), concept change in the SEA dataset (Figure 6.31), as well as class addition in the Gaussian drift experiment (Figure 6.30); and (2) its ability to converge when there is little or no drift, as in checkerboard (Figure 6.33, Figure 6.35) and SEA (Figure 6.31) datasets.

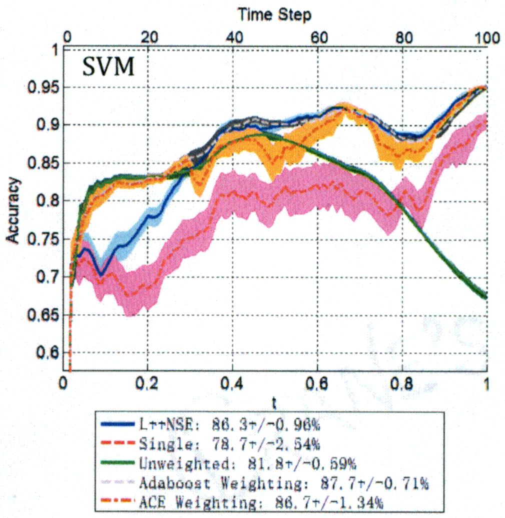


Figure 6.28: Weighting comparison for random Gaussian drift data

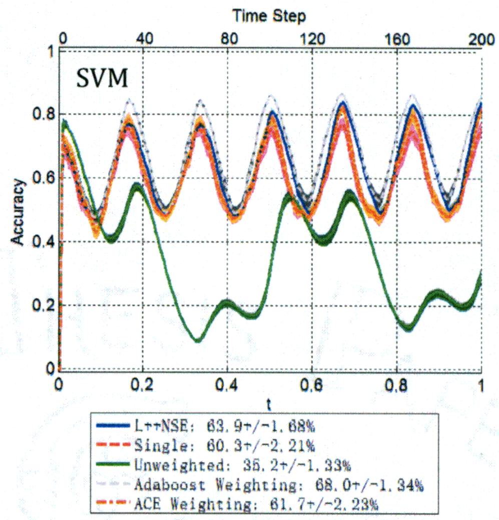


Figure 6.29: Weighting comparison for triangular Gaussian drift data

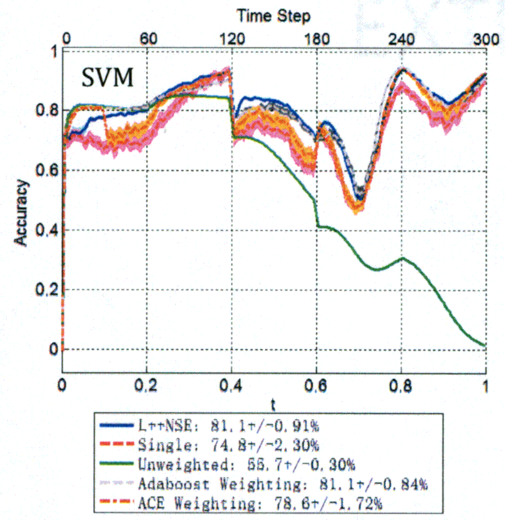


Figure 6.30: Weighting comparison for random Gaussian drift data with class addition/removal

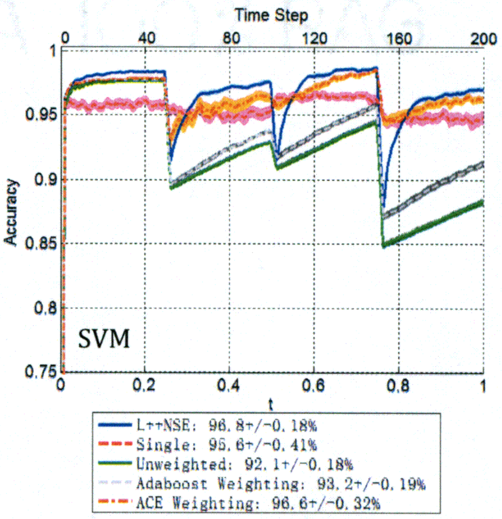


Figure 6.31: Weighting comparison for SEA concepts data

The performance of the unweighted ensemble throughout the synthetic experiments (Figure 6.28-Figure 6.35) clearly shows that it is not a viable scheme for evaluating the knowledge base. Performance in real-world scenarios is improved, yet exceeds that of a single classifier *only* in certain electricity and weather datasets (all

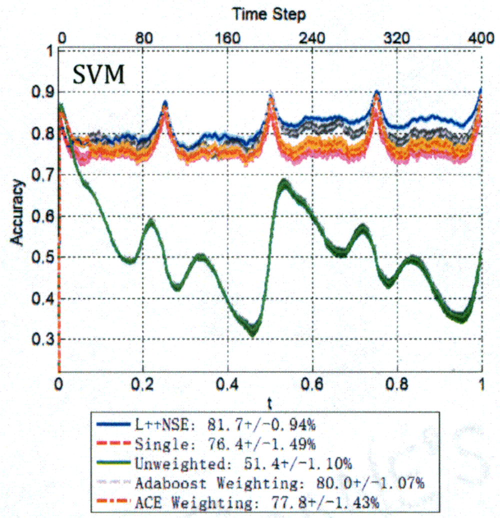


Figure 6.32: Weighting comparison for checkerboard data (constant drift)

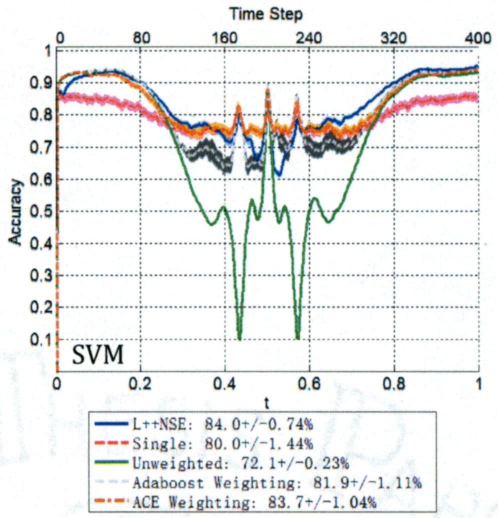


Figure 6.33: Weighting comparison for checkerboard data (pulsing drift)

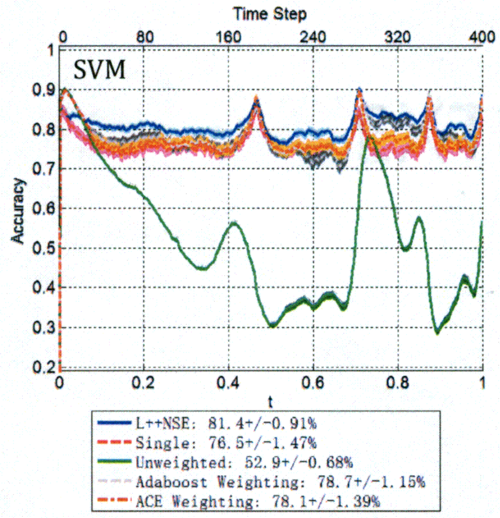


Figure 6.34: Weighting comparison for checkerboard data (exponential drift)

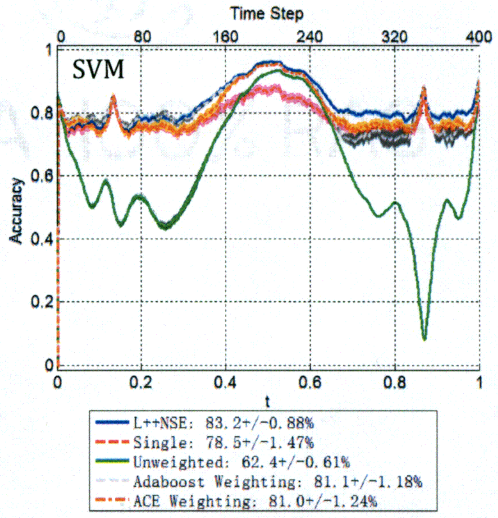


Figure 6.35: Weighting comparison for checkerboard data (sinusoidal drift)

shown in ANOVA plots in Figure 6.39), and not in the spam dataset (Figure 6.38). Good performance on the weather data can be partially explained by the fact that *most* knowledge is relevant at any given point in time because of the recurrence of environments on a yearly basis.

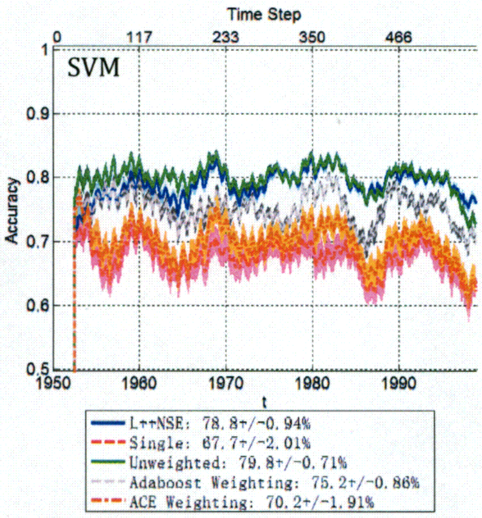


Figure 6.36: Weighting comparison for weather prediction data m=30

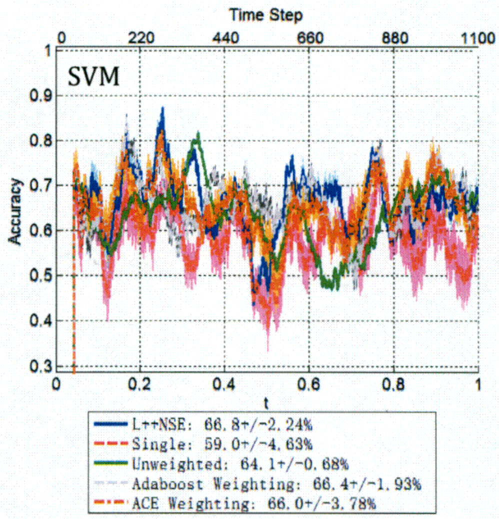


Figure 6.37: Weighting comparison for electricity pricing prediction data, m=24

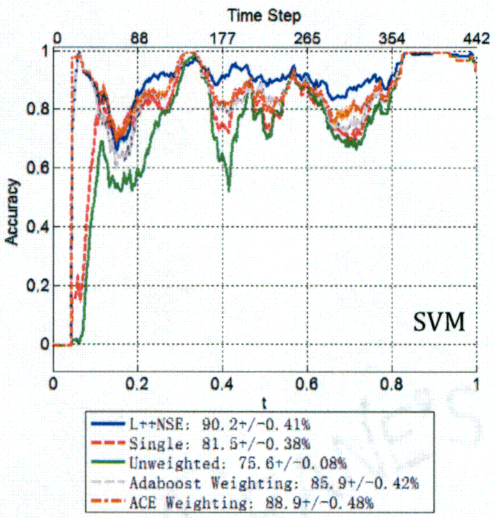


Figure 6.38: Weighting comparison for spam prediction data, m=20

In general, such a passive approach to knowledge categorization is simply not prudent, even for the prediction problems in the real-world experiments. This seriously conflicts with some key assertions made by Gao in [5] which favor an unweighted ensemble. They key problem with supporting an unweighted ensemble is the unwillingness to assume that data from consecutive batches of data (from training to testing) will come from similar environments. However, Gao's notion appears to ignore the equally important learning effect known as *baggage*. Without some level of classifier categorization, irrelevant classifiers are bound to degrade classification performance.

Adaboost weighting appears to be most useful in drifting environments where class distributions are slow-drifting and simple in nature (i.e. Gaussian datasets in Figure 6.28-Figure 6.30). However, datasets with added complexity (checkerboard data), drift rate, or concept drift (SEA data) reveal many inconsistencies in performance. At some point in each of these examples, performance drops at best to the same level as a single classifier and sometimes lower. Performance curves on checkerboard data with pulsing drift rate (Figure 6.32), checkerboard data with sinusoidal drift (Figure 6.35), and SEA concepts data (Figure 6.31) specifically indicate an inability to react quickly to severe drift cases. The possible cause for these shortfalls is discussed further on.

ACE attempts to mitigate the problem of *baggage* by only using the best-performing classifiers. Here, the underlying problem with ACE appears to be an undersampling of the knowledge base, where classifiers which may contain relevant information are ignored because they do not fall within the confidence interval for classifier selection. This method causes a tradeoff between baggage reduction and effective use of knowledge, which is closely related to the confidence interval used for selecting experts. Learn^{++}.NSE appears to better mitigate this problem using both the penalty distribution and time-based error weighting.

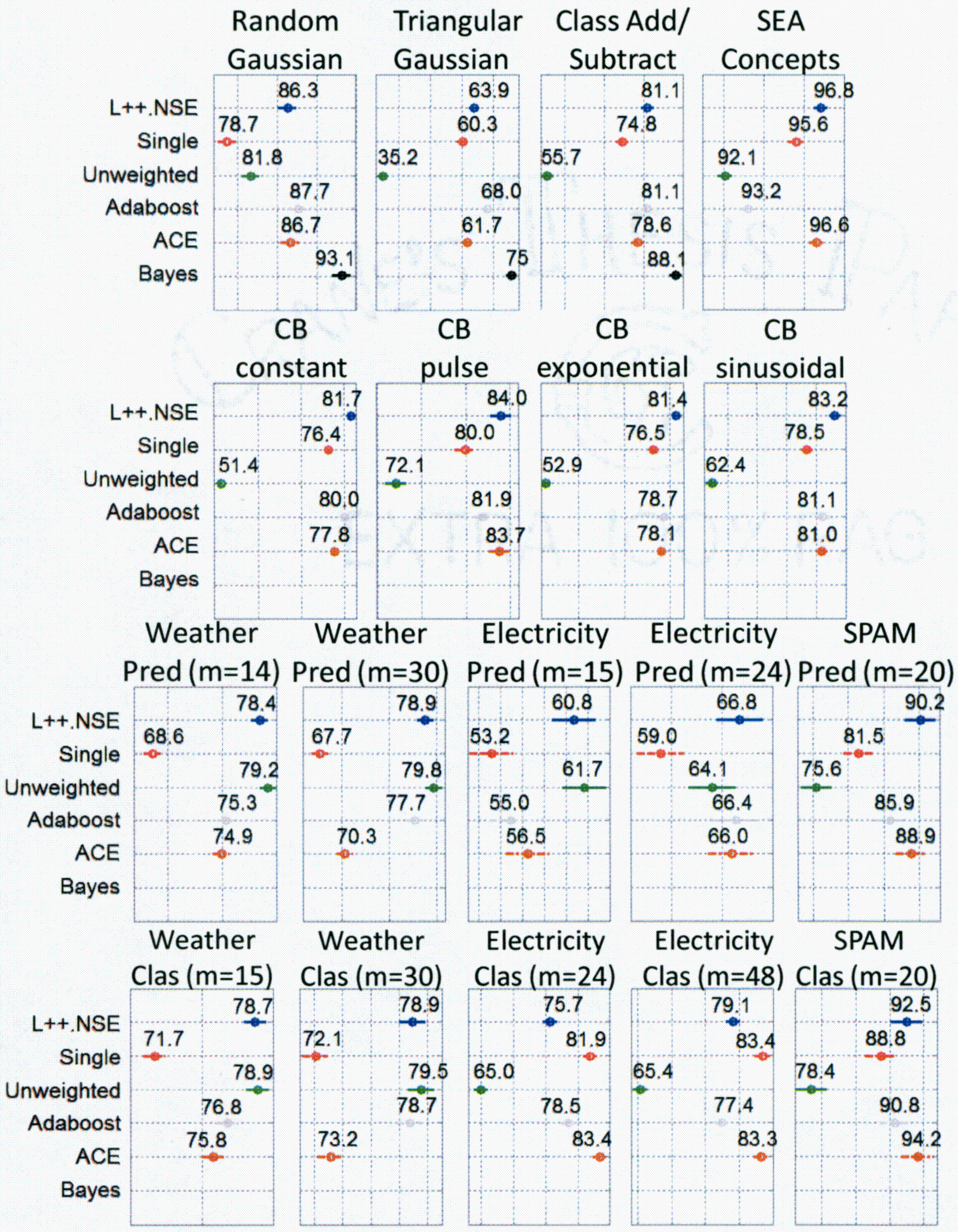


Figure 6.39: ANOVA comparison for ensemble weighting methods (SVM)

Although there is seldom a case where any weighting method significantly outperforms another, the Learn^{++}.NSE ensemble appears to be the most consistent performer throughout all experiments, and makes the most effective use of prior knowledge especially in the presence of recurring data (Figure 6.32-Figure 6.35). General trends can especially be seen in the ANOVA comparison in Figure 6.39. Other methods show moderate improvement during recurring environments, yet none are as significant as Learn^{++}.NSE.

The effectiveness of using prior data can be further illustrated using a 3-dimensional plot, showing the weight assignment for each classifier through the course of the entire experiment, averaged over 50 trials (Figure 6.40-Figure 6.46). Specifically, a comparison is made between the Learn^{++}.NSE and Adaboost weighting techniques (ACE weights are simply a pruned version of Adaboost weights). Log-based plotting is used to accentuate the differences in weight over time. In each plot, the diagonal from the upper left to the lower right is the weight granted to the most recent classifier. The shaded area below the diagonal represents weights of previous classifiers at a given time. Zero weight is indicated by a lack of color.

It is interesting to note the effectiveness of using prior knowledge; this is especially noticeable in experiments involving recurring environments which are shown. The triangular Gaussian drifting data, checkerboard data, and even the weather data all show periodic recall of prior knowledge created 100 time steps, 200 time steps, and 365 days (or 1 year) ago, respectively. The periodicity in the weather data is especially interesting and gives further evidence that the classes are indeed drifting in a periodic nature. Additional dataset comparisons can be seen in the Appendix B.

Overall, there is an outright difference in what could be called the presupposition about old classifiers' effectiveness. Learn^{++}.NSE gives significant preference to more recent classifiers and tentatively weights prior classifiers, whereas Adaboost seems to have a more polarized approach, giving either a very high or a very low weight to classifiers regardless of age. This difference can only be attributed to Learn^{++}.NSE's penalty distribution, which credits classifiers specifically for performing well on unknown examples. Such a weighting method tends strongly toward the most recent classifiers, and appears to be appropriate and clearly advantageous over the sensitivity of Adaboost. (NOTE: the term Adaboost refers to its *weighting* technique, not the Adaboost algorithm)

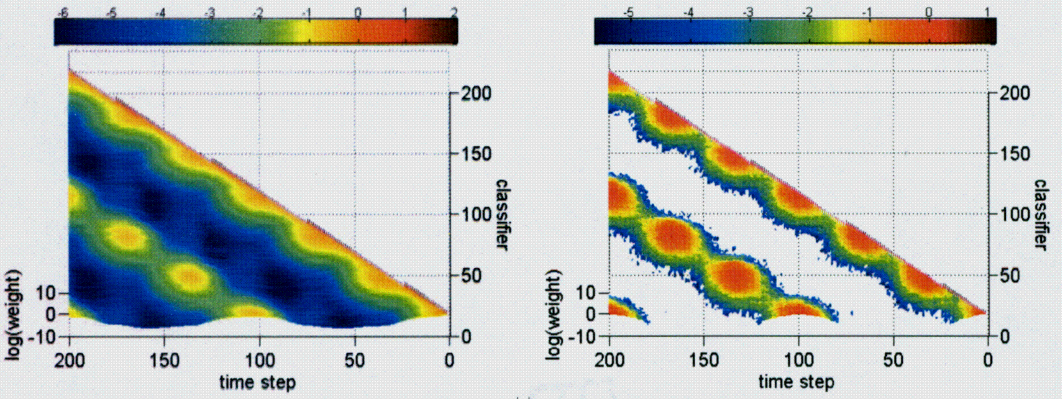


Figure 6.40: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for triangular Gaussian drift data

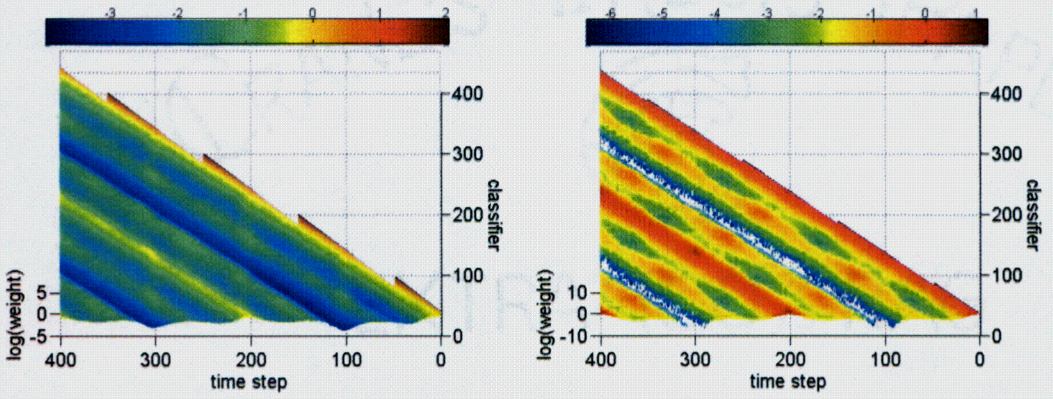


Figure 6.41: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (constant drift)

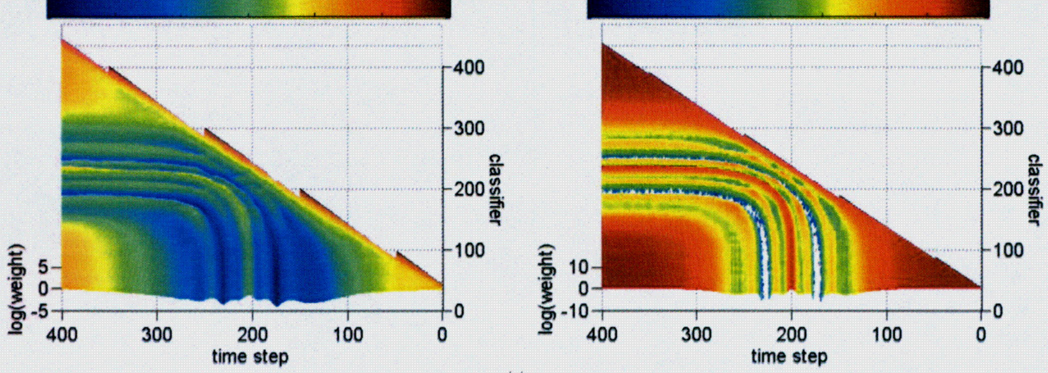


Figure 6.42: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (pulsing drift)

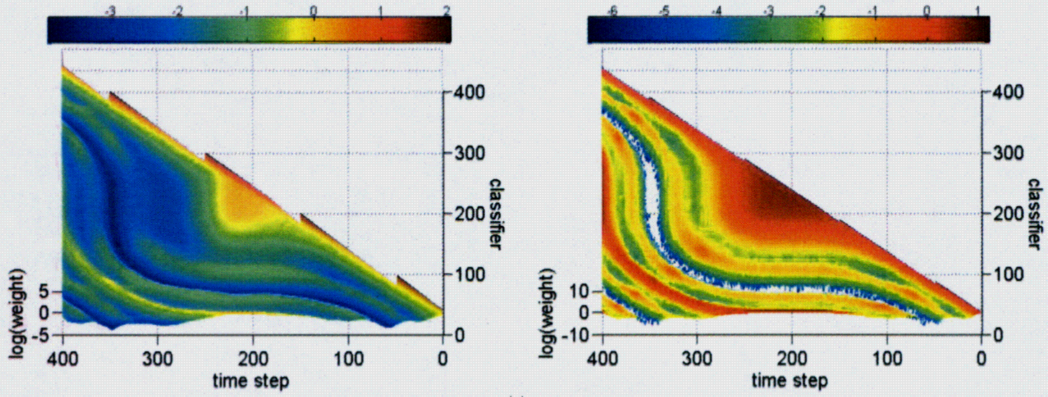


Figure 6.43: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (sinusoidal drift)

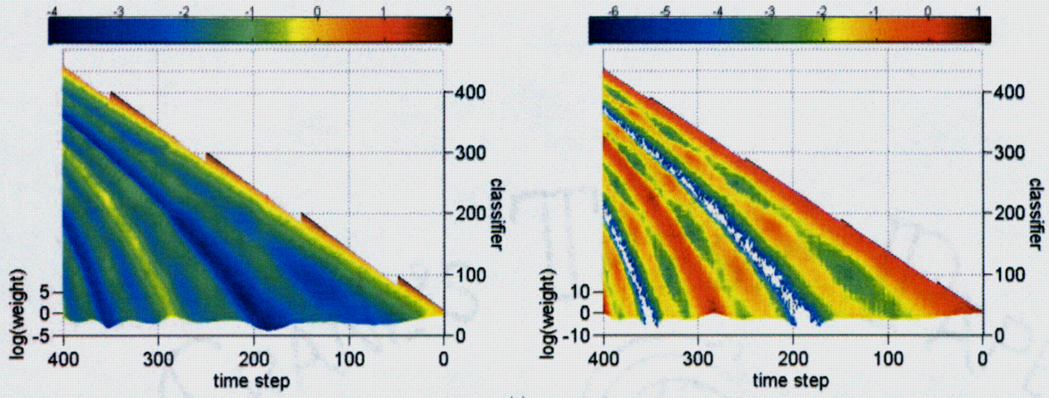


Figure 6.44: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for checkerboard data (exponential drift)

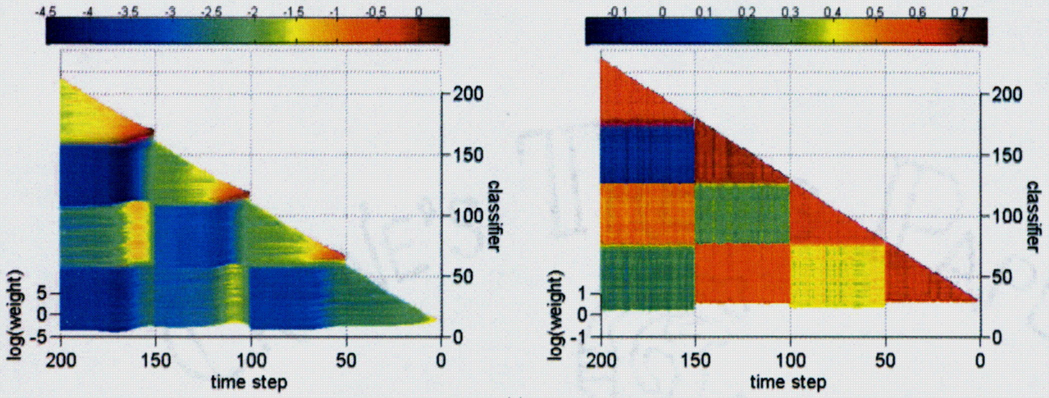


Figure 6.45: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for SEA concepts data

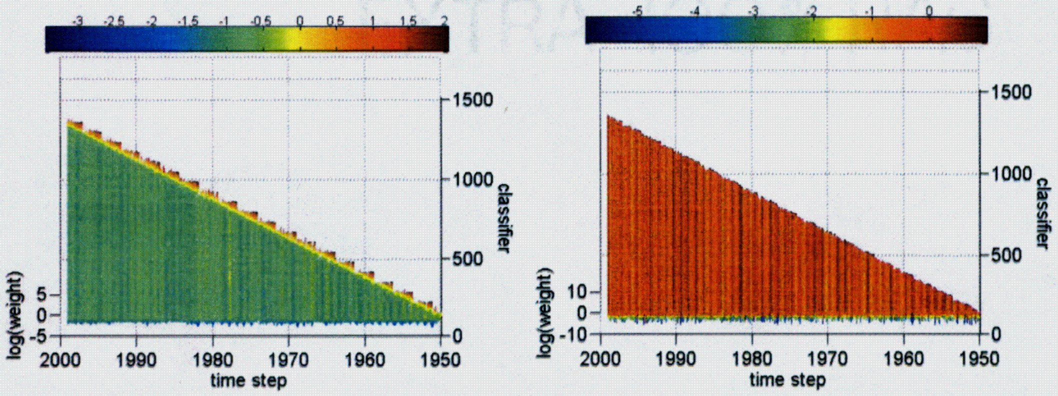


Figure 6.46: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for weather prediction data (training window: 2 weeks)

6.8 Online vs. Batch Learning

The Dynamic Weighted Majority (DWM) algorithm [29] is perhaps the foremost benchmark for *online* learning in non-stationary environments. It follows that an implementation of DWM would be the best selection for comparison between an instance-by-instance learning model versus the Learn^{++}.NSE batch-learning model. DWM maintains an ensemble of online classifiers which are re-weighted at the arrival of a new data instance. The ensemble is supplemented periodically according to an update parameter p. The ensemble may also be pruned if classifier weights decrease below a given threshold. A more detailed description of the algorithm, along with pseudocode, is available back in Section 4.1.5 (p.53). Accompanying this comparison of DWM and Learn^{++}.NSE is a characterization of DWM according to the update period p, at which classifiers are removed or added according to the error threshold θ (a value of $\theta = 0.5$ is used as in [29]).

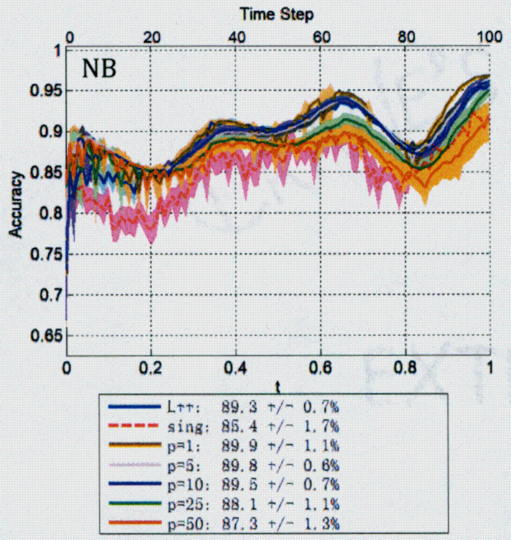


Figure 6.47: DWM weight characterization, random Gaussian drift data

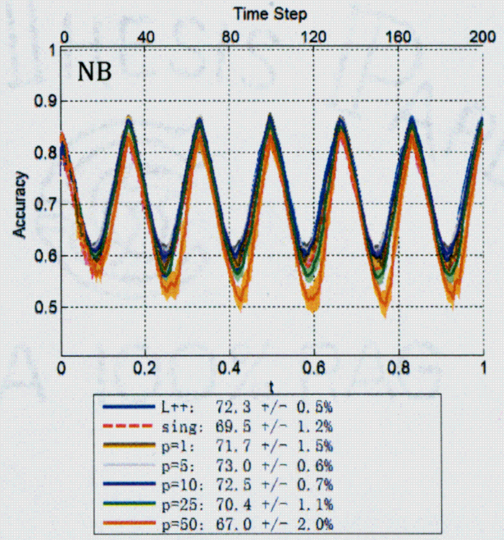


Figure 6.48: DWM weight characterization, triangular Gaussian drift data

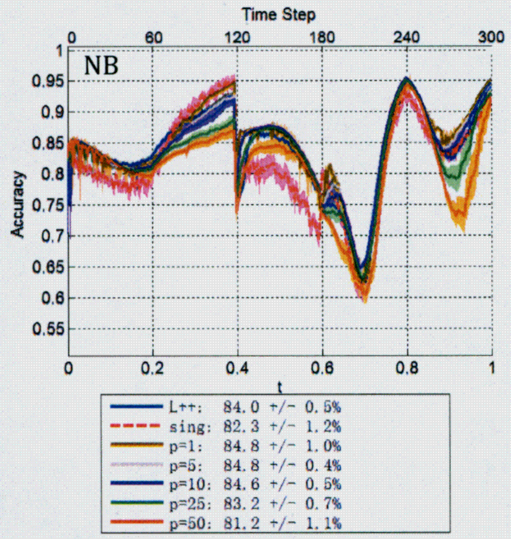


Figure 6.49: DWM weight characterization, random Gaussian drift data with class add/subtract

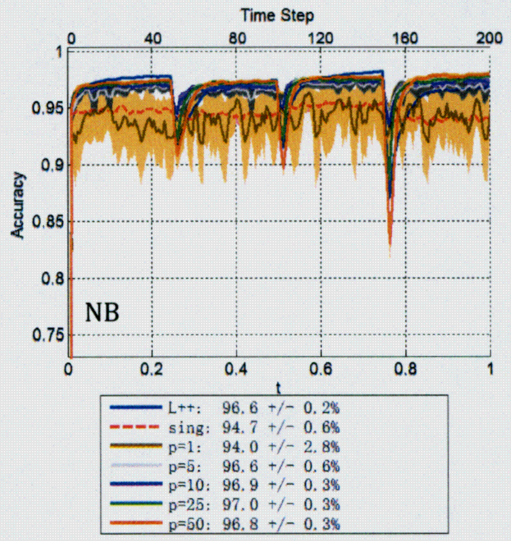


Figure 6.50: DWM weight characterization, SEA concepts data

Such a characterization indicates the practicality of adding and removing knowledge on an online basis and reveal how often an ensemble should be updated (supplemented or pruned).

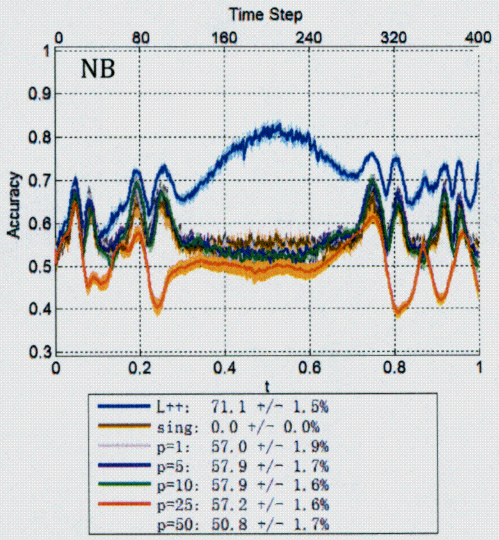


Figure 6.51: DWM weight characterization, checkerboard data (sinusoidal drift)

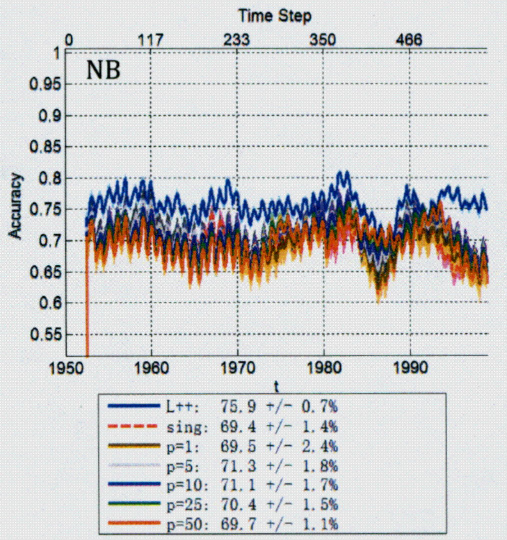


Figure 6.52: DWM weight characterization, Weather prediction data (m=30)

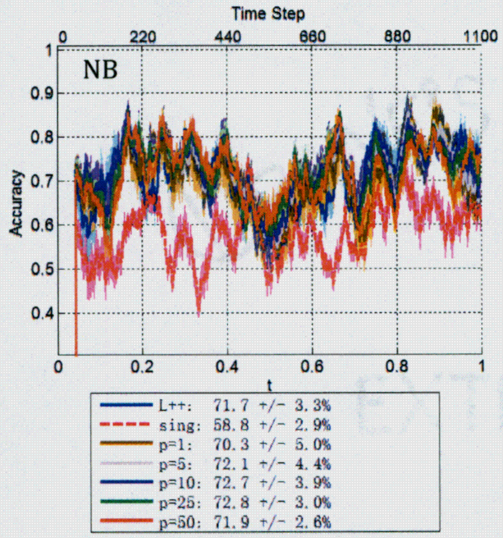


Figure 6.53: DWM weight characterization, electricity pricing prediction data (m=24)

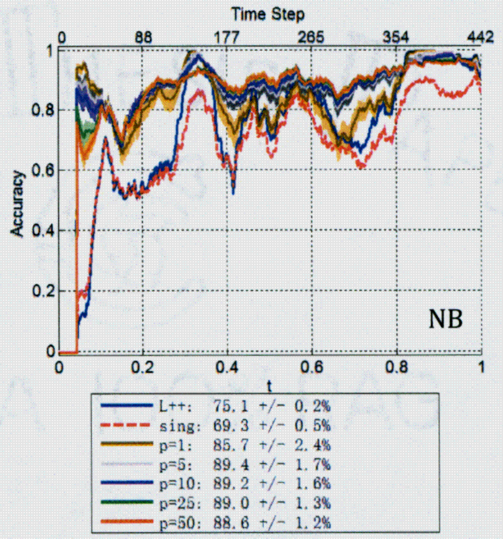


Figure 6.54: DWM weight characterization, spam prediction data (m=20)

There are noticeable differences in performance in the characterization of DWM with respect to the update period. Although difficult to generalize, we see that a frequent update period increases sensitivity to noise (SEA dataset in Figure 6.50), and infrequent updating also has the potential to significantly degrade performance, especially in periods of gradual change as in the Gaussian drift experiments (Figure 6.47-Figure 6.49) and the gradual drift period during the checkerboard dataset with sinusoidal drift rate (Figure 6.51). An update period between 5 and 25 typically has the highest overall performance; this is most clearly shown in the ANOVA comparison in Figure 6.55.

DWM features a higher recovery rate than Learn^{++}.NSE in the presence of concept change because of its quick, instance-by-instance re-weighting, as seen in the SEA dataset (Figure 6.50). Conversely, we also see the tradeoff of low convergence in stationary environment that follows the concept change.

Performance curves on the checkerboard dataset show that DWM is ineffective in learning complex, non-Gaussian distributions (Figure 6.51 and Appendix C). These results, along with the triangular drift problem (Figure 6.48) and weather data experiment (Figure 6.52), also reveal DWM's inability to retain and effectively re-use prior knowledge in the presence of recurring environments. No performance increase can be seen on those datasets which feature previously seen distributions. Although the ensemble size is not strictly limited and could theoretically maintain old knowledge, the combination of ensemble weighting and expert removal are simply too severe to make that knowledge effective at a future point in time.

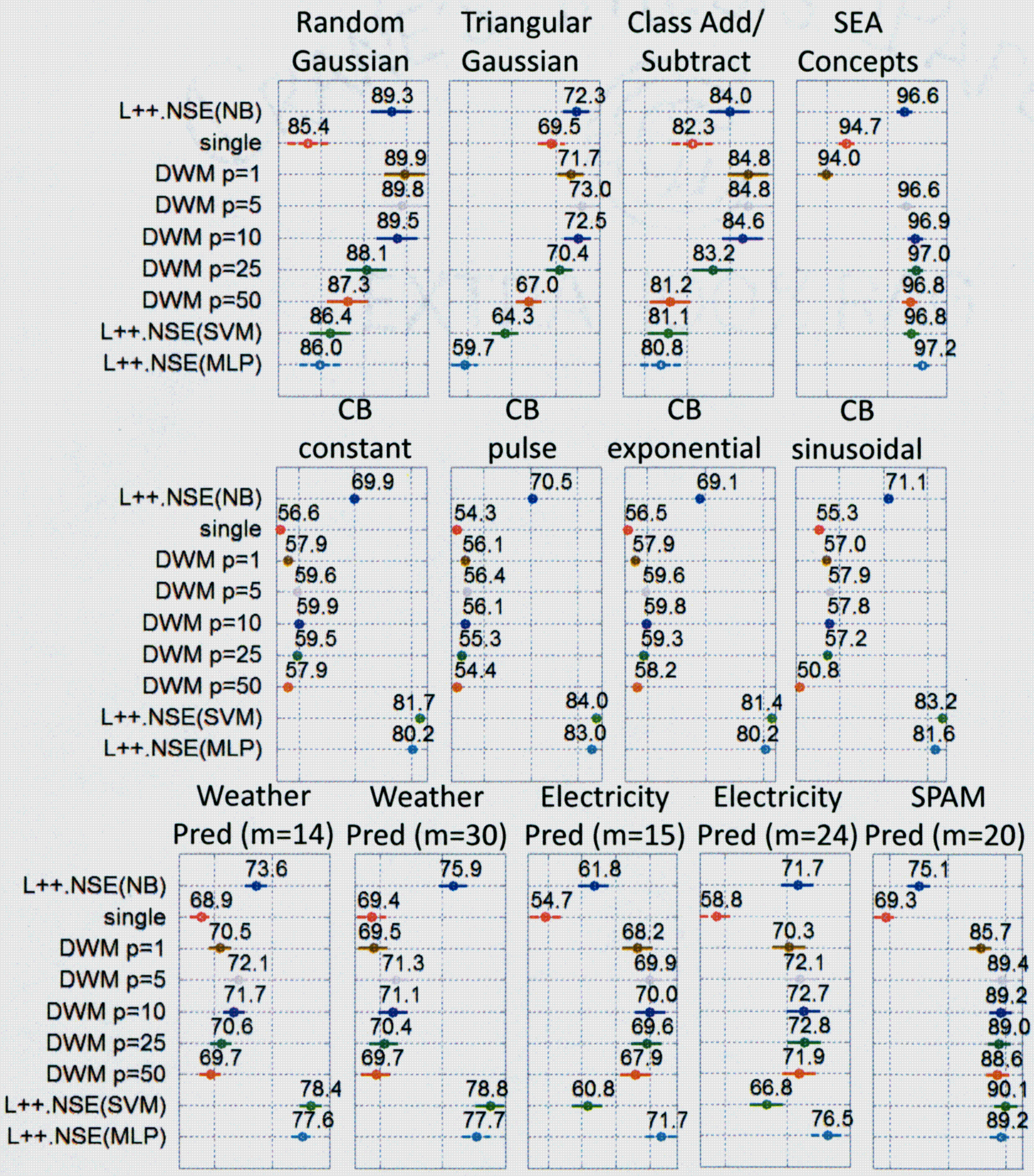


Figure 6.55: ANOVA characterization for Dynamic Weighted Majority update period (Naïve Bayes)

A key advantage of Learn^{++}.NSE lies in the independence of base classifier. Whereas in some cases DWM may match or outperform Learn^{++}.NSE using the Naïve Bayes classifier (e.g. the electricity dataset in Figure 6.53 and spam dataset in Figure 6.54), the batch based ensemble can be further enhanced by selecting an MLP neural network or

SVM as seen in the ANOVA tests in Figure 6.55, where performances of Learn^{++}.NSE using alternative base classifiers are displayed.

Once again, Learn^{++}.NSE achieves consistent performance across synthetic and real-world datasets which is unmatched by the online-learning competitor DWM. Finally, we note that although Learn^{++}.NSE is batch-based, most of these experiments use a minimal training data size, allowing for increased sensitivity to gradual changes and sufficient (yet improvable) sensitivity to abrupt change.

6.9 Controlled Forgetting (Permanent)

In this section, we compare the aforementioned controlled forgetting/pruning methods which establish a threshold or limit on the size of the knowledge base. The ensemble size is maintained by removing irrelevant classifiers to make way for the addition of new classifiers. Methods for determining which classifier is least relevant include age-based pruning, current error-based pruning, average error-based pruning, and Streaming Ensemble Algorithm (SEA). Each approach is evaluated for a given experiment under three select ensemble size thresholds for short-term memory (k=5), medium-term memory (k=25), and long-term memory (k=100). Results are accompanied by performance plots of a single classifier and the Learn^{++}.NSE *without* pruning. Select experiments are illustrated along with ANOVA comparison plots, and additional results are located in Appendix D.

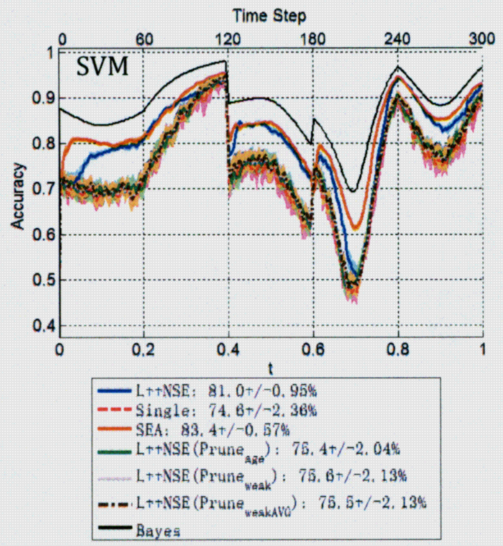


Figure 6.56: Short-term memory (k=5) for random Gaussian drift data with class addition subtraction

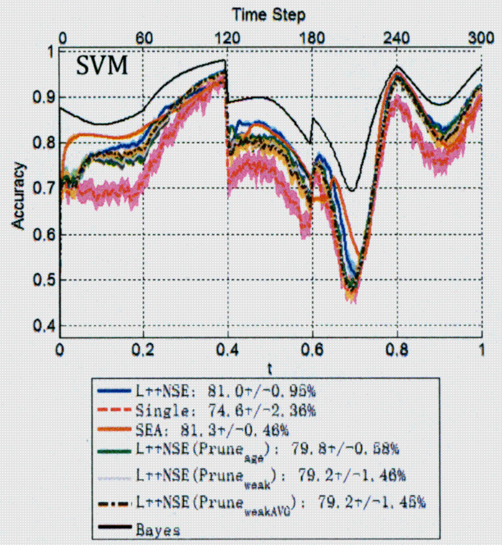


Figure 6.57: Medium-term memory (k=25) for random Gaussian drift data with class addition subtraction

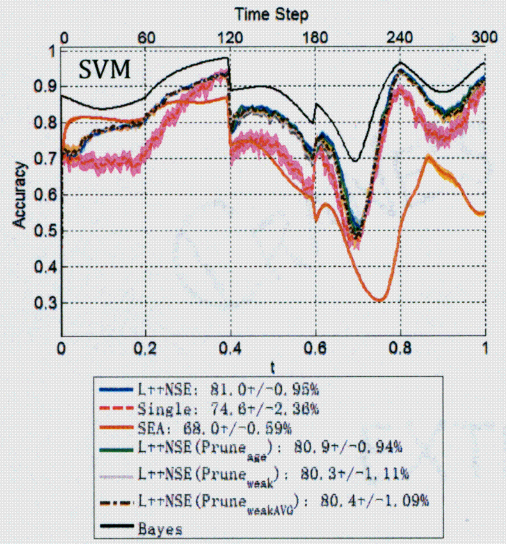


Figure 6.58: Long-term memory (k=100) for random Gaussian drift data with class addition subtraction

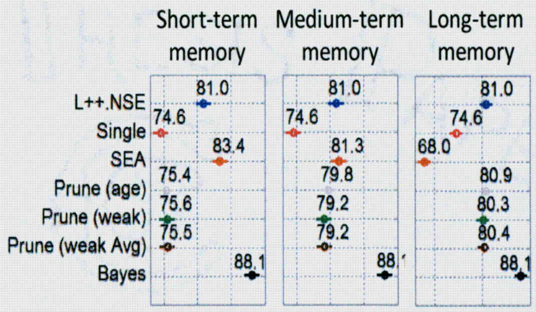


Figure 6.59: ANOVA comparison for random Gaussian drift data (class addition/removal) with varying ensemble size

In general, appears to be no significant difference between discarding classifiers based on classifier performance (current or averaged) or classifier age, the ensemble size notwithstanding.

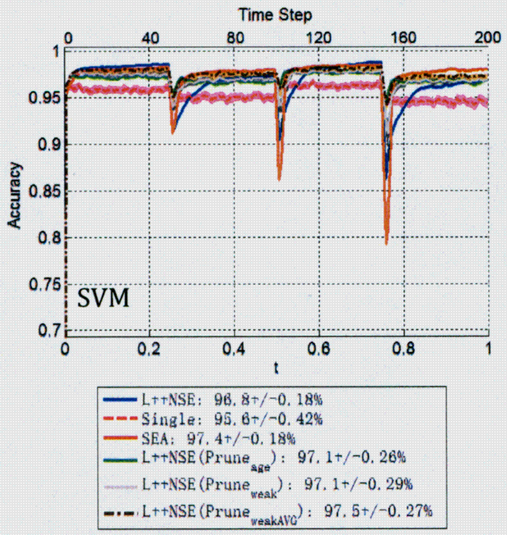


Figure 6.60: Short-term memory (k=5) for SEA concept data

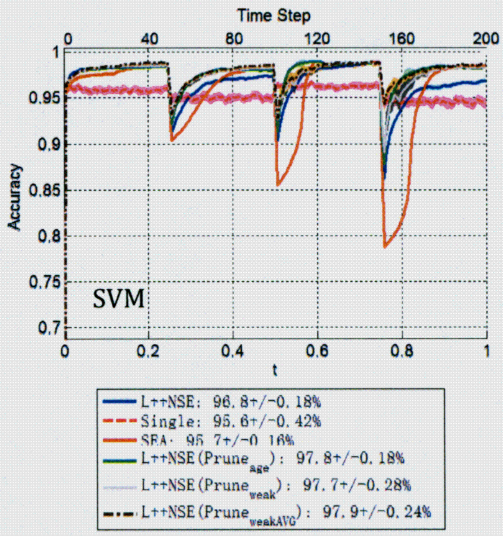


Figure 6.61: Medium-term memory (k=25) for SEA concept data

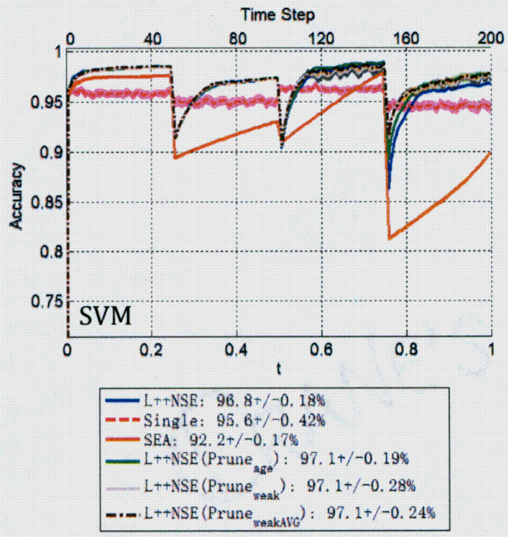


Figure 6.62: Long-term memory (k=100) for SEA concept data

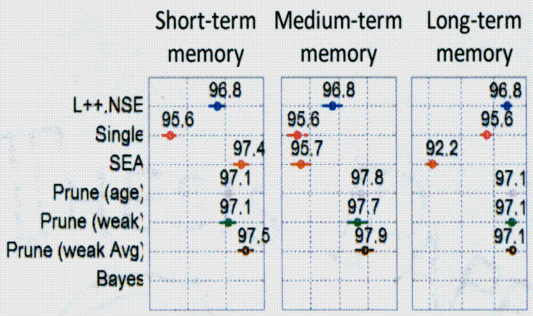


Figure 6.63: ANOVA comparison for SEA concept data with varying ensemble size

This similarity in performance is due to the fact that, most of the time, it *is* the oldest classifier that is least relevant. Time-related forgetting is in many ways implicative and consistent with human cognition.

The SEA algorithm is effective, yet only for a short-term knowledge base. Here, the pruning criterion is not the only consideration. Although SEA uses an error-based

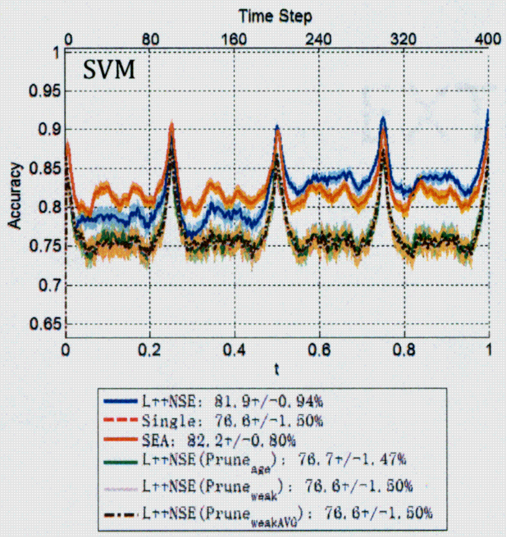


Figure 6.64: Short term memory (k=5) for checkerboard data (constant drift)

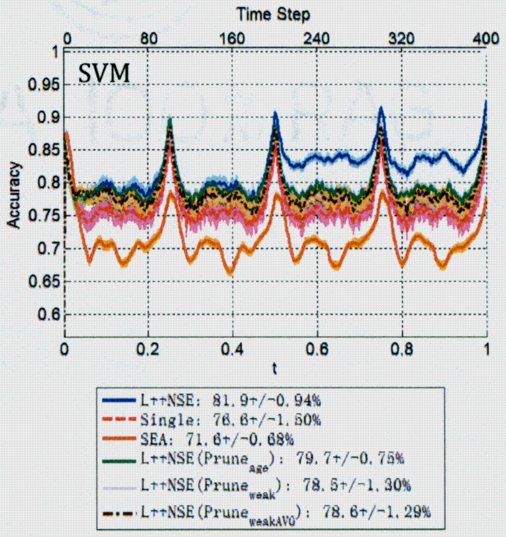


Figure 6.65: Medium term memory (k=25) for checkerboard data (constant drift)

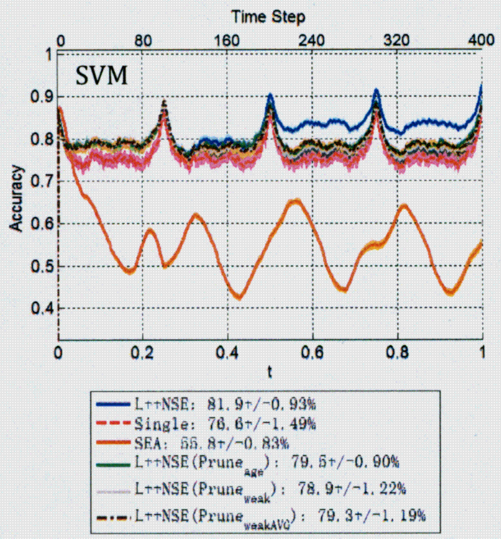


Figure 6.66: Long-term memory (k=100) for checkerboard data (constant drift)

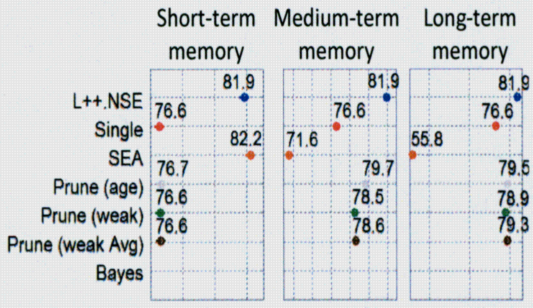


Figure 6.67: ANOVA comparison for checkerboard data (constant drift) with varying ensemble size

rule for discarding classifiers, it evaluates the ensemble using simple majority vote. In [73], Street claims no difference between a weighted and unweighted combination rule. This assessment is accurate for a medium sized ensemble of 25 or less with certain datasets; however, this claim does not apply across all datasets in this study, nor does it apply to larger ensemble sizes. The larger the knowledge base, the greater the propensity for irrelevant knowledge to be present. Since SEA uses a simple majority vote, the baggage from old classifiers increases over time and severely affects performance.

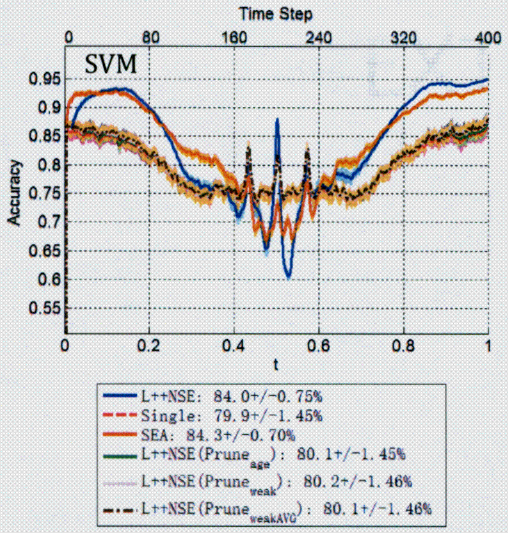


Figure 6.68: Short term memory (k=5) for checkerboard data (pulsing drift)

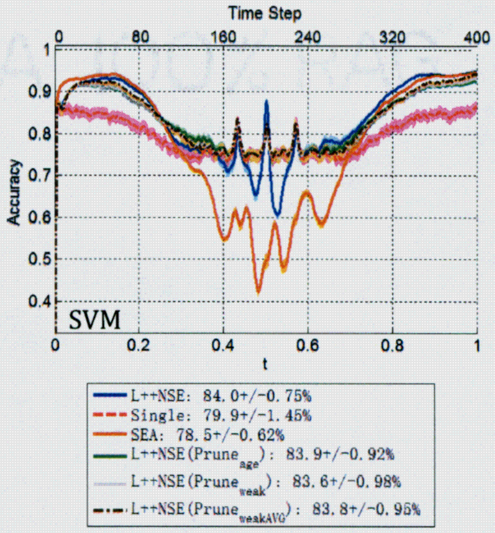


Figure 6.69: Medium term memory (k=25) for checkerboard data (pulsing drift)

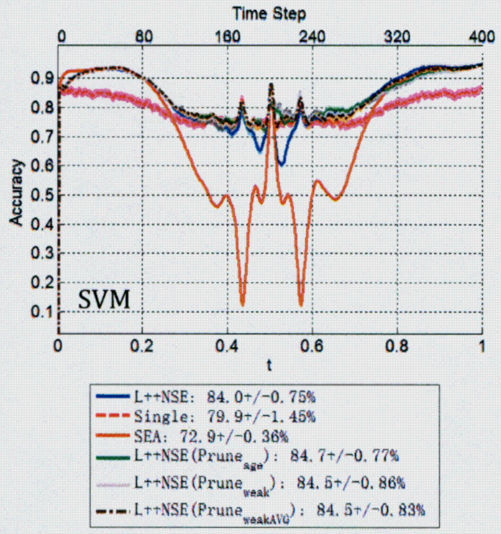


Figure 6.70: Long-term memory (k=100) for checkerboard data (pulsing drift)

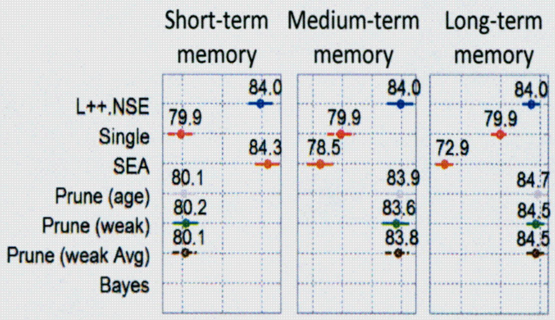


Figure 6.71: ANOVA comparison for checkerboard data (pulsing drift) with varying ensemble size

In periods of sharp drift (SEA concepts and checkerboard with pulsing drift), the large weighted ensemble also suffers slightly from baggage, as virtually *all* prior knowledge becomes irrelevant. In this case, the reaction time is significantly reduced (i.e. improved) with forgetting, inversely proportional to the size of the knowledge base.

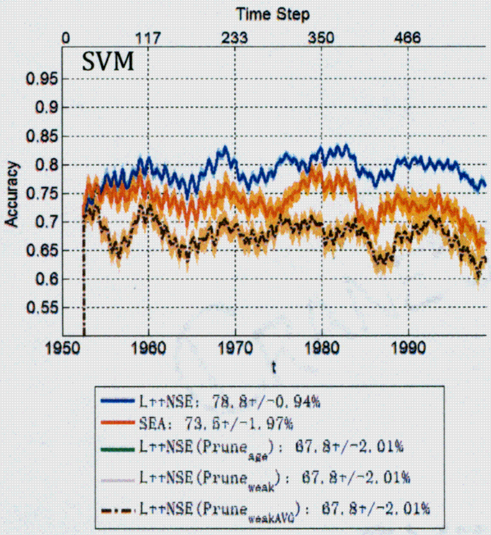


Figure 6.72: Short term memory (k=5) for weather prediction data (m=30)

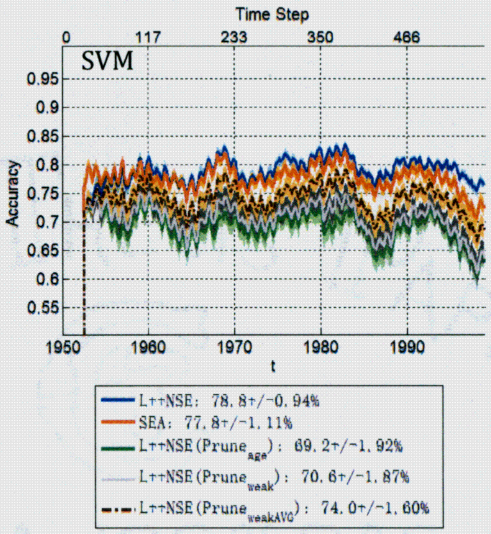


Figure 6.73: Medium term memory (k=25) for weather prediction data (m=30)

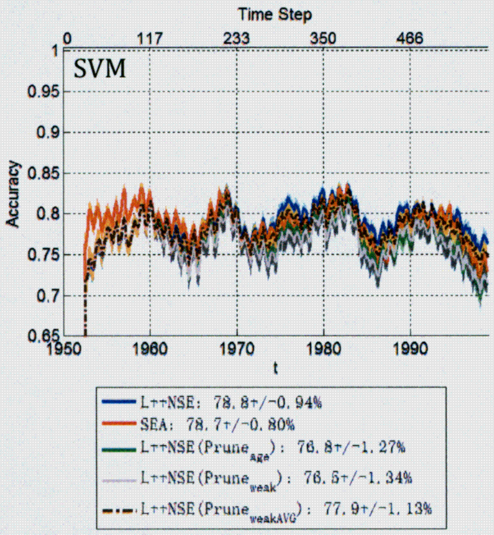


Figure 6.74: Long-term memory (k=100) for weather prediction data (m=30)

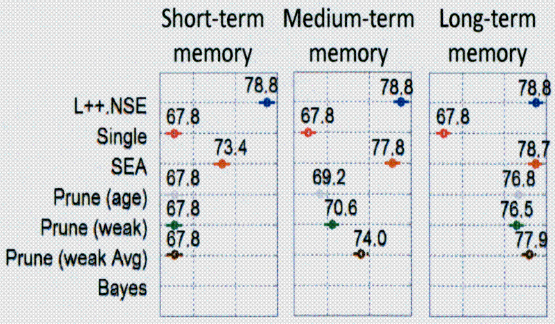


Figure 6.75: ANOVA comparison for weather prediction data (m=30) with varying ensemble size

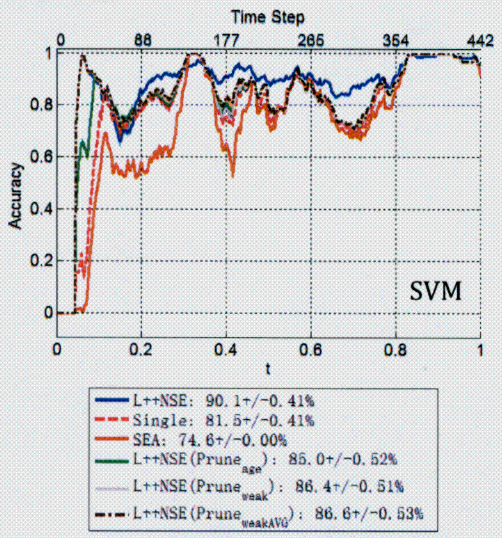


Figure 6.76: Short term memory (k=5) for spam prediction data (m=20)

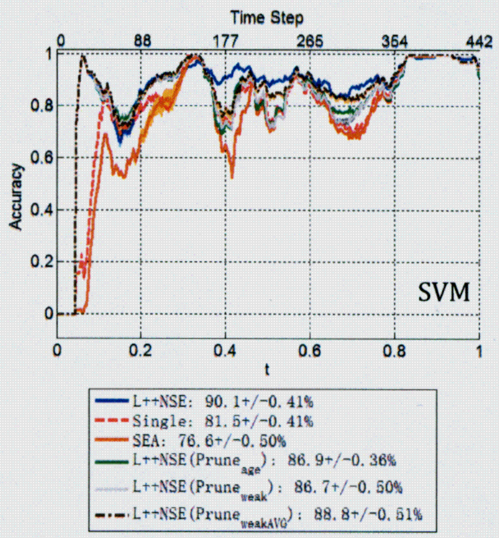


Figure 6.77: Medium term memory (k=25) for spam prediction data (m=20)

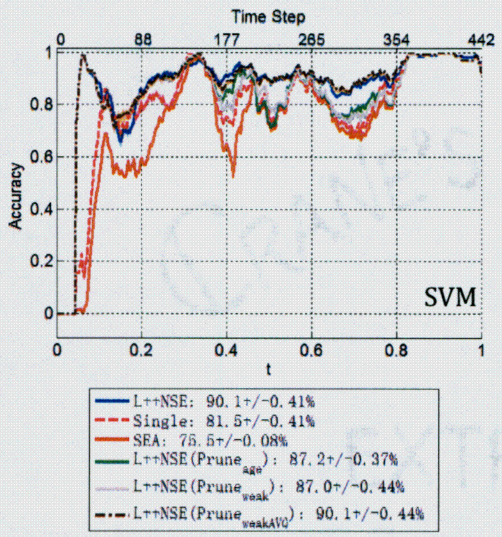


Figure 6.78: Long-term memory (k=100) for spam prediction data (m=20)

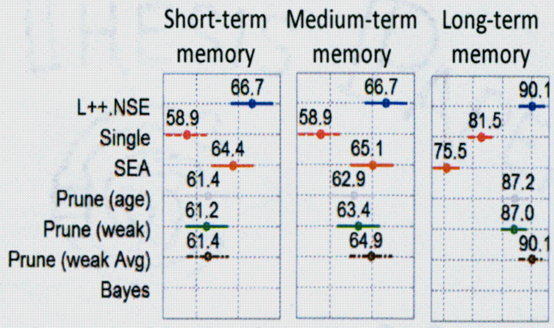


Figure 6.79: ANOVA comparison for spam prediction data (m=20) with varying ensemble size

Performance curves from the checkerboard data (Figure 6.64-Figure 6.71) show that limiting memory using pruning is crippling to the knowledge base, even in long-term memory with a large ensemble size of 100. The detriment is often independent of recurring environments and applies to real-world data as well, as we see a characteristic increase toward the performance of Learn^{++}.NSE as the ensemble size grows. Inability to retain knowledge from previous environments is the most convincing evidence that pruning should be avoided.

A characterization of individual algorithms over ensemble size is provided in the Appendix D for all remaining experiments. The results of these experiments also support the notion that as the ensemble size is increased, performance will approach to that of the *un*pruned (i.e. unlimited) knowledge base. The exception is periods of severe drift or concept change during which the weight allocation method is unable to negate irrelevant knowledge.

6.10 Controlled Forgetting (Temporary)

Temporary pruning (or expert selection) provides benefits that are unique from permanent pruning, since classifiers which are ignored at a given time may in fact be used at a later time because no memory is permanently lost. Temporary pruning avoids the catastrophic nature of the previous forgetting methods. Conversely, temporary pruning does *not* benefit from the computational complexity and memory reduction that accompanies a limited knowledge base. Experts are selected based on performance; thus, the entire knowledge base must still be evaluated and categorized at each time step even though not all knowledge will be used.

Experiments show that expert selection methods are most useful during periods of severe drift or concept change. The significant outperformance over Learn^{++}.NSE in the checkerboard data with pulsing drift (Figure 6.85) and SEA dataset (Figure 6.83) indicates the presence of some baggage in the Learn^{++}.NSE ensemble that can be eliminated with temporary pruning.

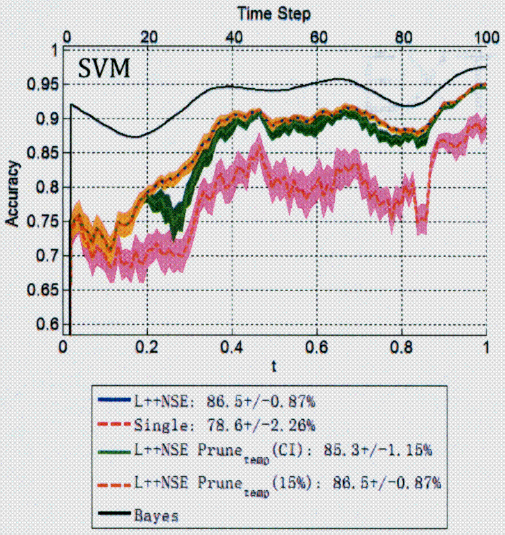


Figure 6.80: Temporary pruning comparison for random Gaussian drift data

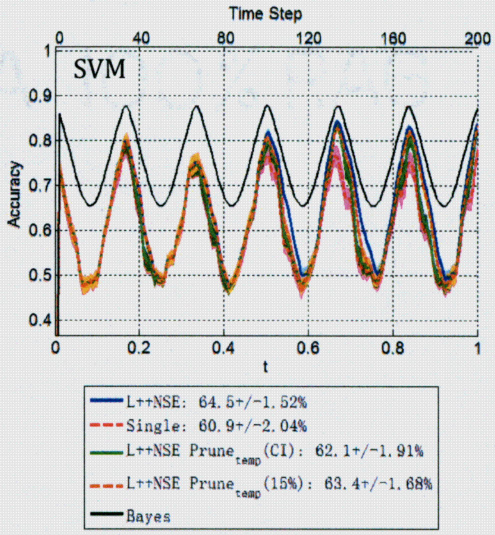


Figure 6.81: Temporary pruning comparison for triangular Gaussian drift data

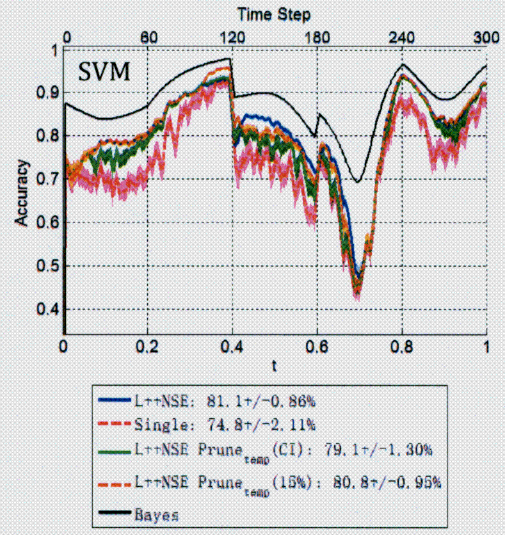


Figure 6.82: Temporary pruning comparison for random Gaussian drift data with class addition/removal

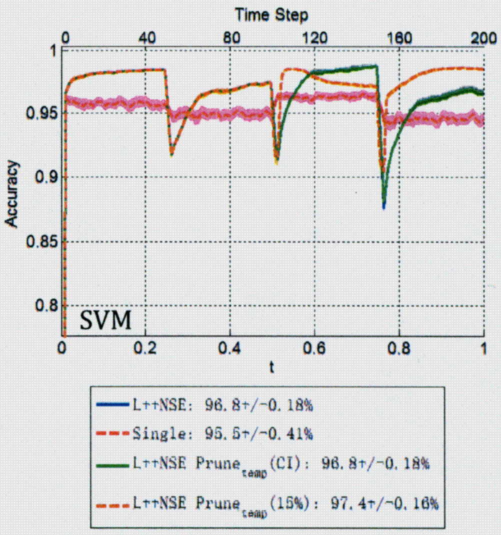


Figure 6.83: Temporary pruning comparison for SEA concepts data

Real-world data provides a variety of inferences, as seen (generally) in the ANOVA plots in Figure 6.91. Because the weather dataset is periodic in nature over the course of the entire experiment, the learner benefits from using all available knowledge as seen in Figure 6.88, since Learn^{++}.NSE significantly outperforms other methods. Drift in the electricity dataset appears to be more sporadic in nature; thus, the possibility of classifier baggage is increased, and the expert selection methods may provide an improvement over the unpruned ensemble (Figure 6.89). The same reasoning applies to the spam dataset as well, where the pruned ensembles perform no worse than unpruned Learn^{++}.NSE (Figure 6.90).

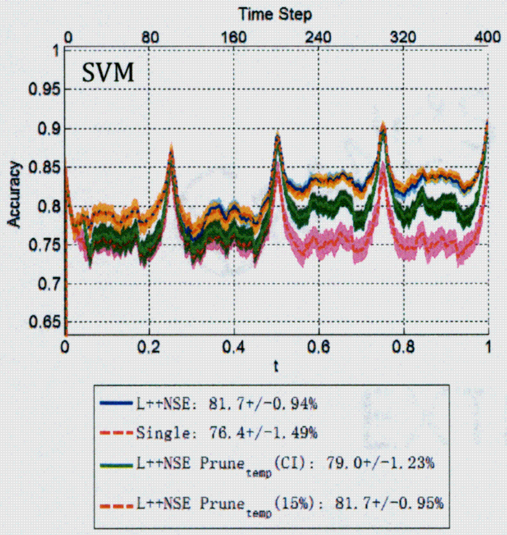


Figure 6.84: Temporary pruning comparison for checkerboard data (constant drift)

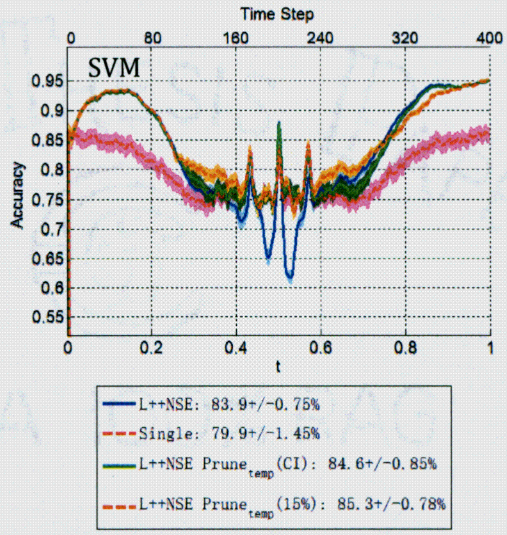


Figure 6.85: Temporary pruning comparison for checkerboard data (pulsing drift)

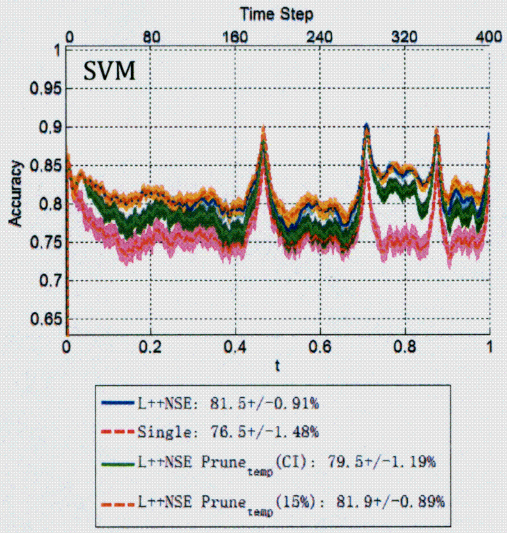


Figure 6.86: Temporary pruning comparison for checkerboard data (exponential drift)

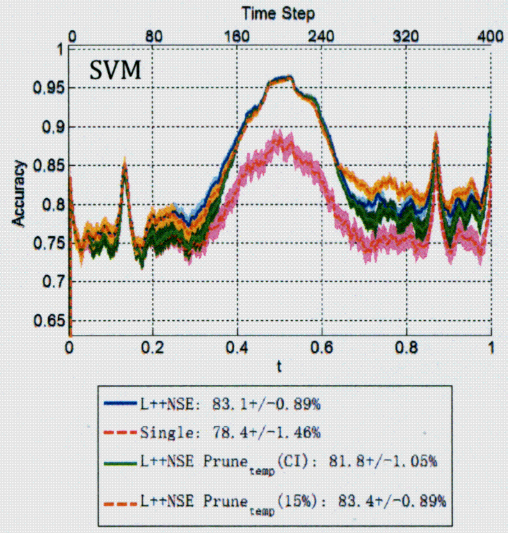


Figure 6.87: Temporary pruning comparison for checkerboard data (sinusoidal drift)

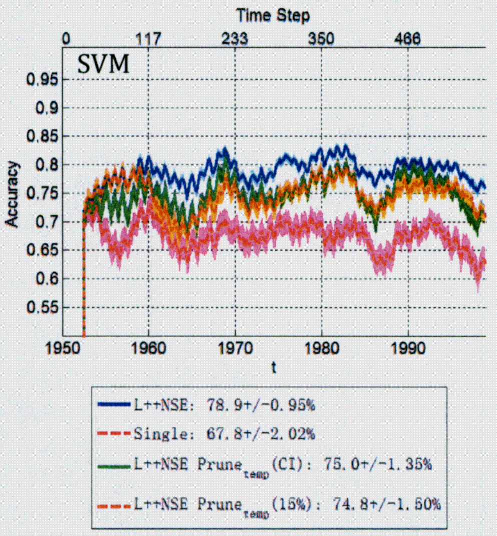


Figure 6.88: Temporary pruning comparison for weather prediction data (m=30)

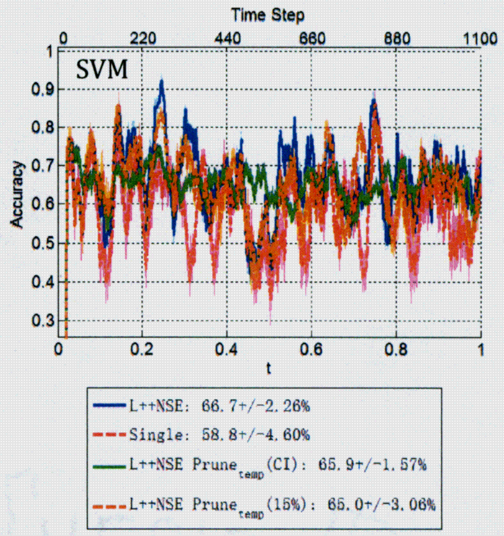


Figure 6.89: Temporary pruning comparison for electricity pricing data prediction (m=24)

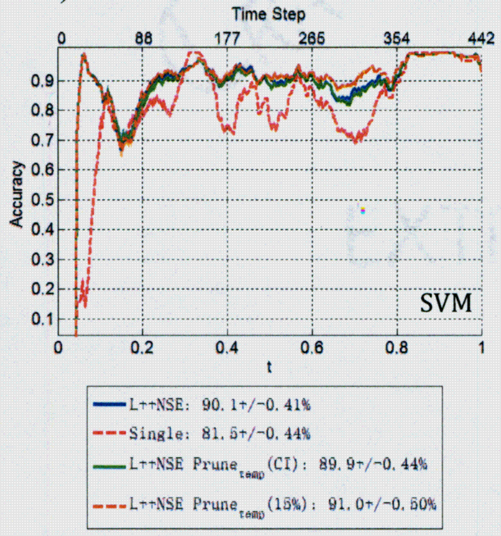


Figure 6.90: Temporary pruning comparison for spam data prediction (m=20)

A confidence interval approach for selecting experts has some inconsistencies in these experiments, and is mostly limited by in the way it is implemented under the Learn^{++}.NSE architecture. The weight analysis in Section 6.7 shows that the most recent classifier at any given time commonly gets a very large weight compared to older classifiers. Confidence interval-based pruning uses the best performer as the basis for creating the confidence interval. In a practical sense, it is difficult a confidence interval sufficiently large confidence interval to include other classifiers. Consequently, the expert ensemble that is selected often excludes classifiers with relevant knowledge, resulting in decreased performance as seen in synthetic experiments undergoing levels of steady drift (Figure 6.84, Figure 6.86-Figure 6.87).

The constant percentage selection method uses the final classifier weight as the basis for ignoring irrelevant classifiers; this appears to be a more consistent and effective basis for selecting classifiers. Percentage-based temporary pruning is seldom out-performed by the unpruned Learn^{++}.NSE ensemble and is able to provide a significant improvement when baggage is a threat, that is, times of severe drift or an occurrence of drift after a stationary period as seen in the checkerboard (Figure 6.85-Figure 6.87) and SEA datasets (Figure 6.83). In nearly all experiments, percentage-based selection performs at least as good as the entire ensemble, and often significantly outperforms, especially (1) when the knowledge base grows (increased baggage), and (2) when drift is severe (few relevant classifiers).

Considering the issue of stability and plasticity in non-stationary environments, we realize that the amount of knowledge that should be used at a given time may vary depending on the drift situation. It is quite possible that consistent improvements can be attained across a variety of drift rates by selecting (or ignoring) and appropriate percentage of the knowledge base. This, however, would necessitate a variable percentage to be employed.

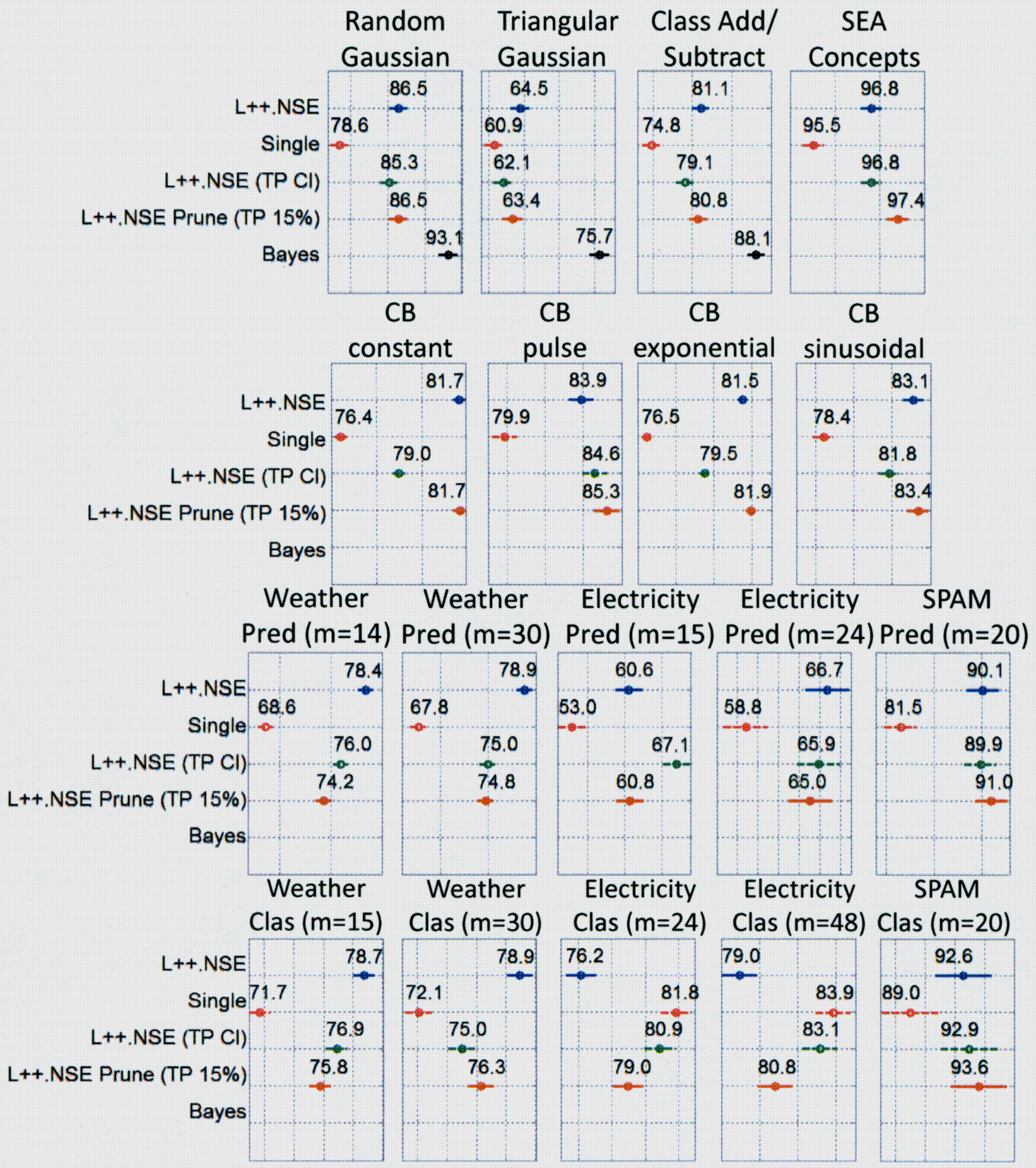


Figure 6.91: ANOVA comparison for temporary pruning comparison (SVM)

6.11 Drift Detection

Earlier, we discussed the role of the supervisor or teacher in the learning process as it pertains to both human and computational learning. Scaffolding theory provides some general guidelines which include monitoring changes in either the incoming data itself *or* the learner's ability to accommodate new data. The purpose of actively detecting changes is two-fold: (1) to allow the learner to make adjustments that best suit the current environment, and (2) to control the flow of data used for training.

We have introduced a drift-detection mechanism described in Section 5.3, which is a performance-based approach to tracking changes from batch to batch as data are presented to the learner. The approach occurs in three phases: training, configuration, and testing. The *training* phase prepares a drift-detection classifier h_{DD} on the current environment using M examples from the training data presented to the learner. The mechanism is *configured* by calculating performance on a configuration sequence (size CS) of data batches; the distribution (mean and variance) of this sequence of performances is used as the null hypothesis θ^0 to which future performances will be compared. Alternative hypotheses are introduced using the confidence intervals about the null hypothesis. The log-likelihood of belonging to either distribution is calculated *within* the configuration sequence and is then used to obtain the threshold for the test. In the *testing* phase, likelihood is computed on performance values for each incoming batch of data. Drift is detected when the configuration threshold is exceeded.

Before putting the performance-based drift detection mechanism to use, it is first important to understand the effect of the parameters involved with configuring the detection test. These include the training size M of the drift detection classifier h_{DD}, the configuration size CS over which to form the null and alternative distributions (P_{θ^0} and P_{θ^1}, respectively), and the proportion of the confidence interval for computing the alternative distributions. The training size M should be sampled as the minimum yet ample amount of data from the current environment in order to create a stable performance curve for configuring the test. The configuration size CS should be larger than the number of batches used to train the configuration classifier, and should be sufficient sample size from which to create a distribution according to the Central Limit Theorem. Finally, the alternative distributions, initially defined by a 95% confidence interval, can be heuristically constrained with the goal of preventing a confidence interval that is too wide or too narrow for considering changes in performance, ultimately resulting in false detections. For example, if there is high variance within the configuration window, the alternative distributions could be drastically different from the null distribution (perhaps beyond 50% away), thus increasing the propensity for false negative detection. *False negative* detection occurs when the drift detector is *not* triggered, even though drift is in fact occurring. Because performance is limited between 0% and 100%, we reason that a cap can be put on the distance between a null and alternative hypothesis, which represents the maximum distance that an alternative hypothesis may lie on the performance chart. Similarly, in the rare event that training data is so invariant as to produce a miniscule difference between the null and alternate hypotheses, we impose a limit on the minimum confidence interval to prevent false-positive detection. *False positive* detection occurs when the drift detector is triggered, even though drift has *not* actually occurred.

A configuration size of $CT = 20$ is selected for the experiments described below (Figure 6.92-Figure 6.95). This value was selected among alternative configurations (see Appendix E) as it provides a good balance between noise-sensitivity (low CT) and overgeneralization/slow reaction (high CT).

The following illustrations show both the performance over a single run of the drift detection mechanism as well as the generalized performance over multiple experiments. In each figure, plot A (upper) shows the performance of the drift-detection classifier over time, showing the configuration phase (black), confidence interval derived from the configuration (green), the testing phase (blue), and each instance where drift is detected (red). Below each drift detection instance is a number indicating which alternative hypothesis triggered the detection. The number "1" refers to the lower confidence interval (m- in the legend), and "2" refers to the upper confidence interval

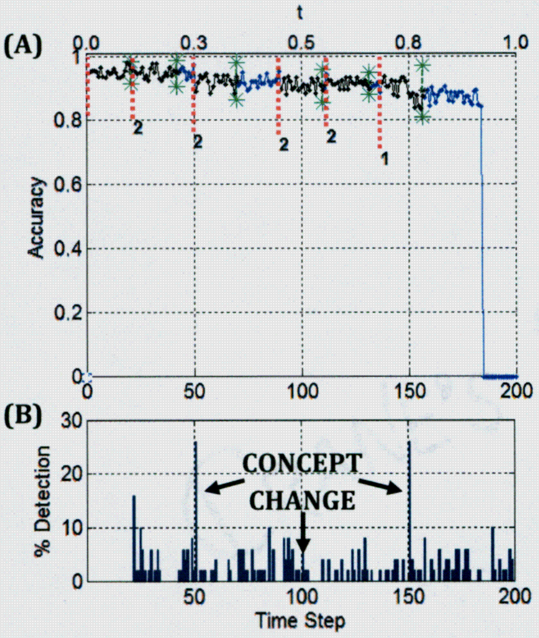


Figure 6.92: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on SEA concepts data

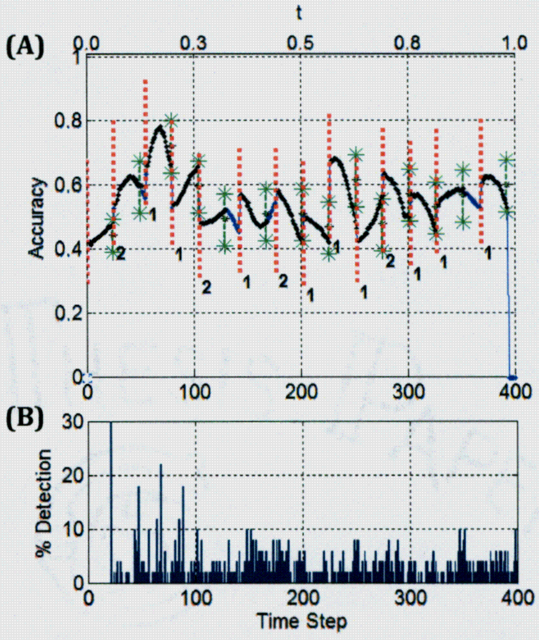
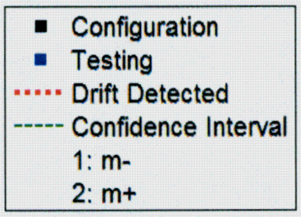


Figure 6.93: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on checkerboard data (constant drift)



(m+ in the legend). Plot B (lower) shows the percentage of drift detection over 50 trials in order to indicate consistency of the drift detector.

The detection mechanism is effective in tracking all types of changes in the data. Experiments such as the SEA (Figure 6.92) and checkerboard (Figure 6.93-Figure 6.95) datasets are especially useful in evaluating accuracy in a variety of drift situations. For environments which are changing steadily such as the constant drift checkerboard data (Figure 6.93), we see a steady decrease in classifier performance from the time of training, often within the configuration period as well, clearly showing that drift detection

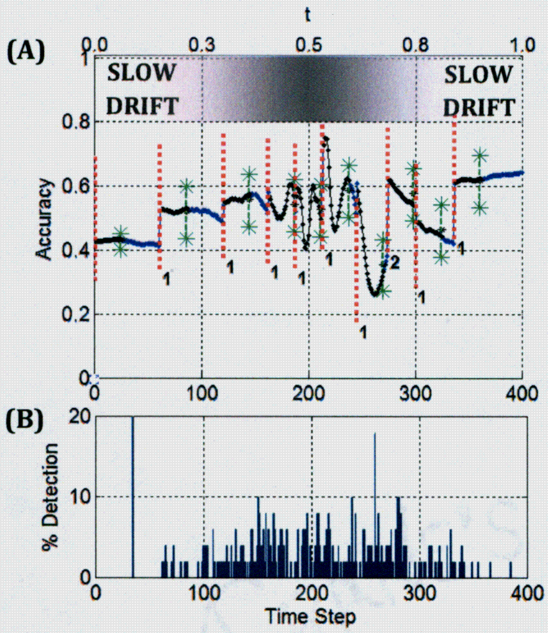


Figure 6.94: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on checkerboard data (pulsing drift)

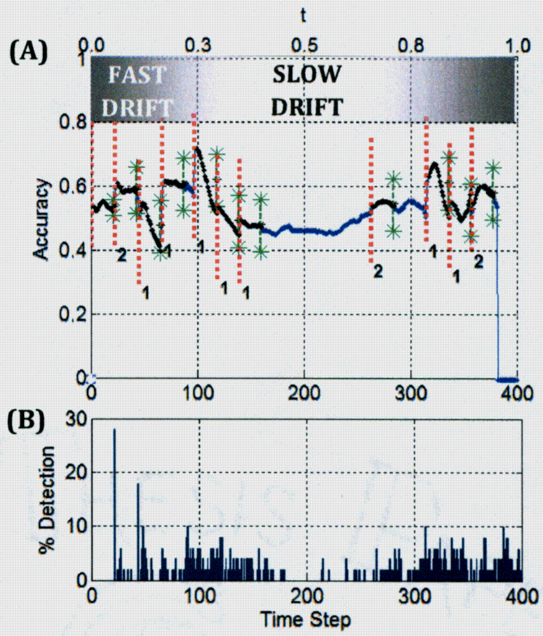


Figure 6.95: Classifier performance (Plot A) and detection % per time step over 50 trials (Plot B) for drift detection on checkerboard data (sinusoidal drift)

■	Configuration
■	Testing
·····	Drift Detected
-----	Confidence Interval
	1: m-
	2: m+

classifier is becoming less and less competent as the data changes. Intervals of gradual or no drift are characterized by low variance within the configuration period.

The mechanism is not perfect, however. From these examples we see some level of inconsistency, mostly in the form of false-positives caused by over-sensitivity. For instance, the periods of stationarity in the SEA data are prone to yield false positives, in part because of noise within the training data (Figure 6.92). Also, within the checkerboard data with varying drift rates (Figure 6.94-Figure 6.95) there is some inconsistency with detecting the stationary periods, resulting in some false positive detection. The repercussions of these inconsistencies are discussed within the following sections as the drift detection is applied to different scaffolding techniques.

6.11.1 Sigmoid Characterization for Appropriate Parameter Selection

A primary goal of using drift detection is to provide the learner with information necessary to alter its architecture to best accommodate the current environment. In this section, we investigate a particular aspect of the Learn^{++}.NSE ensemble architecture which could possibly be adjusted dynamically in accordance with the information provided by a drift detector.

The time-weighted sigmoid is an important feature of the Learn^{++}.NSE algorithm used in the process of calculating classifier weights. The purpose of the sigmoid is to give more consideration to recent error and less consideration to prior error for a given classifier when determining its voting weight. Two parameters, *a* and *b*, define the sharpness of the cutoff and the location of the cutoff (*how much* prior error to consider), respectively. The purpose in characterizing these parameters is to determine whether or not they may be optimized or heuristically adjusted to accommodate various drift situations.

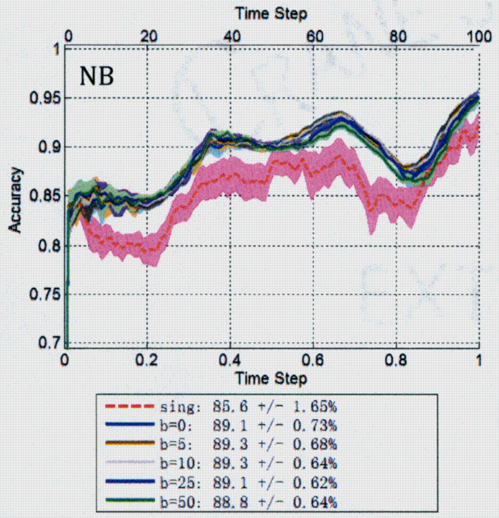


Figure 6.96: Time-weighting sigmoid characterization for random Gaussian drift data ($a = 0.5$)

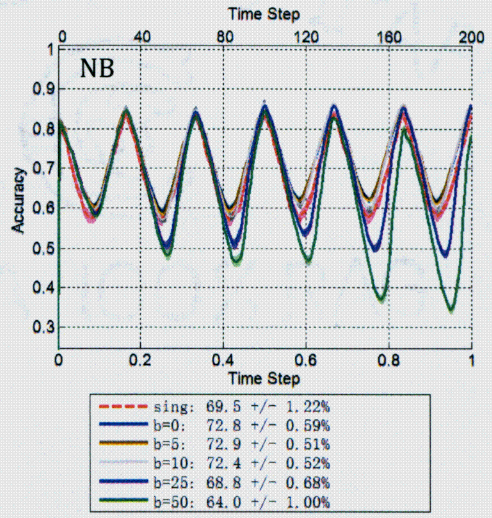


Figure 6.97: Time-weighted sigmoid characterization for triangular Gaussian drift data ($a = 0.5$)

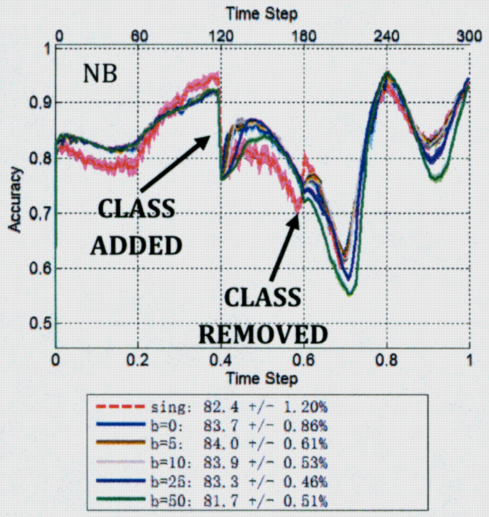


Figure 6.98: Time-weighted sigmoid characterization for random Gaussian drift data with class addition/removal ($a = 0.5$)

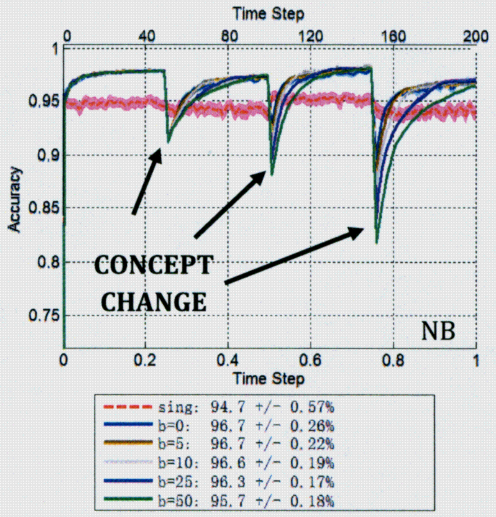


Figure 6.99: Time-weighted sigmoid characterization for SEA concepts data ($a = 0.5$)

Characterization across synthetic datasets reveals that the slope parameter has little or no effect on performance in different drift situations (see Appendix F), whereas the cutoff parameter b does. In characterizing this parameter under different base classifiers (Naïve

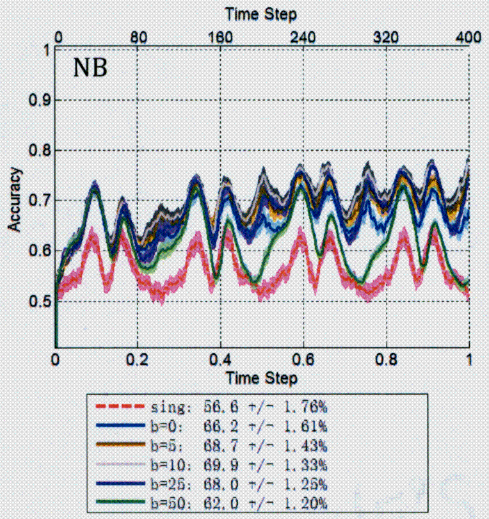


Figure 6.100: Time-weighted sigmoid characterization for checkerboard data (constant drift) ($a = 0.5$)

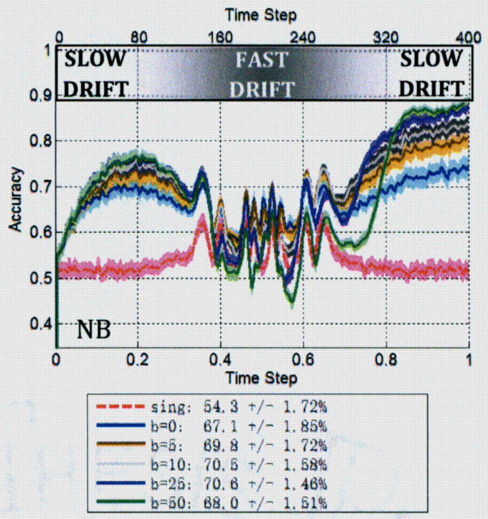


Figure 6.101: Time-weighted sigmoid characterization for checkerboard data (pulsing drift) ($a = 0.5$)

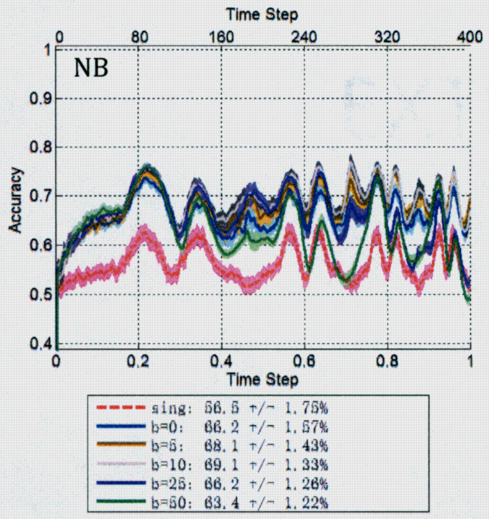


Figure 6.102: Time-weighted sigmoid characterization for checkerboard data (exponential drift) ($a = 0.5$)

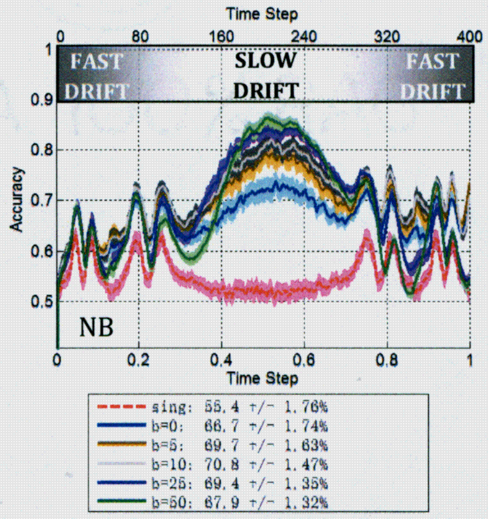


Figure 6.103: Time-weighted sigmoid characterization for checkerboard data (sinusoidal drift) ($a = 0.5$)

Bayes, SVM, MLP), we discover that the Naïve Bayes classifier yields most descriptive characterization in different drift situations (Figure 6.104), whereas the other base classifier models are less descriptive (as seen in the Appendix F).

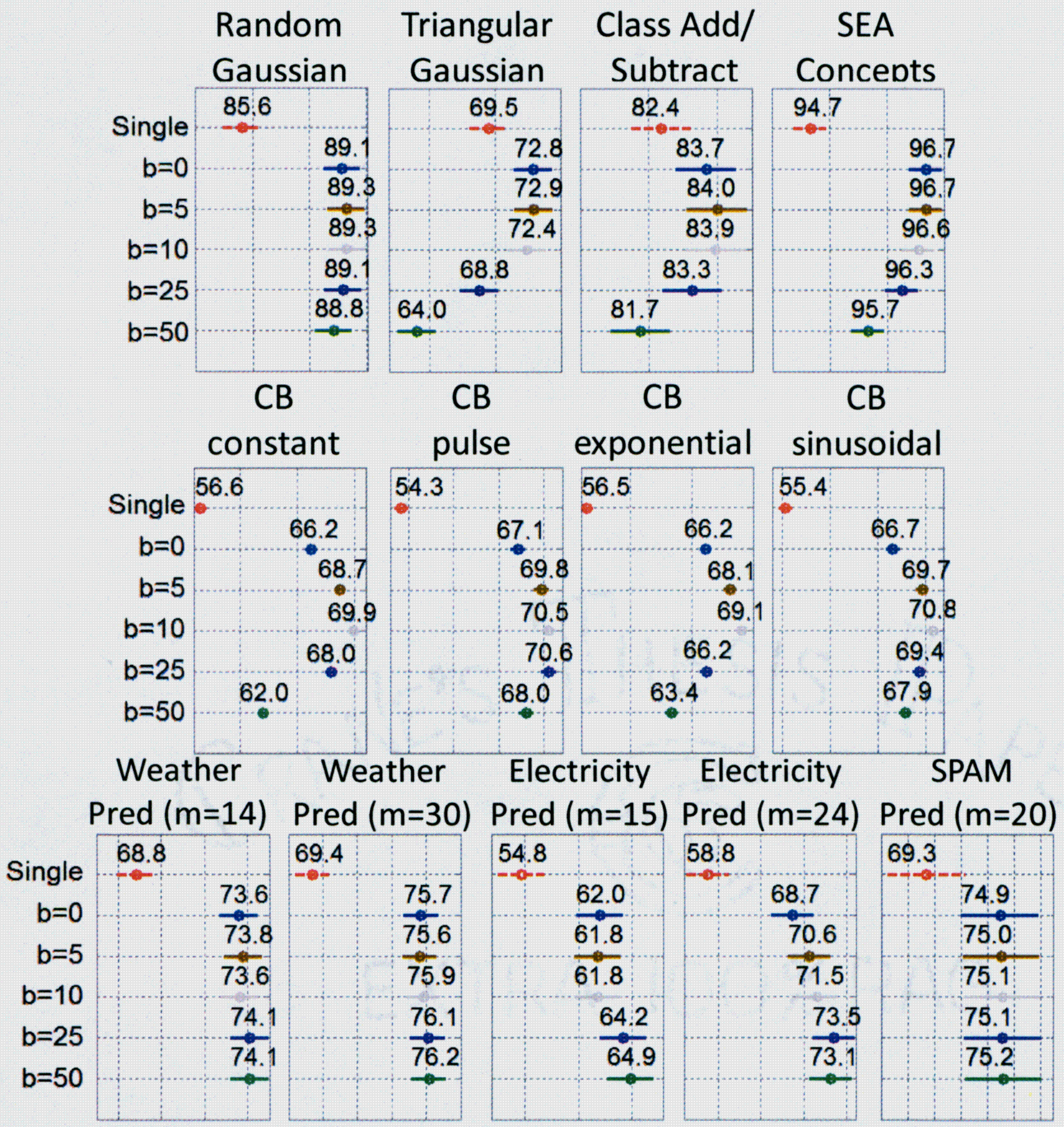


Figure 6.104: ANOVA comparison for time-weighted sigmoid characterization (b parameter, $a = 0.5$) (Naïve Bayes)

In some cases, there is little difference in performance for environments with gradual drift and relatively simple decision boundaries (Figure 6.96). Yet, for some more complex steady drift cases (triangular Gaussian drift in Figure 6.97 and checkerboard data in Figure 6.100), there appears to be a near-optimal setting for the cutoff parameter (b = 10); this is indicated by the ANOVA plots as well (Figure 6.104). This value is used as a default in the base Learn^{++}.NSE model. However, in periods with little or no drift (Figure 6.101, Figure 6.103), a higher cutoff is preferred, so as to allow more previous error to be considered in calculating weights. In periods of sharp drift, especially following periods of stationarity (Figure 6.98-Figure 6.99, Figure 6.101), a low cutoff is desired to allow knowledge to be categorized *only* on the new environment that has suddenly appeared. These trends are the rationale behind choosing sigmoid values in accordance with the drift detection mechanism.

There are two primary concerns that arise from this analysis. The first is the lack of consistency across base classifiers, and second is the lack of consistency in the crossover to real-world datasets, as seen in the ANOVA plots (Figure 6.104).

6.11.2 Scaffolding Techniques Using Drift Detection

Two scaffolding techniques have been introduced under the architecture of the Learn^{++}.NSE algorithm to work in conjunction with the drift detection method. The *fading* technique is introduced to prevent unnecessary growth of the knowledge base with redundant knowledge; this is accomplished by halting the learning process during a perceived stationary environment. Both methods are tested across all base classifiers on all experiments; for steady drift cases, both methods should perform the same as the base Learn^{++}.NSE model, since default sigmoid parameters are used and all incoming data are relevant. However, as periods of slow and fast drift appear, we begin to see characteristics arise from the methods under consideration.

Table 6.7: Dynamic sigmoid adjustment procedure

Drift Scenario	**Response for sigmoid cutoff parameter *b***
Stationary Period (no drift detected after $\gamma = 15$ time steps during testing phase)	Increment by $\phi_s =$ until it reaches stationary parameter b_s.
Drift detected during stationary period	Set to the drift constant b_d and incremented by drift factor ϕ_d until reaching the optimal constant b_o for steady drift
Steady drift (drift detected before 15 time steps during testing phase)	Set to optimal constant b_o for steady drift

Dynamic sigmoid adjustment (*DynSig)* is a heuristic approach to attaining appropriate classifier voting weights for the current drift situation. The approach evaluates the drift situation, and alters the sigmoid cutoff parameter *b*, as seen in the proposed pseudocode in Section 5.3.2, Figure 5.10. The basic protocol, along with exact parameter values used in all experiments, can also be seen in Table 6.7.

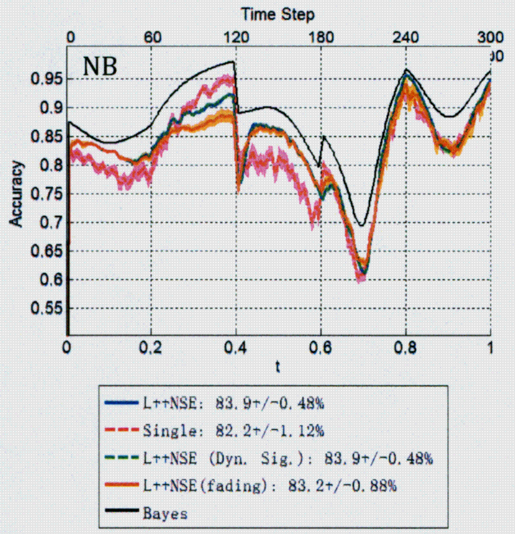


Figure 6.105: Scaffolding comparison for random Gaussian drift data with class addition/removal using Naïve Bayes

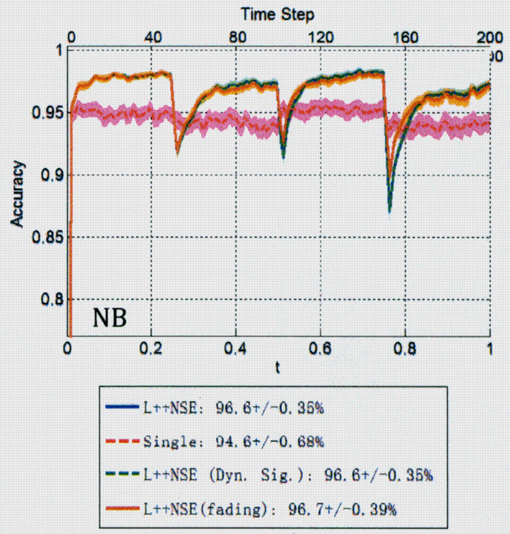


Figure 6.106: Scaffolding comparison for SEA Concepts data using Naïve Bayes

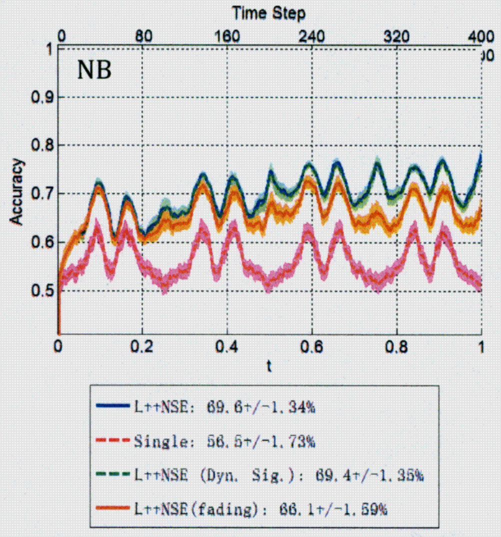


Figure 6.107: Scaffolding comparison for checkerboard data (constant drift) using Naive Bayes

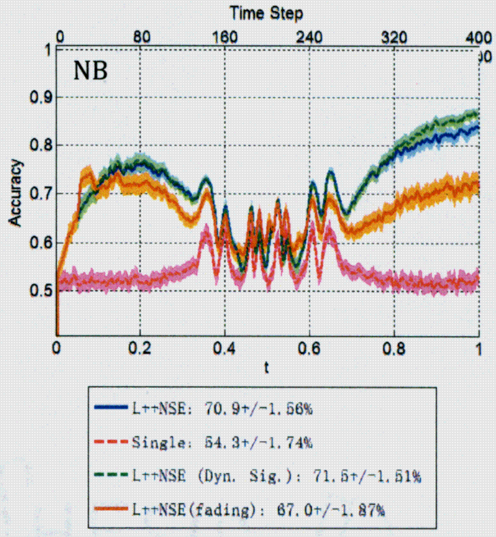


Figure 6.108: Scaffolding comparison for checkerboard data (pulsing drift) using Naive Bayes

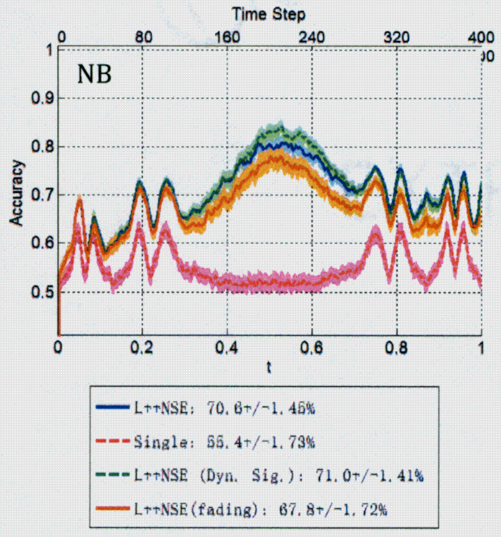


Figure 6.109: Scaffolding comparison for checkerboard data (sinusoidal drift) using Naive Bayes

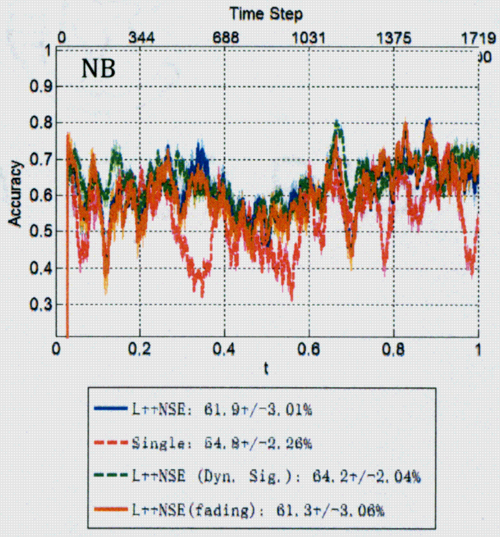


Figure 6.110: Scaffolding comparison for electricity pricing data (m=15) using Naive Bayes

Overall, the *dynamic sigmoid* ensemble performs no worse than the Learn^{++}.NSE ensemble. In cases involving slow drift (checkerboard datasets in Figure 6.108-Figure 6.109), *dynamic sigmoid adjustment* is a statistically significant improvement over Learn^{++}.NSE as it is able to assign appropriate sigmoid parameters for computing classifier voting weights. These trends, however, only appear to apply convincingly with the Naïve Bayes classifier; the sigmoid parameter does not have as strong effect on other base classifiers, such as the SVM (Figure 6.111-Figure 6.114).

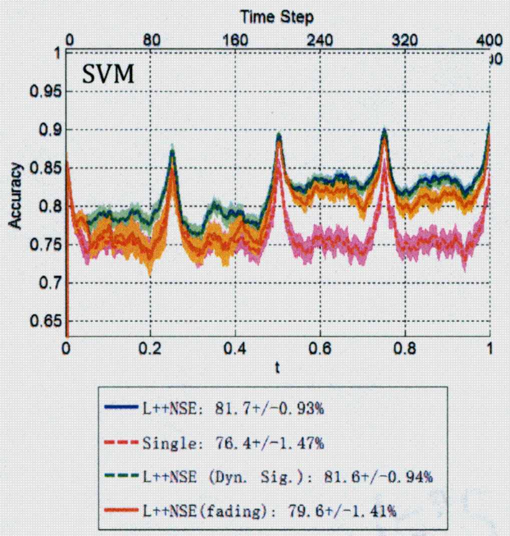


Figure 6.111: Scaffolding comparison for checkerboard data (constant drift) using SVM

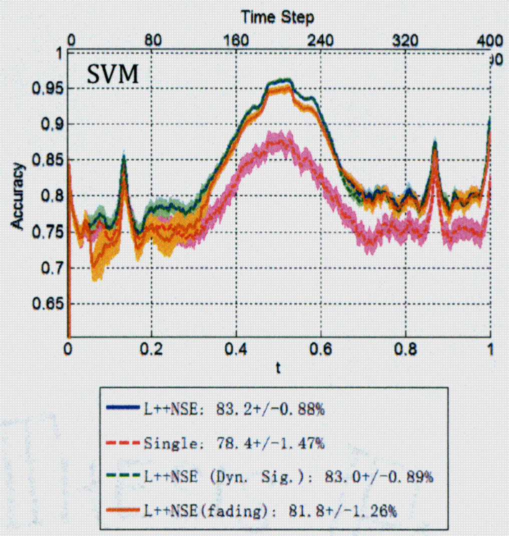


Figure 6.112: Scaffolding comparison for checkerboard data (sinusoidal drift) using SVM

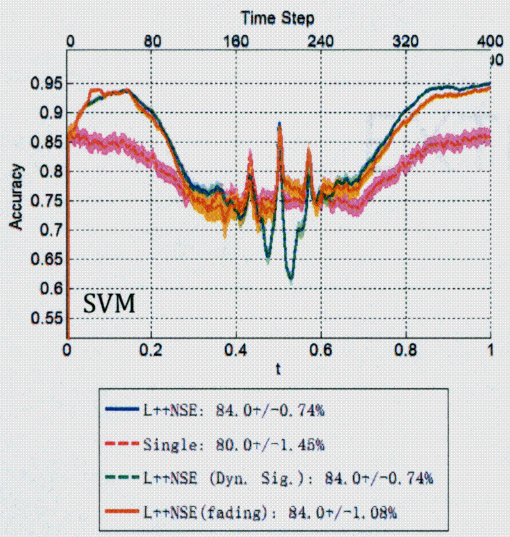


Figure 6.113: Scaffolding comparison for checkerboard data (pulsing drift) using SVM

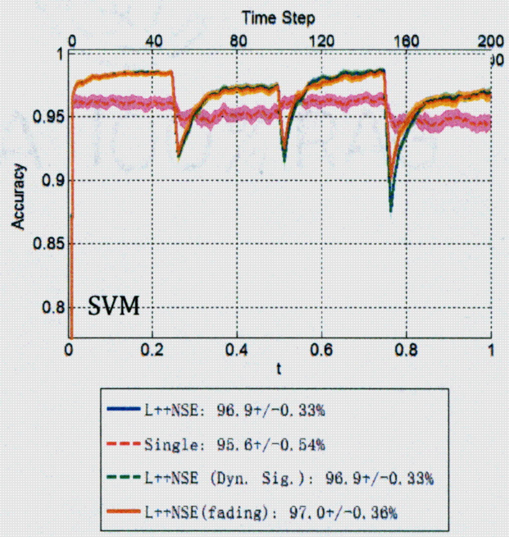


Figure 6.114: Scaffolding comparison for SEA concepts using SVM

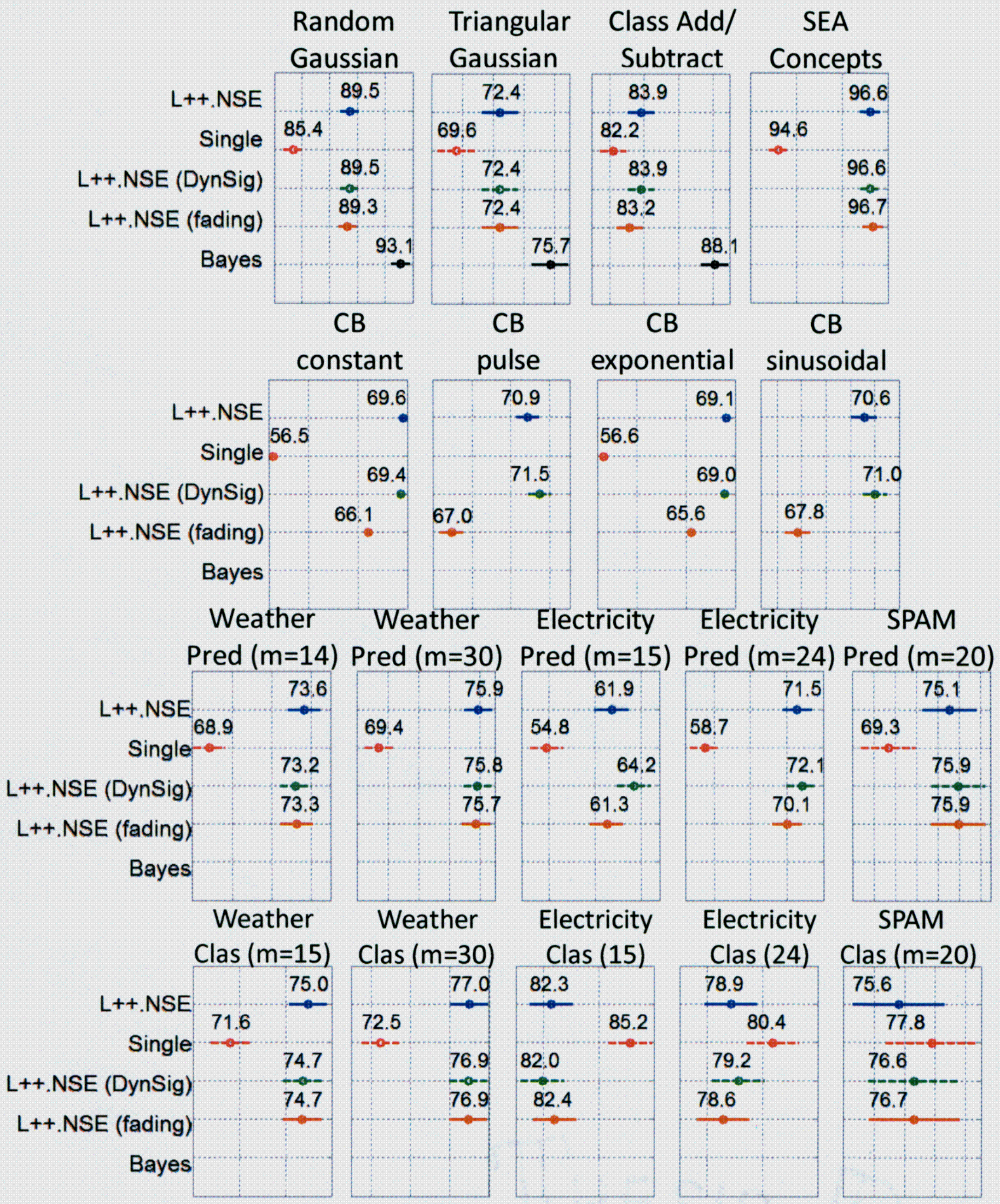


Figure 6.115: ANOVA comparison for scaffolding techniques using Naïve Bayes classifier

Fading, which only learns for a short time after drift is detected, offers good recovery during periods of sharp drift (checkerboard with pulsing drift in Figure 6.108) or concept change (SEA data in Figure 6.106). This rapid recovery alludes to the stability-plasticity dilemma, where the fading ensemble suffers less from the effect of old irrelevant knowledge. However, recovery in sharp drift is merely one side of the tradeoff, for *fading* has an overall negative effect on the ability to acquire knowledge compared to Learn^{++}.NSE. The effect is most noticeable in the Naïve Bayes experiments and is less significant when using SVM as the base classifier. SVM and other descriptive models are able to better represent complex data compared to Naïve Bayes; thus, SVM is able to perform better with limited knowledge. In general, the under-performance of *fading*, regardless of base classifier, leads us to conclude that there is little redundant knowledge acquired during the training of the Learn^{++}.NSE ensemble. As a final note, *fading* boasts one important feature that can be claimed by no other ensemble-limiting algorithm (e.g. pruning): that is, the ability to store and recall old knowledge. This is especially seen the checkerboard experiments using SVM (Figure 6.111-Figure 6.113). For this reason, *fading* offers an excellent alternative to permanent pruning techniques in the effort to reduce the size of memory.

Both *dynamic sigmoid adjustment* and *fading* techniques are highly dependent on the drift detection mechanism being used. Although the inconsistencies (described in Section 6.11) in the drift detection mechanism appear to have a minimal effect on performance, let us note some examples that reflect the importance of accurate drift detection. False negatives (drift not detected when it has actually occurred) are particularly detrimental, often the result of high configuration variance and a corresponding large confidence interval. In such a case, the *fading* model ceases to learn even though a new environment has been encountered – this is evident in the second SEA concept shift in Figure 6.106 (Naïve Bayes) and Figure 6.114 (SVM), as the ensemble converges to a low

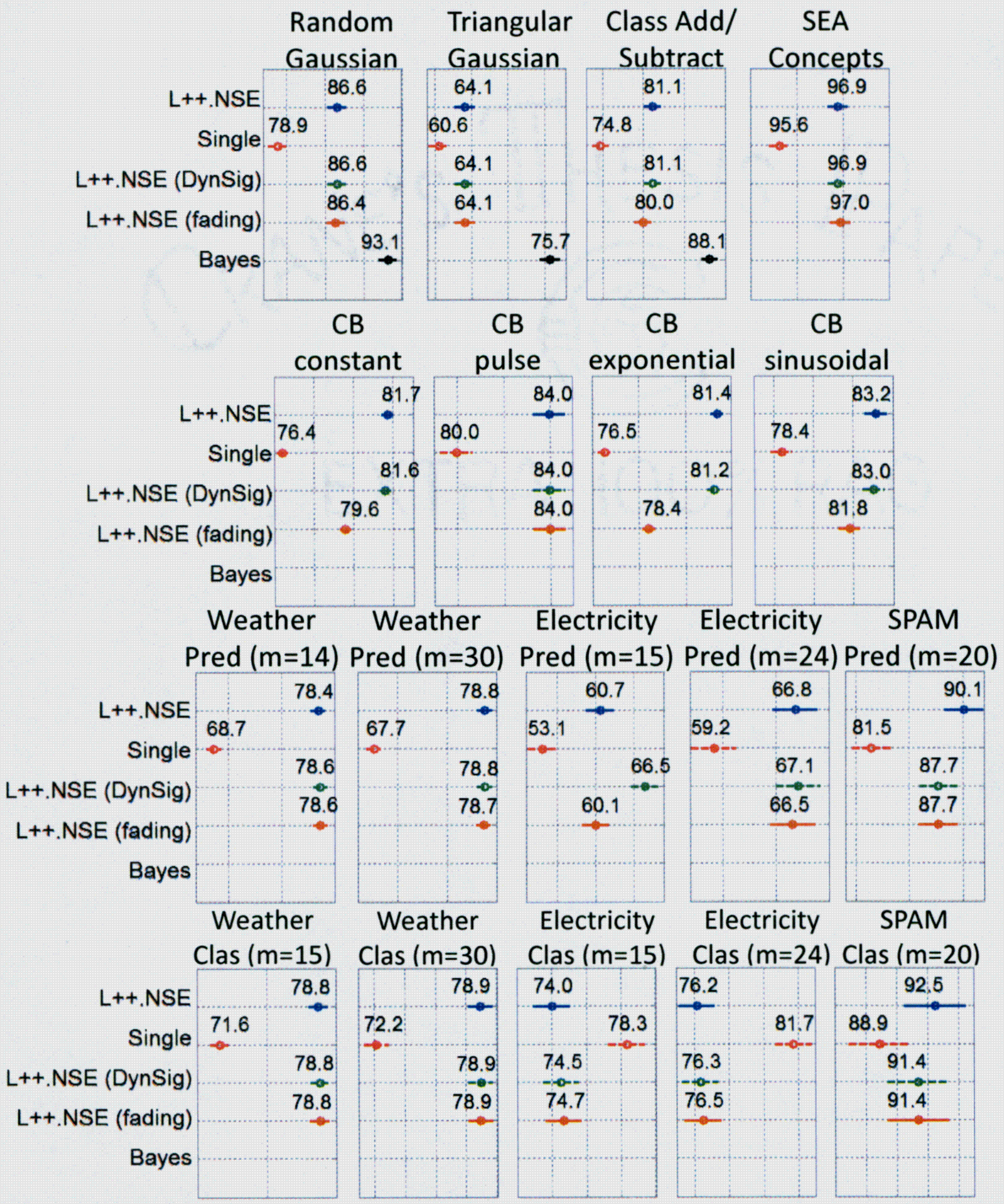


Figure 6.116: ANOVA comparison for scaffolding techniques using SVM classifier

performance value on the third concept because it has not learned it. In the *dynamic sigmoid* model, a false negative detection results in an increase of the cutoff parameter and causes less-than-optimal weighting for the ensemble, and is indicated by a slight lag in recovering to the third SEA concept.

CHAPTER 7: CONCLUSIONS

The primary goal of this work was to find the best approach to constructing and organizing knowledge for classification using a computer framework that incorporates an ensemble of classifiers as the knowledge base. The rationale, guidelines, and constraints of our computational approach are guided significantly by the rudimentary principles in *human* cognition and learning theory, developed primarily by Piaget and Vygotsky.

We have developed an algorithm which takes a batch-based approach to incremental learning that can operate independent of the base classifier selected. Classifiers are evaluated and categorized in a unique way according to their relevance in the current learning environment using a penalty weight such that the basis of evaluation is specifically those points which were previously unlearned. Furthermore, a time-weighted sigmoid is imposed on each classifier's error, so that each classifier will be weighted only on its most recent error (pertaining to the current environment).

7.1 CONTRIBUTIONS OF THIS WORK

The Learn^{++}.NSE algorithm is designed to accommodate the guidelines which have been suggested from the theories of computational *and* human learning. Specifically, we address how human cognition principles correlate to the Learn^{++}.NSE computational model. The homology of human learning terminologies and specific features of the Learn^{++}.NSE algorithm are diagramed in Figure 7.1.

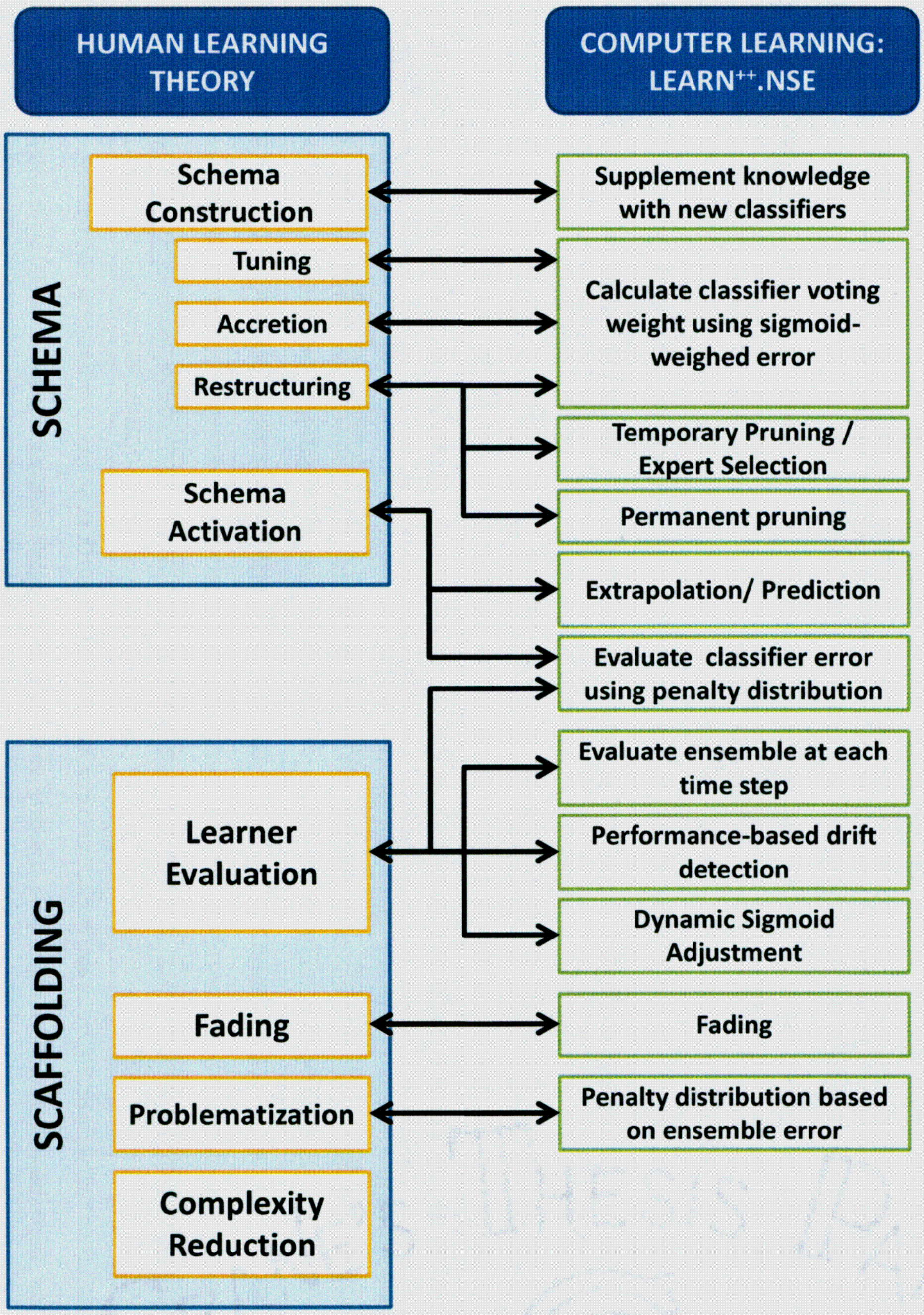

HUMAN LEARNING THEORY
COMPUTER LEARNING: LEARN++.NSE
SCHEMA
Schema Construction
Tuning
Accretion
Restructuring
Schema Activation
Supplement knowledge with new classifiers
Calculate classifier voting weight using sigmoid-weighed error
Temporary Pruning / Expert Selection
Permanent pruning
Extrapolation/ Prediction
Evaluate classifier error using penalty distribution
SCAFFOLDING
Learner Evaluation
Fading
Problematization
Complexity Reduction
Evaluate ensemble at each time step
Performance-based drift detection
Dynamic Sigmoid Adjustment
Fading
Penalty distribution based on ensemble error

Figure 7.1: Homology of supervised human learning and Learn^{++}.NSE computational learning

In addition to evaluating the algorithm itself, our work has introduced several variations to the model for the purpose of proving or *improving* the algorithm's effectiveness. These variations include the following:

1. Alternate weighting methods – Adaboost weighting is a similar form of the proposed method in Learn^{++}.NSE, yet it includes neither the penalty weight for calculating classifier error nor the time-weighted sigmoid over classifier error for calculating voting weights. The weighting scheme proposed in the Adaptive Classifiers Ensemble (ACE) algorithm extends Adaboost weighting with an expert selection method. Finally, the use of simple majority allows us to determine whether or not weighting is desirable in the first place.
2. Permanent ensemble pruning – the algorithm is augmented with three separate controlled forgetting scenarios to impose a limit on the size of the knowledge base. Classifiers are discarded with respect to their *age*, *current error*, or *average error*.
3. Temporary ensemble pruning (expert selection) – the number of classifiers used for a final ensemble vote is limited by imposing a performance cutoff (using a confidence interval about the best performer) or by ignoring all members with weights that do not lie in a top percent range. Classifiers are selected with replacement and may be used at any future time step.
4. Drift Detection – a performance-based detection mechanism is derived from the CUSUM quality control test. The test requires few parameters and is self-configuring, making it adaptable to a variety of drift scenarios without tweaking. Performance of a single batch classifier is monitored over time for significant

changes in the log-likelihood between a null hypothesis and two alternative hypotheses (upper and lower confidence interval bound).

5. Dynamic Sigmoid Adjustment – this heuristic approach alters parameters in the time-weighted sigmoid which can be optimized for peak performance in certain drift situations.
6. Fading – derived from scaffolding theory, this approach operates on the basis that the learner should acquire knowledge only when a new environment is seen. Thus, it works in tandem with the drift detection mechanism to only create classifiers in periods after drift is detected.

Each of these variations, along with comparisons with other incremental learning algorithms, are tested using a wide scope of synthetic and real-world experiments which incorporate an extensive range of drift situations.

7.2 Summary of Experimental Findings

The Learn^{++}.NSE algorithm boasts the following advantages over alternatives. With regard to weighting, Learn^{++}.NSE maintains a consistent high performance compared to approaches taken by Adaboost, Adaptive Classifiers Ensemble (ACE), and an unweighted ensemble. Additionally, Learn^{++}.NSE consistently and significantly outperforms a single classifier in nearly all experiments, providing convincing evidence in support of using an ensemble-based learning system.

There is an evident tradeoff between decreasing the knowledge base (using controlled forgetting/ensemble pruning), and maintaining a maximum level of performance, especially in the ability to recognize recurring environments. Another apparent tradeoff lies between attaining high convergence in a stationary environment and reacting quickly in the midst of severe drift or concept change. These are especially supported by the results seen in the *controlled forgetting* and *fading* experiments. These results clearly support the notion that decreasing the size of the knowledge base (classifier ensemble) will often result in some performance degradation, with the exception of severe drift and concept change.

The Streaming Ensemble Algorithm (SEA) is implemented as an additional comparison. In a number of synthetic experiments, SEA yields exceptional performance, exceeding all other algorithms; yet, this only occurs when the ensemble size is very small. SEA yields inferior classification results (even below single classifier performance) when the ensemble size is increased; this poor performance can be attributed to the use of simple majority vote. The requirement for a small ensemble size is extremely limiting to the SEA algorithm, as it is unable to store prior knowledge for potentially recurring environments.

An online learning comparison is introduced using an implementation of the Dynamic Weighted Majority (DWM) algorithm. DWM is seemingly effective for keeping up-to-date with the current environment (good plasticity) as the ensemble pruning rule consistently maintains a small ensemble size. The downfall of the algorithm is that the update rule for adding and removing knowledge effectively nullifies the ability to handle recurring data (poor stability). Another limitation is the base model for classification; DWM is limited to online learners such as Naïve Bayes or decision trees, which tend to be inferior in classifying complex, non-Gaussian datasets.

The parameter-based drift detection mechanism offers a good alternative to detecting drift using either data distribution estimation (often complex and/or inconsistent) or classifier architecture (too ambiguous). Being consistent with the guidelines of scaffolding theory, the drift detector is able to, with good consistency, track changes in an environment using the performance of a single classifier. One weakness that arises is the inability to differentiate between moderate drift and severe drift.

Scaffolding approaches (*dynamic sigmoid adjustment* and *fading*) proved to be useful developments, although neither provided a striking improvement over the basic Learn^{++}.NSE model. *Dynamic sigmoid* adjustment provides significant improvements with the ability to heuristically improve classifier categorization; however, its effectiveness appears to be limited to the Naïve Bayes classifier. Parameter characterization with other base classifier models shows insufficient differentiability across the scope of drift situations. *Fading* improves the learning model by increasing performance during severe drift and concept change (good plasticity), yet overall there is a decrease in performance (inferior stability), although not always statistically significant, especially when descriptive models are used. On one hand, we see that, in general, continual learning is preferable. However, in considering how to *best* reduce the size of the knowledge base, ensemble *fading* is superior to all other methods in that it is the only approach that allows *retention of prior knowledge.* The inability to do so is inherent in the other approaches that were tested (Learn^{++}.NSE with pruning, Streaming Ensemble Algorithm, and Dynamic Weighted Majority).

In conclusion, Learn^{++}.NSE strikes a good balance in the stability-plasticity dilemma, evident by its consistent behavior across all experiments, and the ability to recall old knowledge effectively. In cases of severe drift or concept change, we see that it leans slightly to the side of stability; however, the scaffolding (drift detection) and/or pruning techniques are effective in mitigating these effects.

7.3 RECOMMENDATIONS FOR FUTURE WORK

Throughout this study, we investigate a combination of many heuristic approaches. Finding the *optimal* algorithm which will handle all drift scenarios appears to be an unreasonable goal. This is best known as the "no free lunch" theorem. However, there are some measures that can be taken to bring ensemble systems, specifically Learn^{++}.NSE, closer to the ideal functionality.

While heuristics are often effective and even necessary, it would be desirable to take a new look at the Learn^{++}.NSE algorithm from an optimality standpoint. The study reveals specific scenarios where characteristics (such as pruning, or the sigmoid parameters) perform better at one time, and worse at another. It is simple to *observe* these trends, but it is much more difficult to *apply* these characteristics on-the-fly by relating them to drift in some way. This requires more than a drift-detection mechanism that provides a binary response ("drift" or "no drift"). Rather, the *quantity* and *rate* of drift are desired in order to properly adjust the learner's architecture.

It is always difficult to know how much experimentation is enough, as there is an infinite number of scenarios that can be tested, both synthetic and in the real-world. The ensemble of datasets presented in this work are quite comprehensive; yet, they could benefit from additional scenarios, namely 1) multi-class real-world datasets, especially those with class addition and removal, 2) synthetic datasets involving other non-Gaussian distributions, 3) combining concept change with recurring environments, and 4) an learning in the presence of unbalanced data.

Other analyses which were not explicitly covered in this work, such as noise/random drift tolerance, window size sensitivity, and the use of decision tree models for classification would be beneficial for the sake of completeness.

REFERENCES

[1] S. Grossberg, "Nonlinear neural networks: principles, mechanisms, and architectures," *Neural Networks*, vol. 1, no. 1, pp. 17-61, 1988.

[2] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, "Handling local concept drift with dynamic integration of classifiers: domain of antibiotic resistance in nosocomial infections," in *19th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2006)* 2006, pp. 679-684.

[3] M. Muhlbaier, "Boosted ensemble algorithm strategically trained for incremental learning of unbalanced data." M.S. thesis, Rowan University,Glassboro, NJ, 2006.

[4] M. Muhlbaier, A. Topalis, and R. Polikar, "Learn^{++}.NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 152-168, 2009.

[5] J. Gao, W. Fan, and J. Han, "On appropriate assumptions to mine data streams: analysis and practice," in *International Conference on Data Mining* 2007, pp. 143-152.

[6] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning*. Cambridge, Massachusetts: The MIT Press, 2009.

[7] F. L. Minku, A. White, and X. Yao, "The impact of diversity on on-line ensemble learning in the presence of concept drift," *IEEE Transactions on Knowledge and Data Engineering (accepted for future publication)*, 2009.

[8] L. I. Kuncheva, "Classifier ensembles for changing environments," in *Multiple Classifier Systems (MCS 2004),* 3077 ed 2004, pp. 1-15.

[9] L. I. Kuncheva, "Classifier ensembles for detecting concept change in streaming data: overview and perspectives," in *European Conference on Artificial Intelligence (ECAI)* 2008, pp. 5-10.

[10] J. Piaget, *Six Psychological Studies*. New York: Random House, Inc., 1967.

[11] M. H. Appel and L. S. Goldberg, *Equilibration: Theory, Research, and Application*. New York: Plenum Press, 1977.

[12] J. Flavell, "Piaget's legacy," *Psychological Science*, vol. 7, no. 4, pp. 200-203, 1996.

[13] Widmayer, Sharon, "Schema theory: an introduction," 2005. [Online]. Available: http://www2.yk.psu.edu/~jlg18/506/SchemaTheory.pdf. Accessed: 11-16-0009.

[14] B. Armbruster, "Schema theory and the design of content-area textbooks," *Educational Psychologist*, vol. 21, pp. 253-276, 1996.

[15] D. Miller, *Reading with meaning*. Portland Maine: Stenhouse Publishers, 2002.

[16] L. S. Vygotsky, *Mind and Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press, 1978.

[17] B. J. Reiser, "Scaffolding complex learning: the mechanisms of structuring and problematizing student work," *The Journal of the Learning Sciences*, vol. 13, no. 3, pp. 273-304, 2004.

[18] D. Wood and H. Wood, "Vygotsky, tutoring and learning," *Oxford Review of Education*, vol. 22, no. 1, pp. 5-16, 1996.

[19] D. Wood, "Scaffolding, contingent tutoring and computer-based learning," *International Journal of Artificial Intelligence in Education*, vol. 12, pp. 280-292, 2001.

[20] B. V. Dasarathy and B. V. Sheela, "Composite classifier system design: concepts and methodology," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 708-713, 1979.

[21] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993-1001, 1990.

[22] Polikar, R, "Ensemble based systems in decision making," *IEEE Circuits and Systems Magazine,* no. 3, 21-45, 2006.

[23] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197-227, June1990.

[24] L. I. Kuncheva, *Combining Pattern Classifiers, Methods and Algorithms*. New York, NY: Wiley Interscience, 2005.

[25] J. Kittler, M. Hatef, R. P. W. Duin, and J. Mates, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226-239, 1998.

[26] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," in *Foundations of Computer Science, 1989., 30th Annual Symposium on* 1989, pp. 256-261.

[27] N. Littlestone and M. Warmuth, "Weighted majority algorithm," *Information and Computation*, vol. 108, pp. 212-261, 1994.

[28] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift," in *International Conference on Machine Learning,* 119 ed 2005, pp. 449-456.

[29] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: an ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755-2790, 2007.

[30] H. Wang, W. Fan, P. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 2003, pp. 226-235.

[31] Y. Freund and R. E. Schapire, "Decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119-139, 1997.

[32] M. Kendall, *Rank Correlation Methods*, 4 ed. London: Griffin, 1975.

[33] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, "Dynamic integration of classifiers for handling concept drift," *Information Fusion*, vol. 9, no. 1, pp. 56-68, Jan.2008.

[34] H. S. Mohammed, J. Leander, M. Marbach, and R. Polikar, "Comparison of ensemble techniques for incremental learning of new concept classes under hostile non-stationary environments," in *Systems, Man and Cybernetics, 2006. ICSMC '06. IEEE International Conference on,* 6 ed 2006, pp. 4838-4844.

[35] K. Nishida, K. Nishida, and K. Yamauchi, "Adaptive classifiers-ensemble system for tracking concept drift," in *Machine Learning and Cybernetics, 2007 International Conference on,* 6 ed. K. Yamauchi, Ed. 2007, pp. 3607-3612.

[36] A. Dries and U. Ruckert, "Adaptive concept drift detection," in *SIAM International Conference on Data Mining* 2009, pp. 233-244.

[37] V. N. Vapnik, *Statistical Learning Theory.* New York: Wiley, 1998.

[38] C. Bishop, *Pattern Recognition and Machine Learning.* New York: Springer Science + Business Media LLC, 2006.

[39] M. Baena-Garcia, J. del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Bueno-Morales, "Early drift detection method," in *ECML PKDD Workshop on Knowledge Discovery from Data Streams* 2006, pp. 77-86.

[40] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: a new ensemble method for tracking concept drift," in *3rd IEEE Int. Conf. on Data Mining (ICDM 2003)* 2003, pp. 123-130.

[41] M. Lazarescu and S. Venkatesh, "Using selective memory to track concept drift effectively," in *Intelligent Systems and Control* 2003, pp. 14-20.

[42] A. Bifet and R. Gavalda, "Adaptive parameter-free learning from evolving data streams," Department de Llenguatges i Sistemes Informatics, Universitat Politecnica de Catalunya,LSI R09-9, 2009.

[43] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence - SBIA 2004,* 3171 ed 2004, pp. 286-295.

[44] C. Alippi and M. Roveri, "An adaptive CUSUM-based test for signal change detection," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on* 2006, p. 4.

[45] B. J. Oommen and L. Rueda, "Stochastic learning-based weak estimation of multinomial random variables and its applications to pattern recognition in non-stationary environments," *Pattern Recognition*, vol. 39, no. 3, pp. 328-341, Mar.2006.

[46] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers in non-stationary conditions," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on* 2007, pp. 1014-1019.

[47] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers; part I: detecting nonstationary changes," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1145-1153, 2008.

[48] G. Robertson, *Quality Through Statistical Thinking: Improving process control and capability*. Dearborn, Michigan: ASI Press, 1989.

[49] W. A. Shewhart, *Economic Control of Quality of Manufactured Product*. New York: Van Nostrand, 1931.

[50] S. W. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1 1959.

[51] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100-115, 1954.

[52] J. Lucas and R. Crosier, "Fast initial response for CUSUM quality-control schemes: give your CUSUM a head start," *Technometrics*, vol. 42, no. 1, pp. 102-107, 1982.

[53] J. M. Lucas, "Combined Shewhart-CUSUM quality control schemes," *Journal of Quality Technology*, vol. 8, pp. 1-12, 1982.

[54] J. M. Lucas, "Cumulative sum (CUSUM) control schemes," *Communications in Statistics - Theory & Methods*, vol. 14, no. 11, pp. 2689-2704, 1985.

[55] W. D. Ewan, "When and how to use cu-sum charts," *Technometrics*, vol. 5, no. 1, p. 22, 1963.

[56] R. MacNally and B. T. Hart, "Use of CUSUM methods for water-quality monitoring in storages," *Environmental Science Technology*, vol. 31, pp. 2114-2119, 1997.

[57] M. Reynolds and Z. Stoumbos, "A CUSUM chart for monitoring a proportion when inspecting continuously," *Journal of Quality Technology*, vol. 31, no. 1, pp. 87-108, 1999.

[58] B. Manly and D. Mackenzie, "A cumulative sum type of method for environmental monitoring," *Environmetrics*, vol. 11, pp. 151-166, 2000.

[59] G. V. Moustakides, "Decentralized CUSUM change detection," in *Information Fusion, 2006 9th International Conference on* 2006, pp. 1-6.

[60] C. Alippi and M. Roveri, "A computational intelligence-based criterion to detect non-stationarity trends," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on* 2006, pp. 5040-5044.

[61] G. A. Barnard, "Control charts and stochastic processes," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 21, no. 2, pp. 239-271, 1959.

[62] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, pp. 50-60, 1947.

[63] W. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583-621, 1952.

[64] F. Milton, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675-701, 1937.

[65] D. A. Cieslak and N. V. Chawla, "A framework for monitoring classifiers' performance: when and why failure occurs?," *Knowledge and Information Systems*, vol. 18, no. 1, pp. 83-109, 2009.

[66] G. Widmer and M. Kubat, "Effective learning in dynamic environments by explicit context tracking," in *Machine Learning, ECML93* 1993, pp. 227-243.

[67] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69-101, 1996.

[68] R. Klinkenberg and T. Joachims, "Detecting concept drift with support vector machines," in *17th International Conference on Machine Learning* 2000, pp. 487-494.

[69] N. C. Oza, "Online bagging and boosting," in *IEEE International Conference on Systems, Man and Cybernetics,* 3 ed Waikoloa, HI: 2005, pp. 2340-2345.

[70] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: an incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 31, no. 4, pp. 497-508, 2001.

[71] F. Minku and X. Yao, "Using diversity to handle concept drift in online learning," in *International Joint Conference on Neural Networks* 2009, pp. 2125-2132.

[72] R. Klinkenberg, "Learning drifting concepts: example selection vs. example weighting," *Intelligent Data Analysis, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, vol. 8, no. 3, pp. 281-300, 2004.

[73] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Seventh ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-01)* 2001, pp. 377-382.

[74] I. Katakis, G. Tsoumakas, and I. Vlahavas, "An ensemble of classifiers for coping with recurring contexts in data streams," in *European Conference on Artificial Intelligence,* 178 ed 2008, pp. 763-764.

[75] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: an application to email filtering," *Knowledge and Information Systems*, pp. 1-23, 2009.

[76] K. Nishida, K. Yamauchi, and O. Takashi, "ACE: Adaptive Classifiers-Ensemble system for concept-drifting environments," in *Multiple Classifier Systems,* 3541 ed. N. Oza, R. Polikar, J. Kittler, and F. Roli, Eds. 2005, pp. 176-185.

[77] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, "LEARN++: an incremental learning algorithm for multilayer perceptron networks," in *Acoustics, Speech, and Signal Processing, 2000.ICASSP '00.Proceedings.2000 IEEE International Conference on,* 6 ed 2000, pp. 3414-3417.

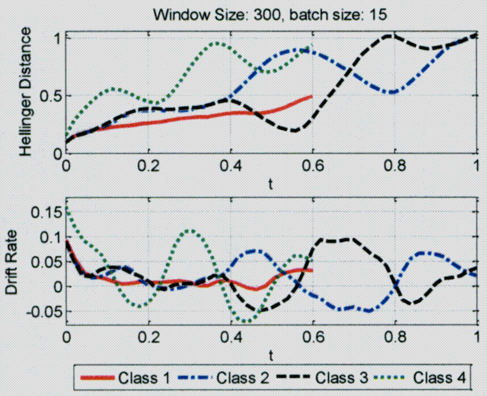
[78] Polikar, Robi, Advanced Topics in Patter Recognition. Class Lecture, Topic: "Learn++ & its applications: error weighting in Learn++.NSE." Department of Electrical & Computer Engineering, Rowan University, Glassboro, NJ, 2007.

[79] E. B. Wilson, "Probable Inference, the law of succession, and statistical inference," *Journal of the American Statistical Association*, vol. 22, no. 157, pp. 209-212, 1927.

[80] M. B. Harries, "SPLICE-2 comparative evaluation: electricity pricing," The University of New South Wales, Sydney, Australia,1999.

APPENDICES

APPENDIX A: HELLINGER DISTANCE FOR SYNTHETIC DATASETS



A.1: Class-based Hellinger distance for random Gaussian drift data with class addition/removal

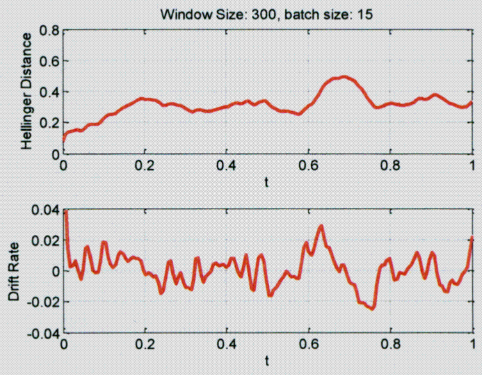


Figure A.2: Class independent Hellinger distance for random Gaussian drift data with class addition/removal

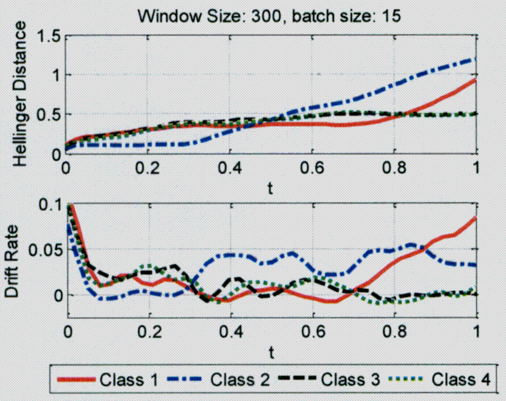


Figure A.3: Class dependent Hellinger distance for random Gaussian drift data

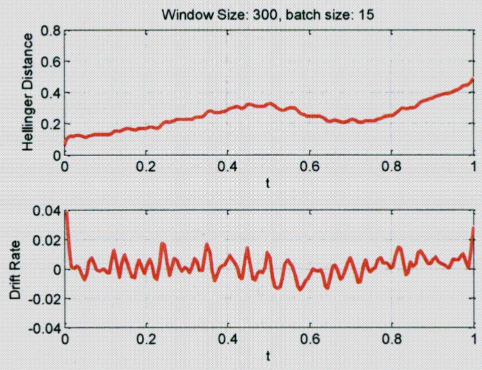


Figure A.4: Class-independent Hellinger distance for random Gaussian drift data

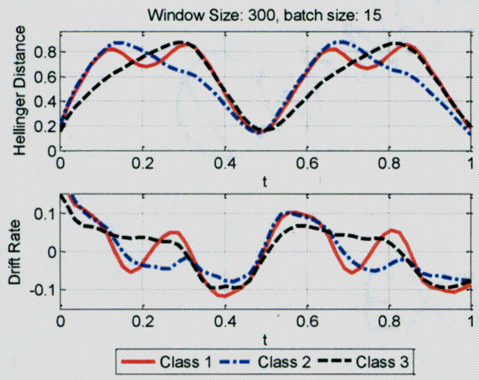


Figure A.5: Class-dependent Hellinger distance for triangular Gaussian drift data

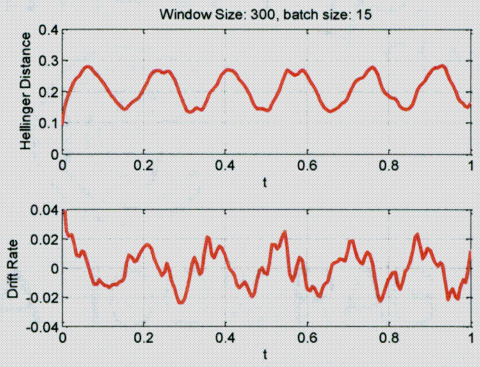


Figure A.6: Class-independent Hellinger distance for triangular Gaussian drift data

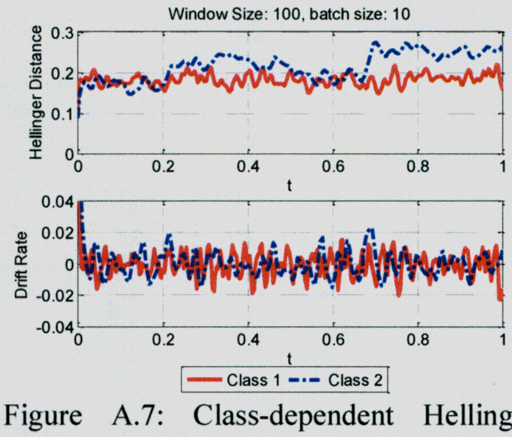


Figure A.7: Class-dependent Hellinger distance for SEA data

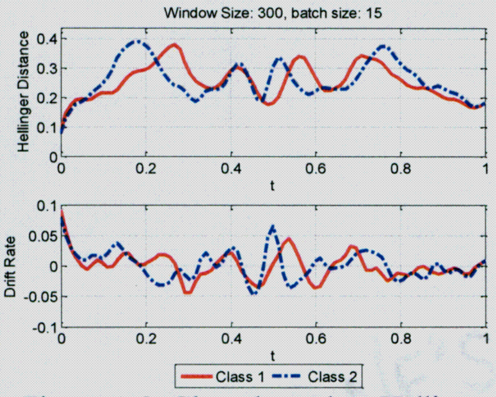


Figure A.8: Class-dependent Hellinger distance for checkerboard data (pulsing drift)

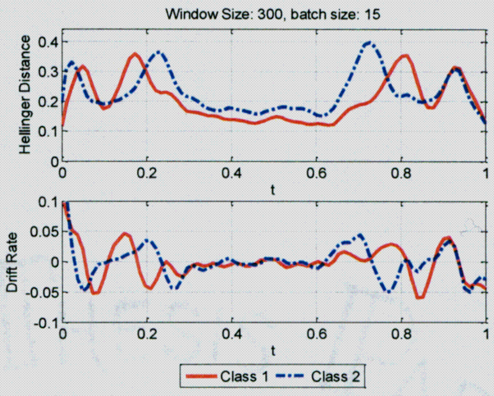


Figure A.9: Class-dependent Hellinger distance for checkerboard data (sinusoidal drift)

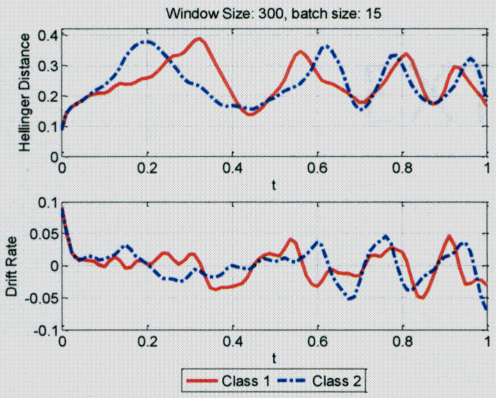


Figure A.10: Class-dependent Hellinger distance for checkerboard data (exponential drift)

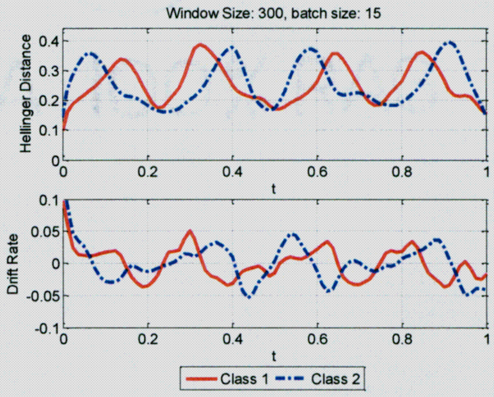


Figure A.11: Class-dependent Hellinger distance for checkerboard data (constant drift)

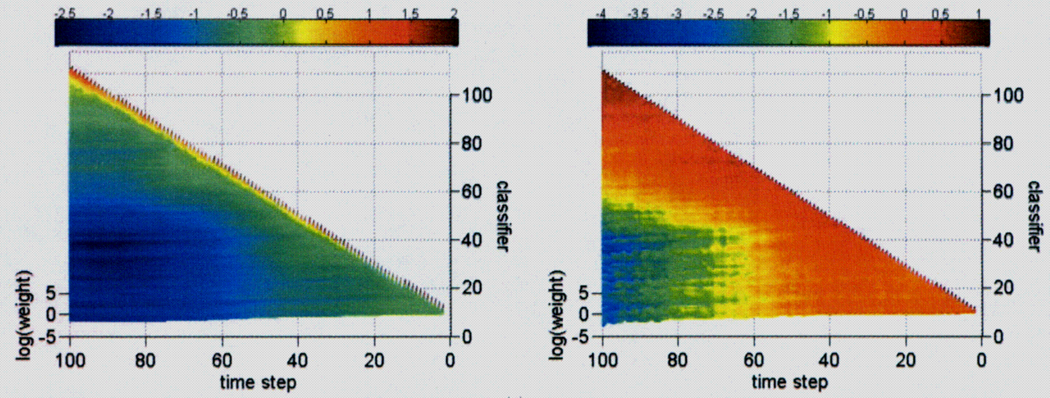


Figure B.1: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for random Gaussian drift data

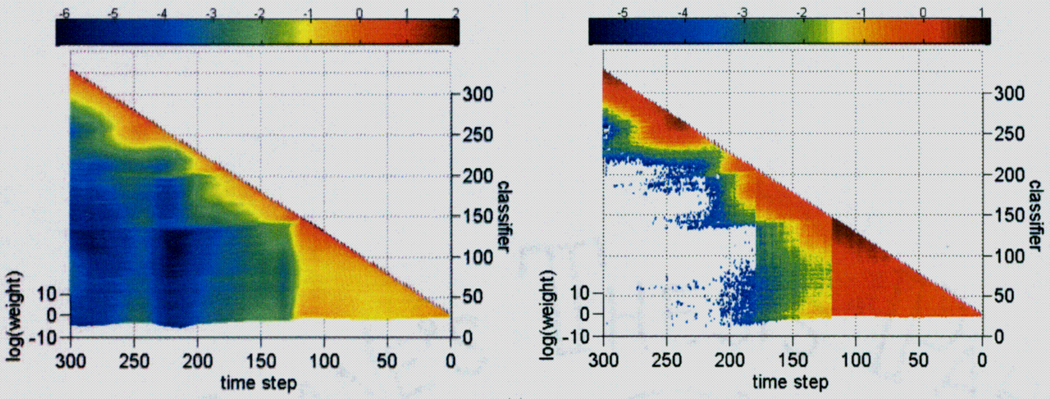


Figure B.2: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for random Gaussian drift data with class add/subtract

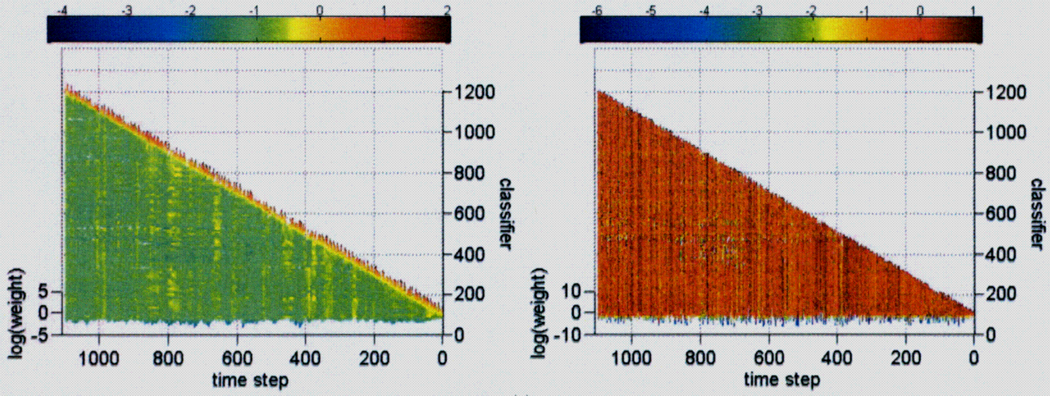


Figure B.3: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for electricity prediction data (training window size: 24)

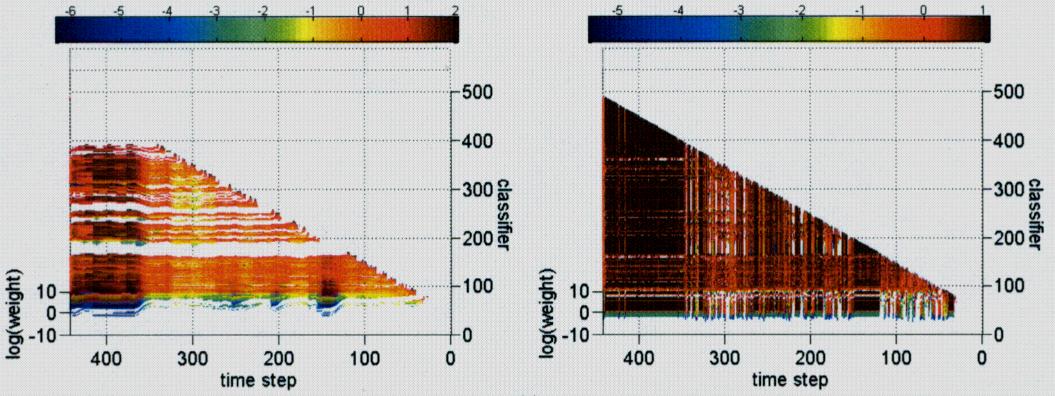


Figure B.4: Classifier weights for Learn^{++}.NSE (left) and Adaboost (right) ensemble weighting methods for spam prediction data (training window size: 20)

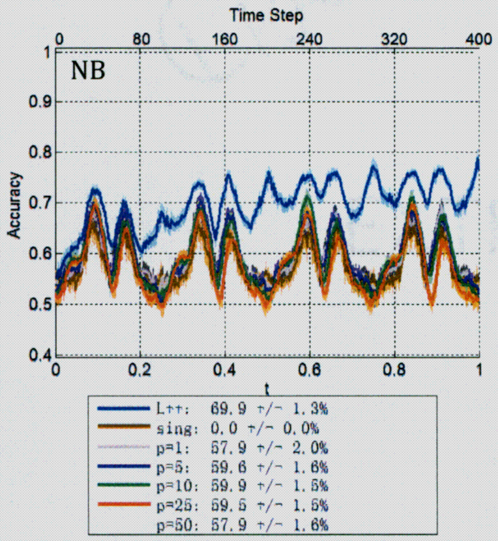


Figure C.1: DWM weight characterization, checkerboard data (constant drift)

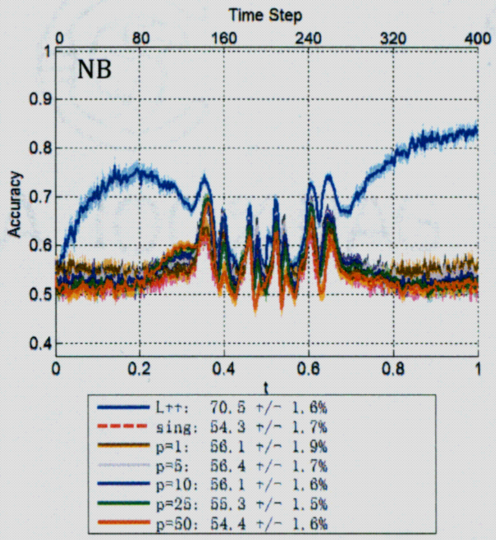


Figure C.2: DWM weight characterization, checkerboard data (pulsing drift)

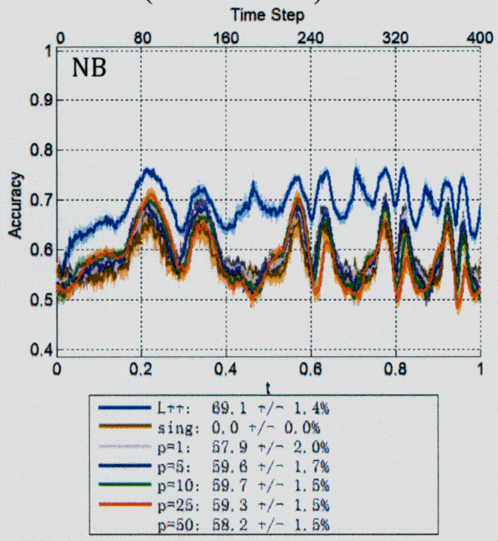


Figure C.3: DWM weight characterization, checkerboard data (exponential drift)

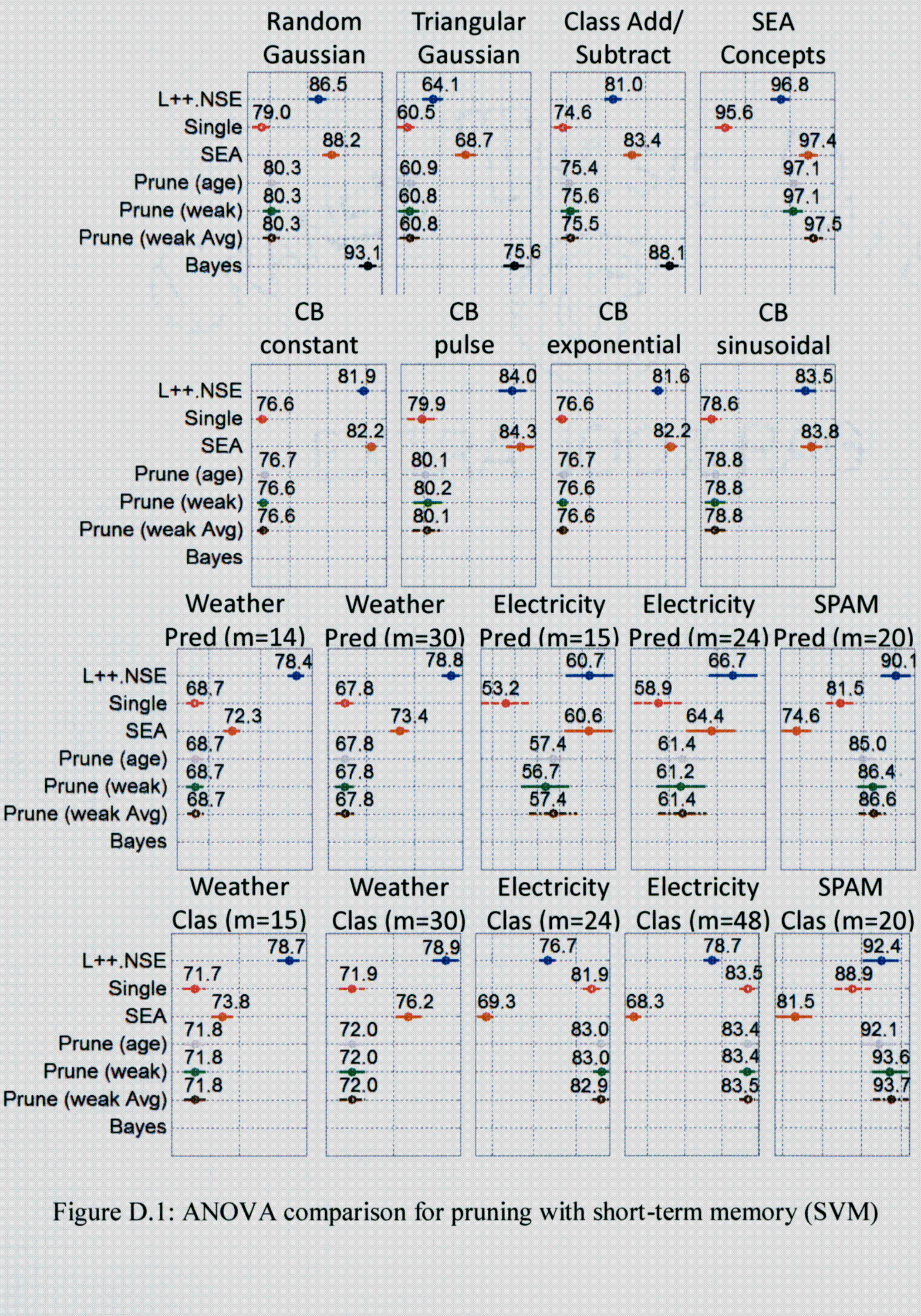


Figure D.1: ANOVA comparison for pruning with short-term memory (SVM)

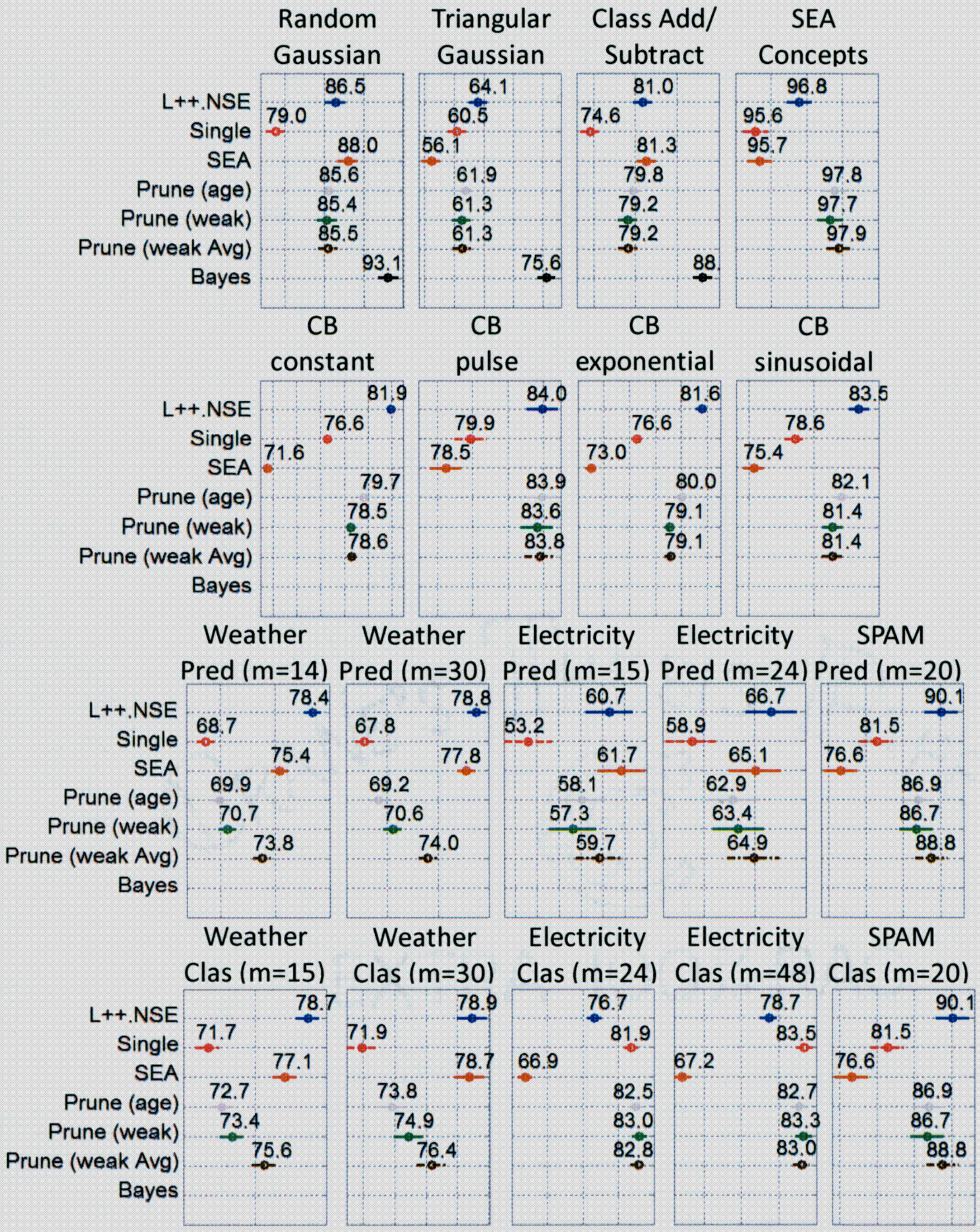


Figure D.2: ANOVA comparison for pruning with medium-term memory (SVM)

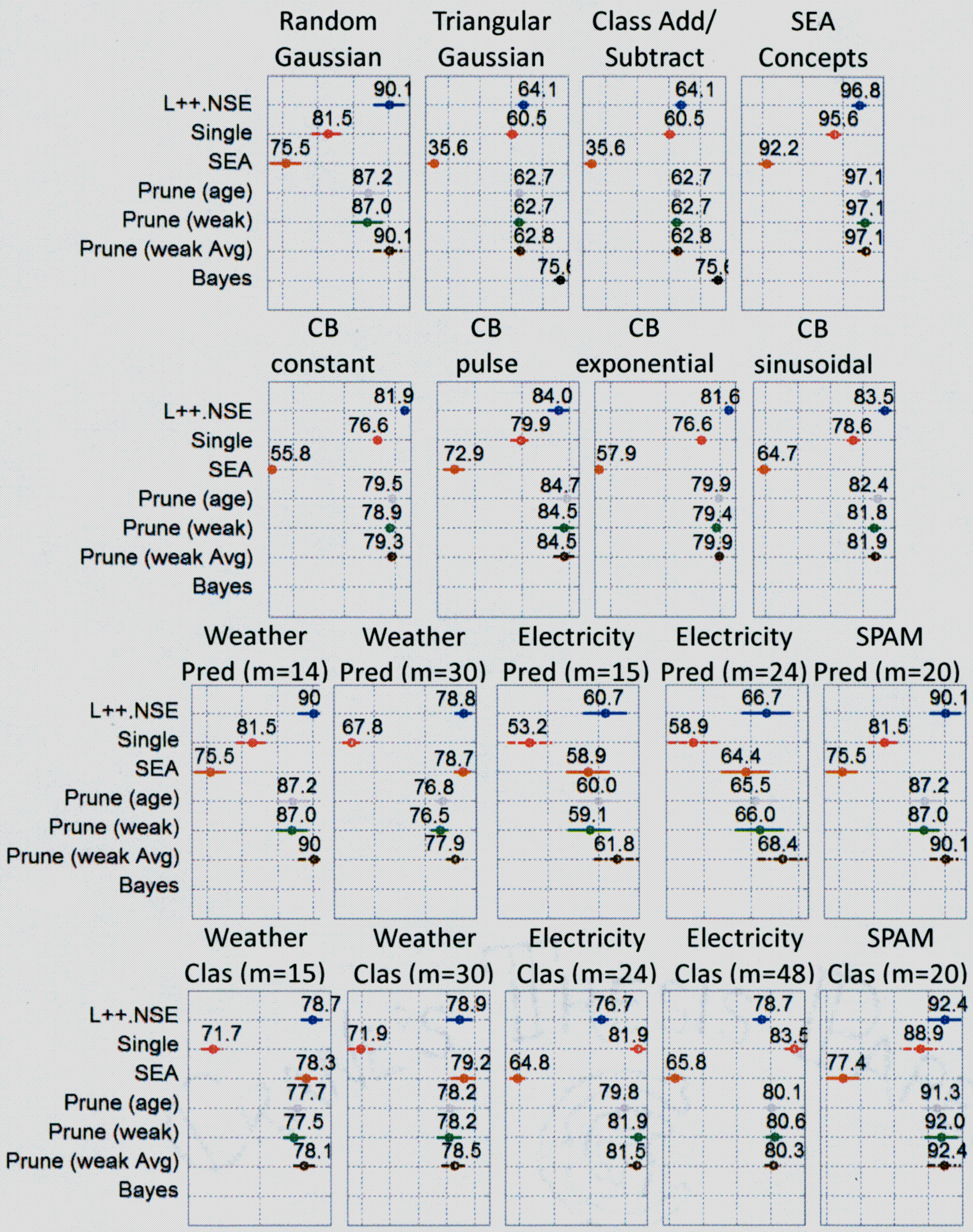


Figure D.3: ANOVA comparison for pruning with long-term memory (SVM)

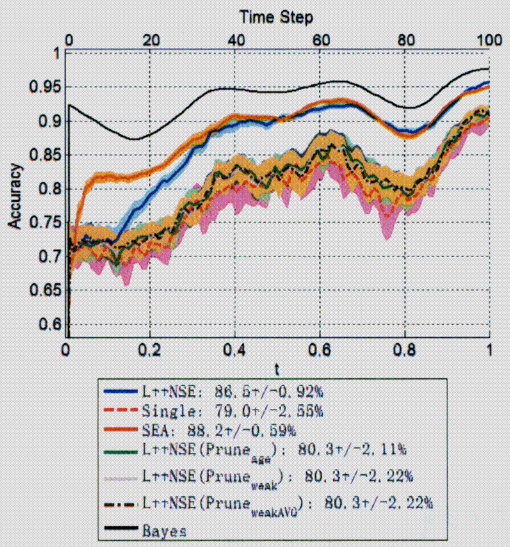


Figure D.4: Short-term memory (k=5) for random Gaussian drift data

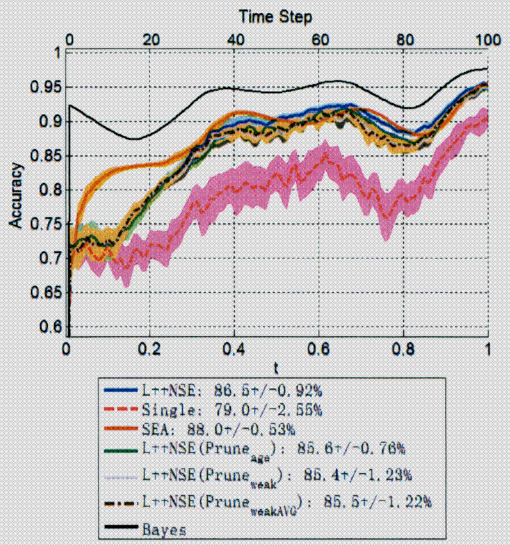


Figure D.5: Medium-term memory (k=25) for Gaussian drift data

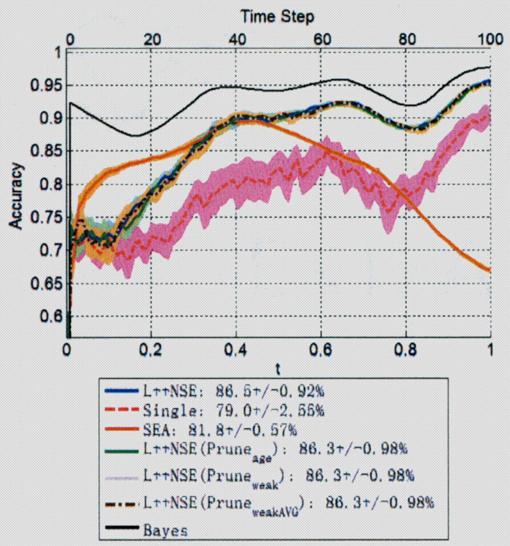


Figure D.6: Long-term memory (k=100) for Gaussian drift data

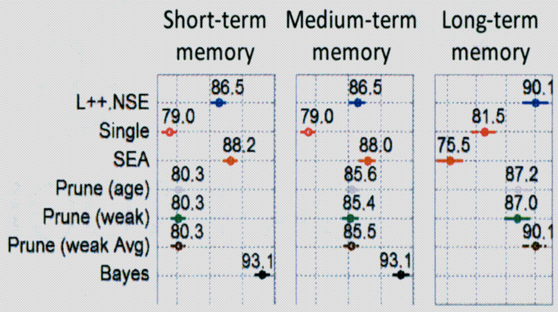


Figure D.7: ANOVA comparison for random Gaussian drift data with varying ensemble size (SVM)

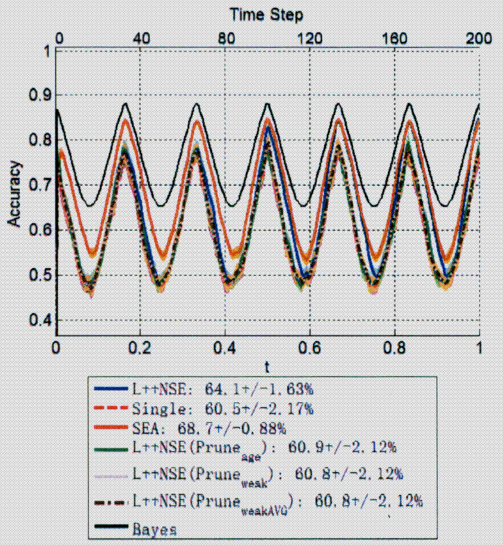


Figure D.8: Short-term memory (k=5) for triangular Gaussian drift data

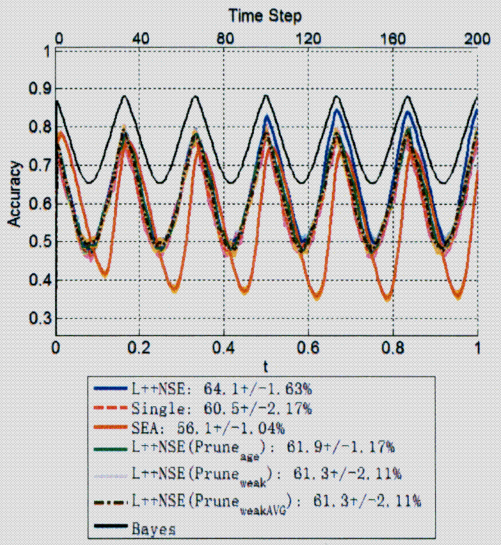


Figure D.9: Medium-term memory (k=25) for triangular Gaussian drift data

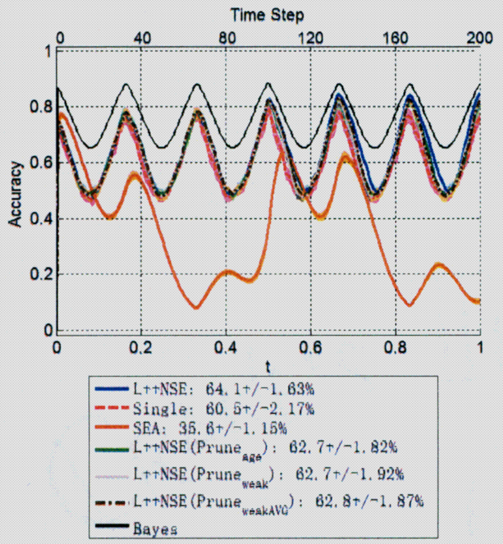


Figure D.10: Long-term memory (k=100) for triangular Gaussian drift data

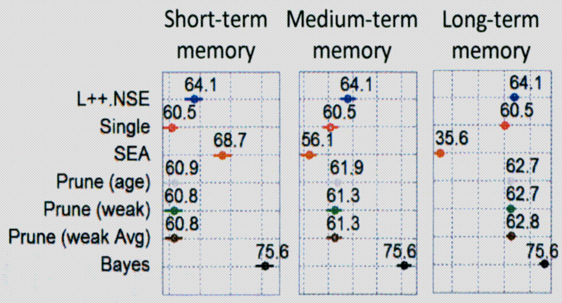


Figure D.11: ANOVA comparison for triangular Gaussian drift data with varying ensemble size (SVM)

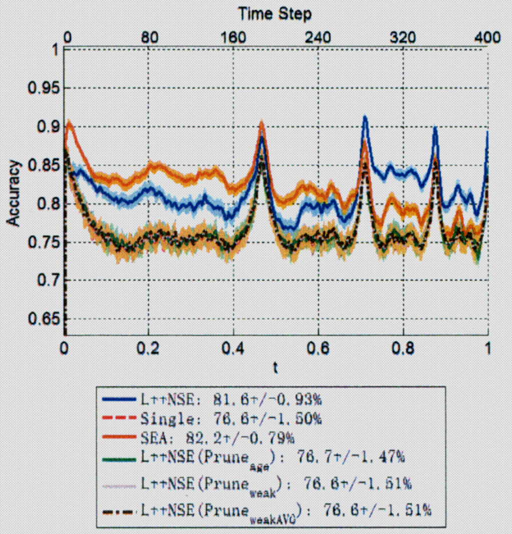


Figure D.12: Short-term memory (k=5) for checkerboard data (exponential drift)

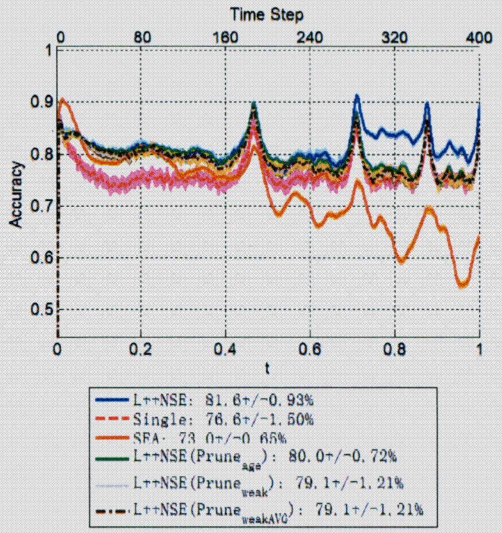


Figure D.13: Medium-term memory (k=25) for checkerboard data (exponential drift)

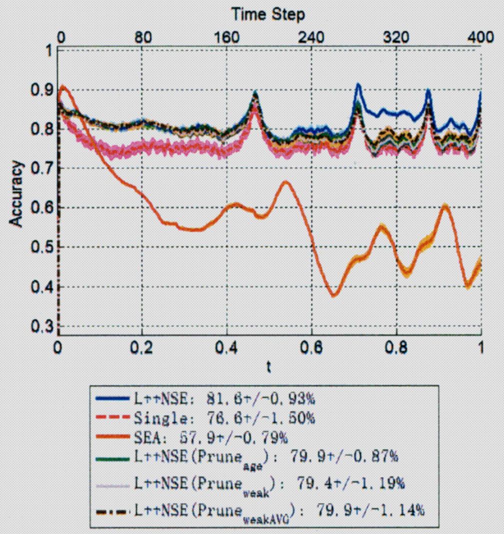


Figure D.14: Long-term memory (k=100) for checkerboard data (exponential drift)

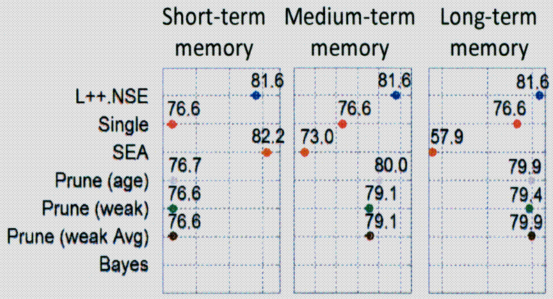


Figure D.15: ANOVA comparison for checkerboard data (exponential drift) with varying ensemble size (SVM)

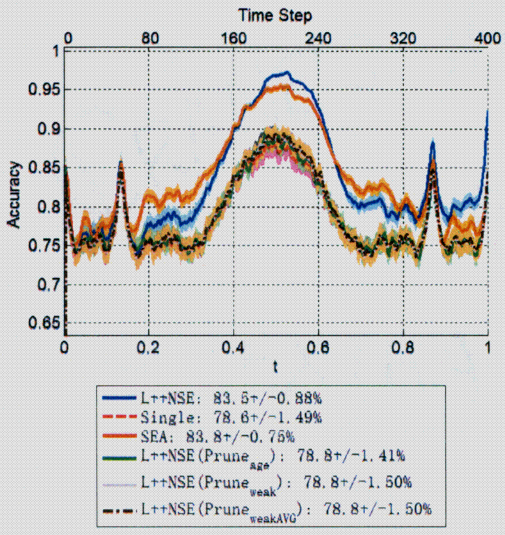


Figure D.16: Short-term memory (k=5) for checkerboard data (sinusoidal drift)

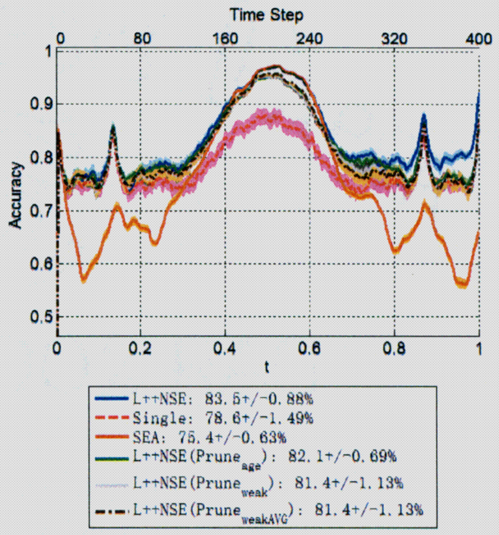


Figure D.17: Medium-term memory (k=25) for checkerboard data (sinusoidal drift)

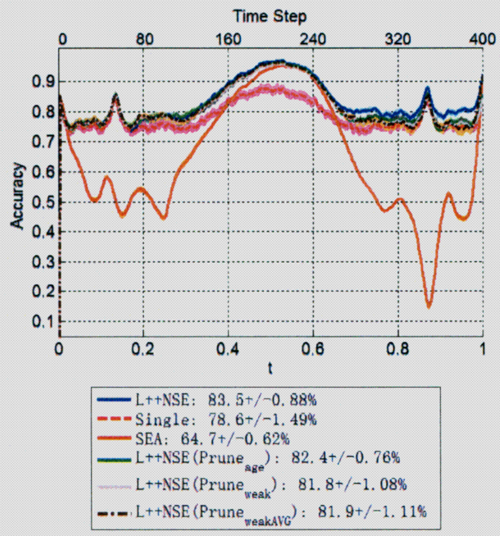


Figure D.18: Long-term memory (k=100) for checkerboard data (sinusoidal drift)

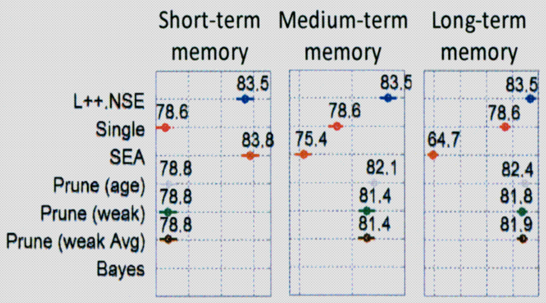


Figure D.19: ANOVA comparison for checkerboard data (sinusoidal drift) with varying ensemble size (SVM)

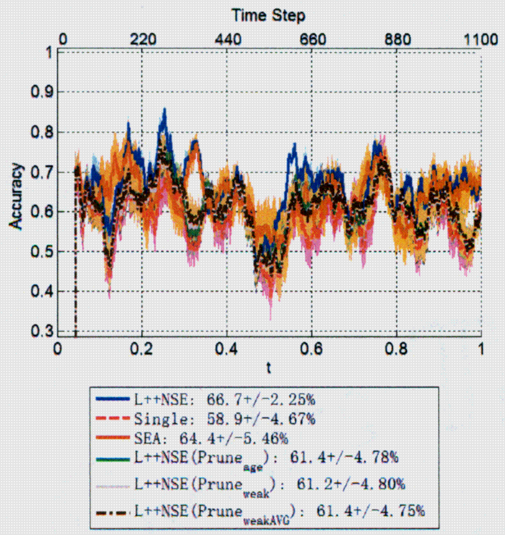


Figure D.20: Short-term memory (k=5) for electricity pricing prediction (m=24)

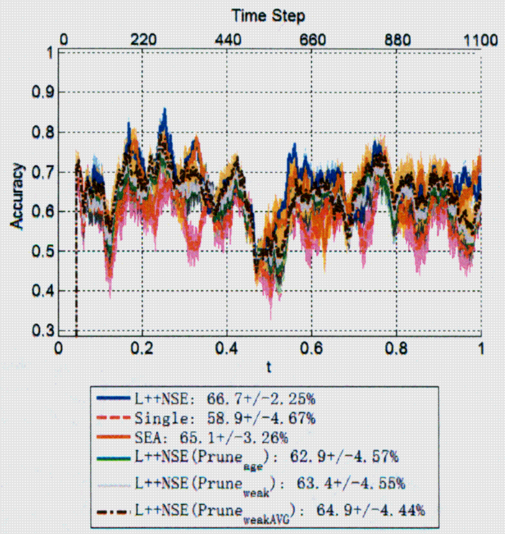


Figure D.21: Medium-term memory (k=25) for electricity pricing prediction (m=24)

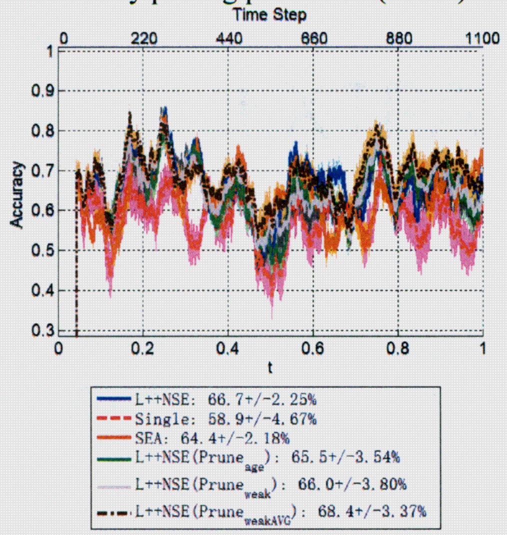


Figure D.22: : Long-term memory (k=100) for electricity pricing prediction (m=24)

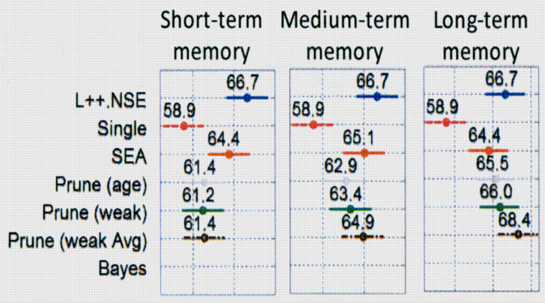


Figure D.23: ANOVA comparison for electricity pricing prediction (m=24) with varying ensemble size (SVM)

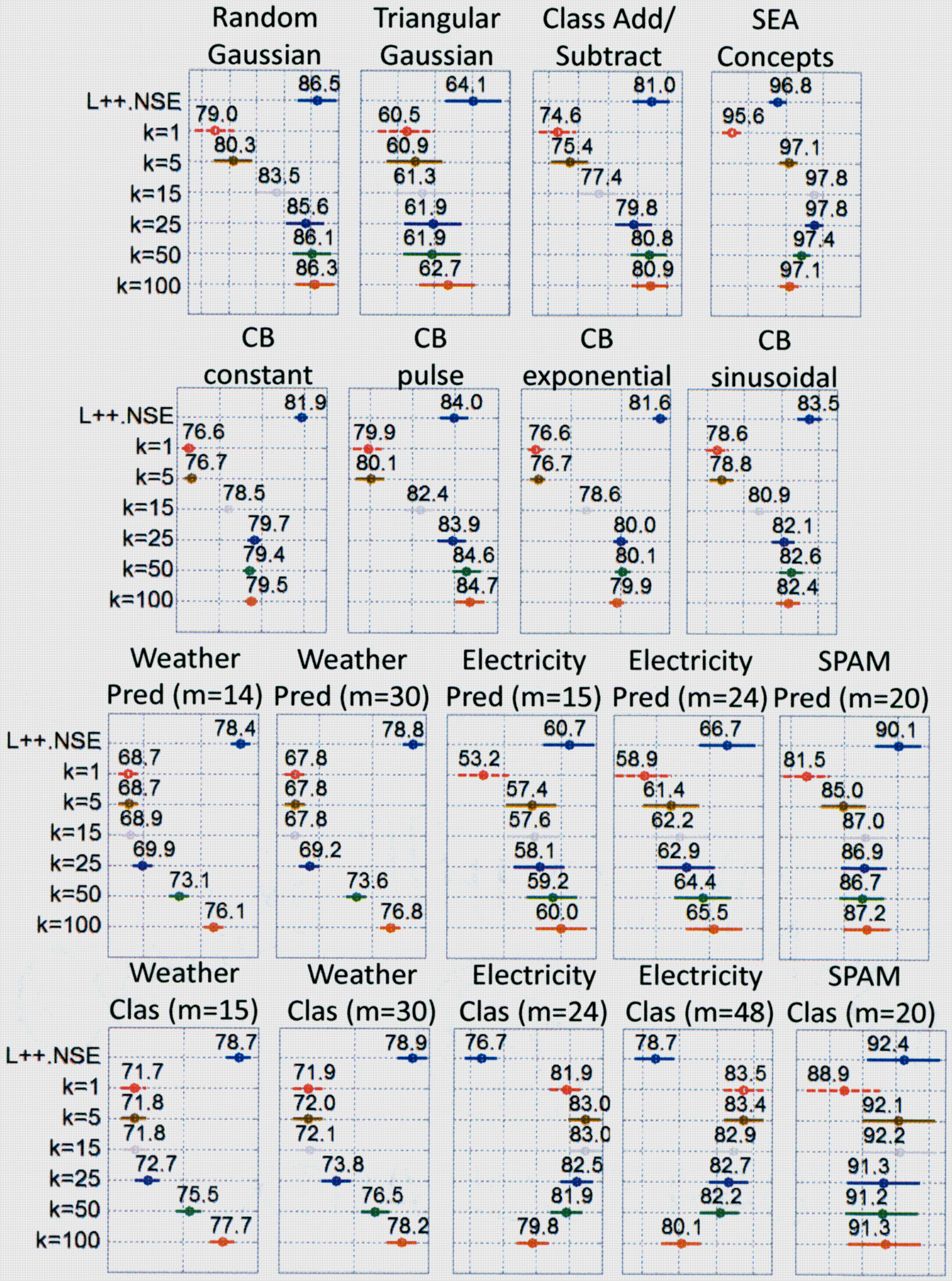


Figure D.24: ANOVA Comparison for age-based pruning characterization

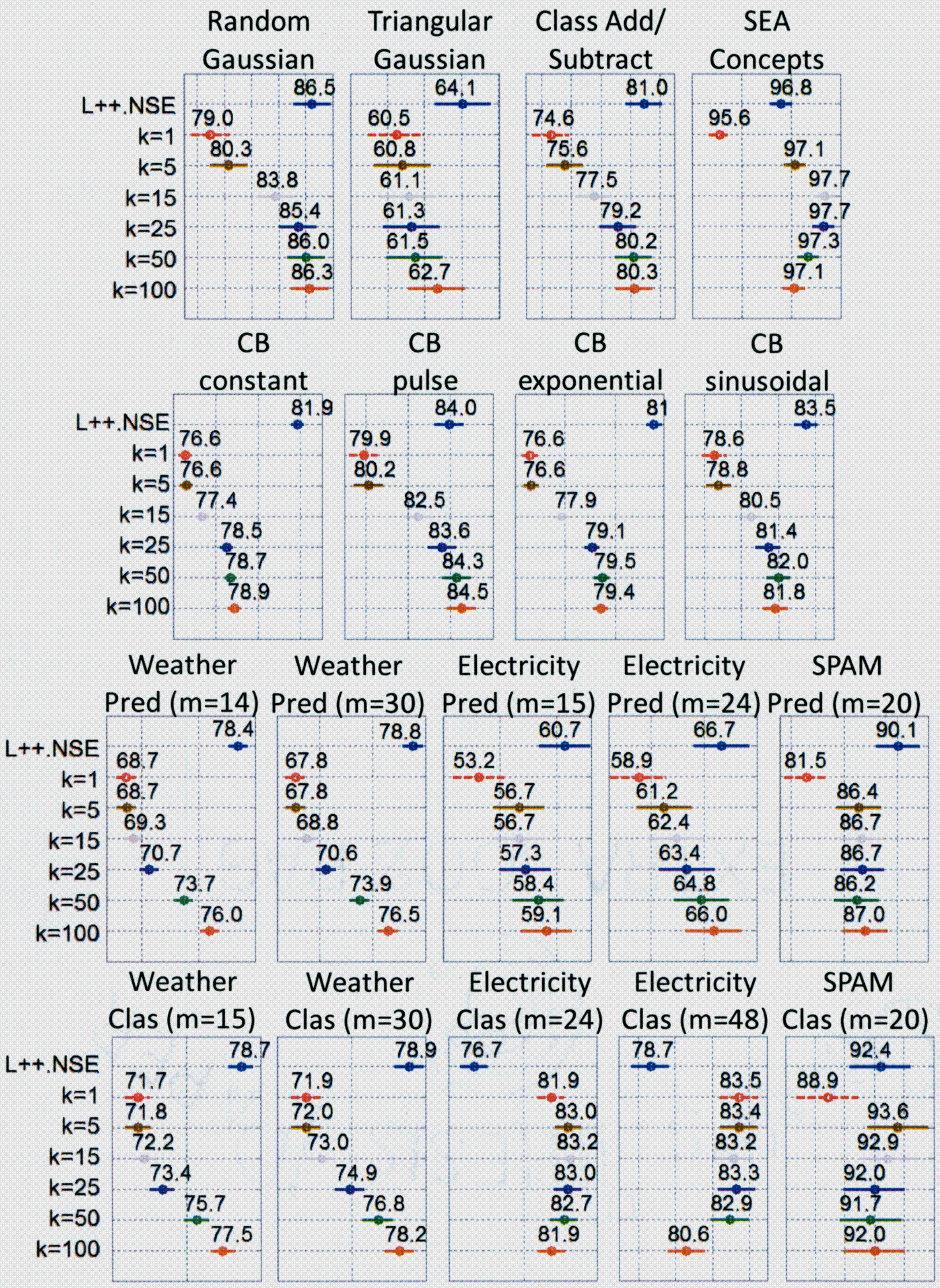


Figure D.25: ANOVA Comparison for error-based pruning characterization

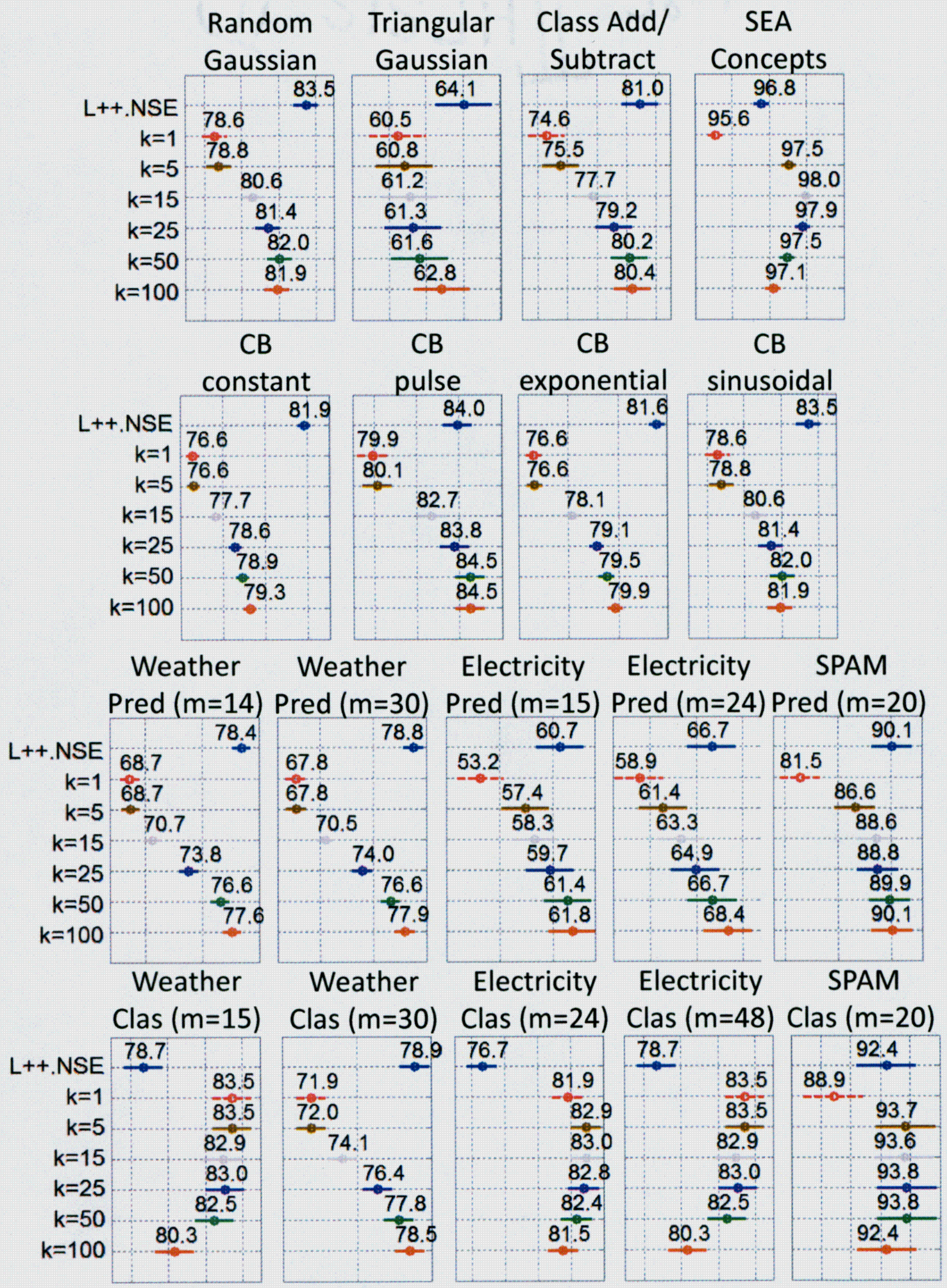


Figure D.26: ANOVA comparison for average error-based pruning characterization

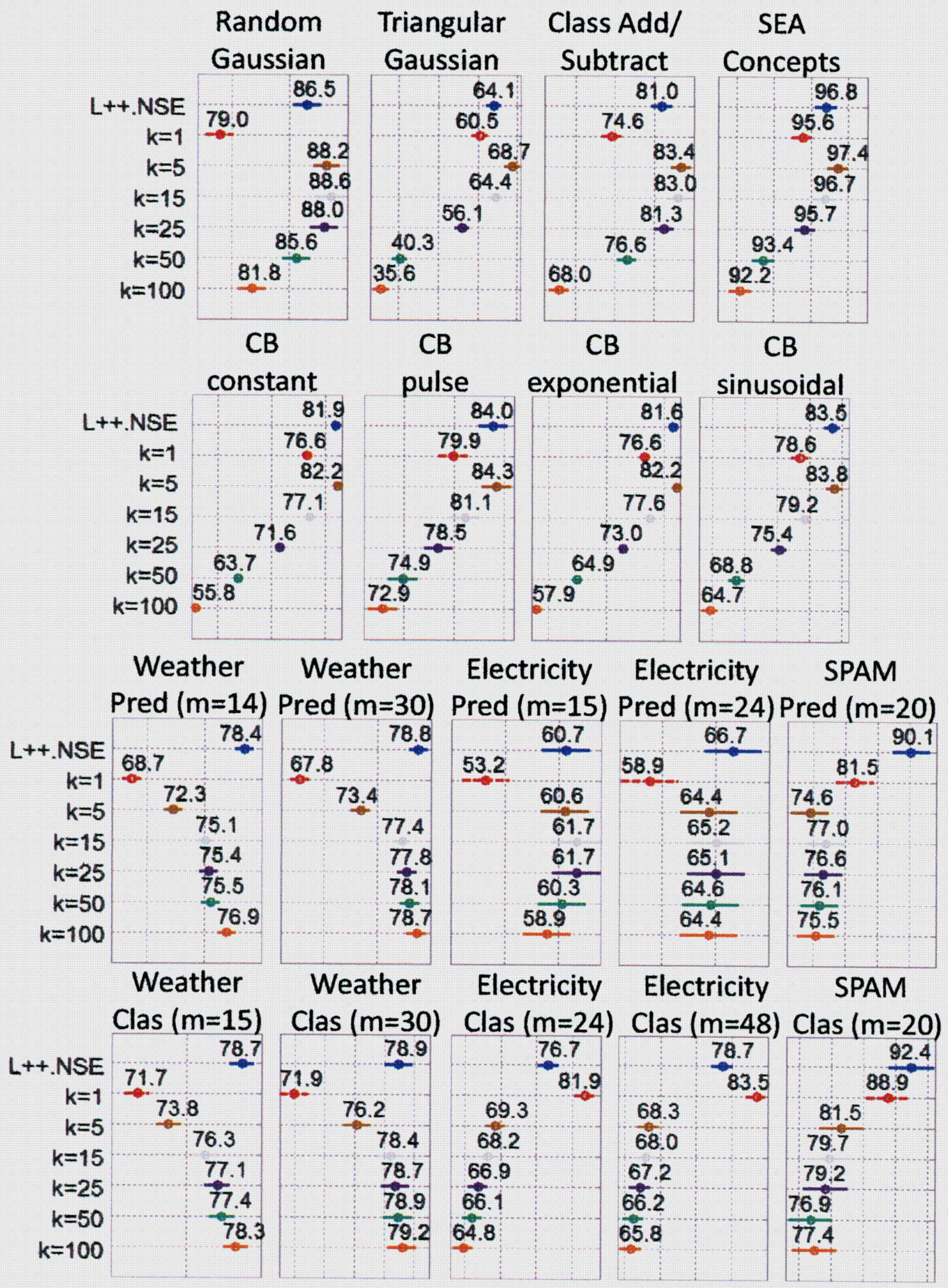


Figure D.27: ANOVA comparison for Streaming Ensemble Algorithm (SEA) pruning characterization

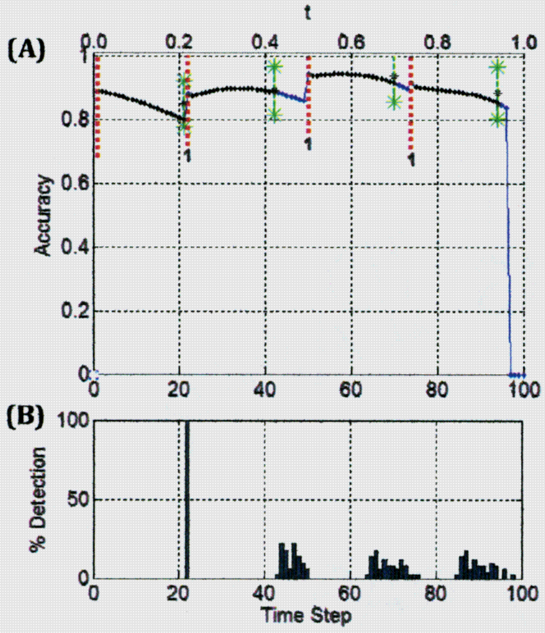


Figure E.1: Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on random Gaussian drift data

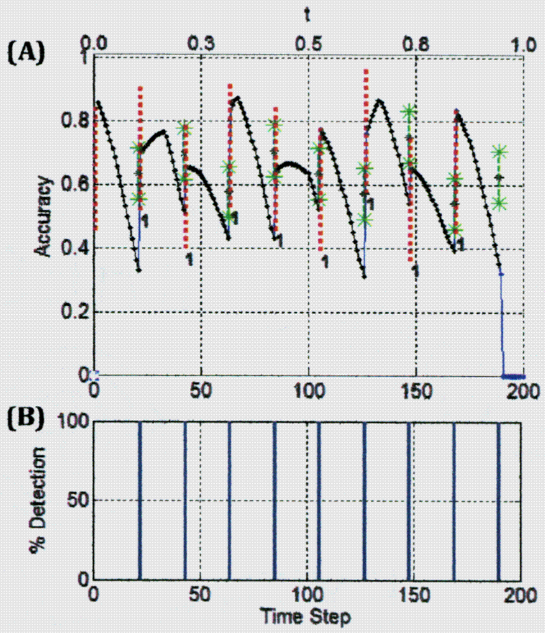


Figure E.2:Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on triangular Gaussian drift data

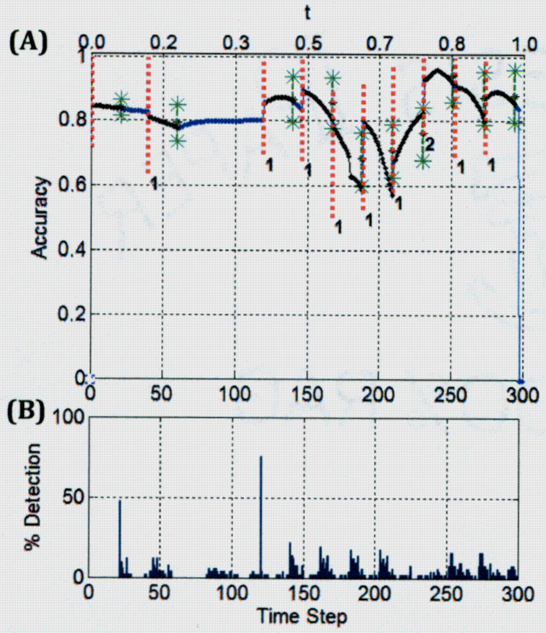


Figure E.3:Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on random Gaussian drift data with class addition/removal

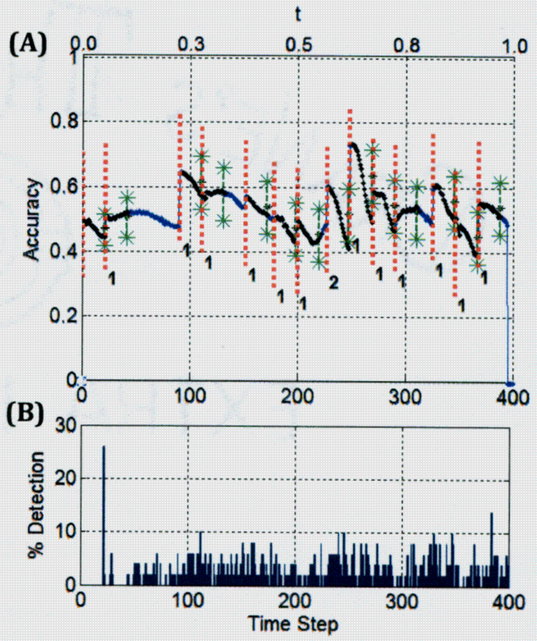


Figure E.4: Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on checkerboard data (exponential drift)

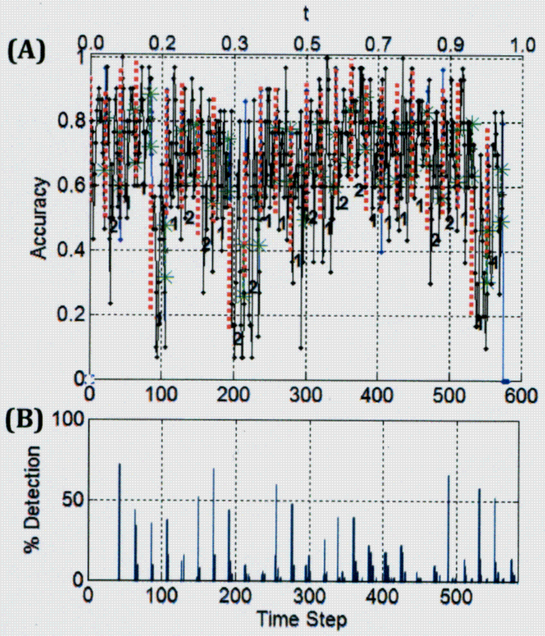


Figure E.5:Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on weather prediction data

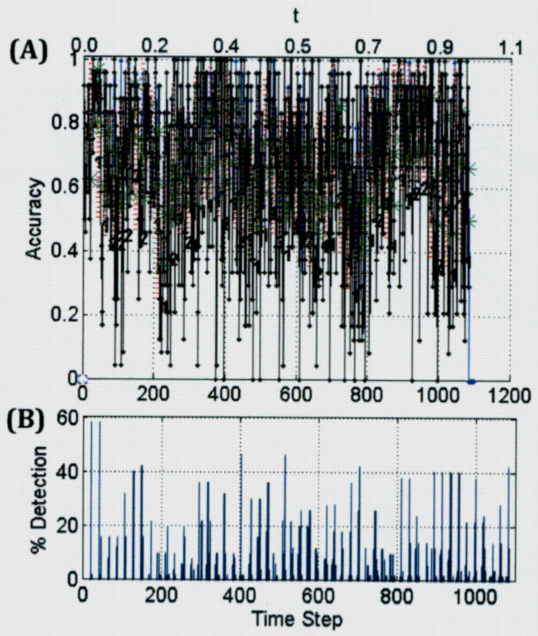


Figure E.6:Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on electricity pricing data

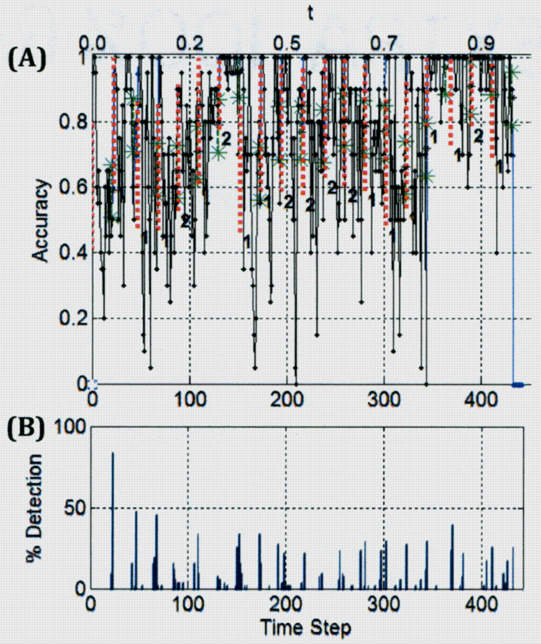


Figure E.7:Classifier performance (upper) and detection % per time step over 50 trials (lower) for drift detection on spam prediction data

Appendix F: Sigmoid Characterization for Support Vector Machine (SVM)

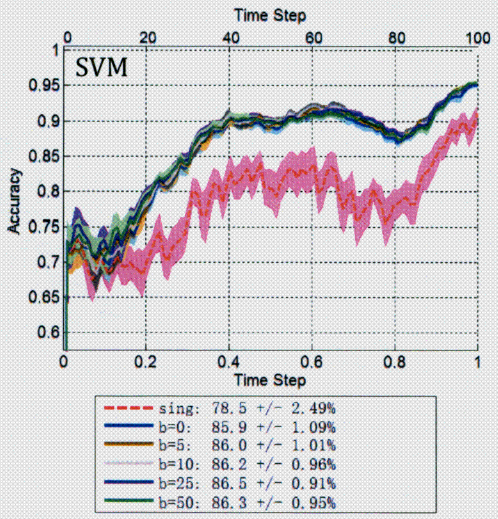


Figure F.1: Sigmoid characterization for random Gaussian drift data

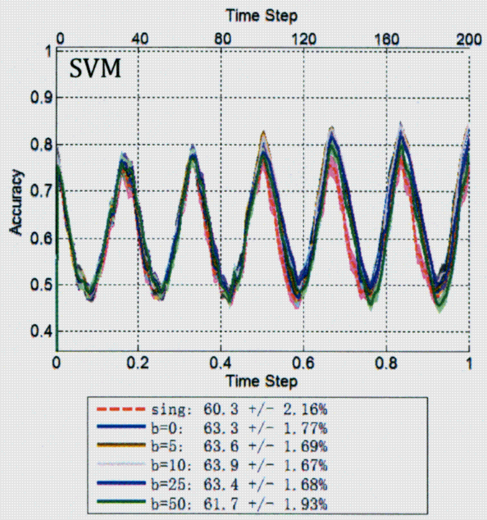


Figure F.2: Sigmoid characterization for triangular Gaussian drift data

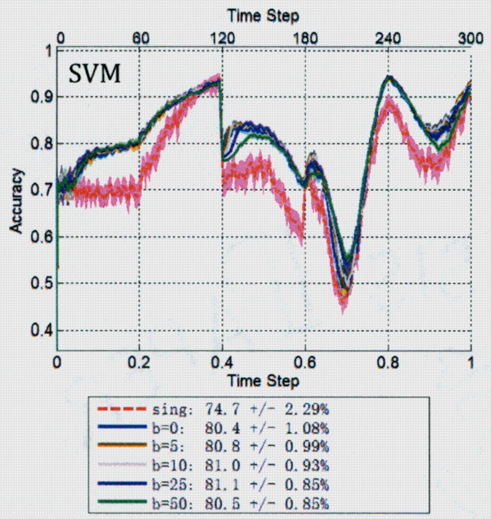


Figure F.3: Sigmoid characterization for random Gaussian drift with class addition/removal

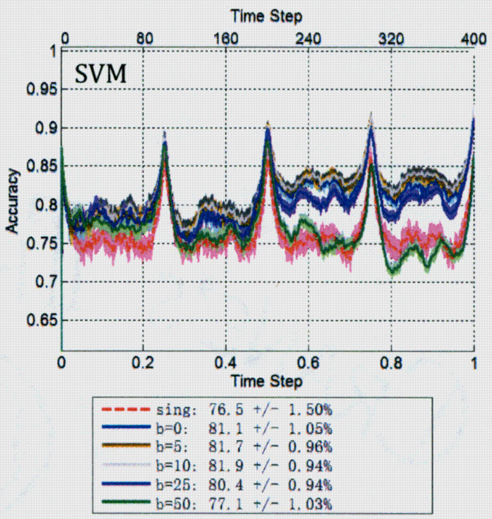


Figure F.4: Sigmoid characterization for checkerboard data (constant drift)

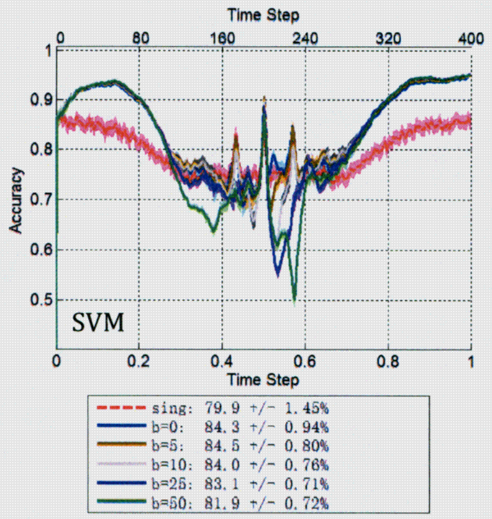


Figure F.5: Sigmoid characterization for checkerboard data (pulsing drift)

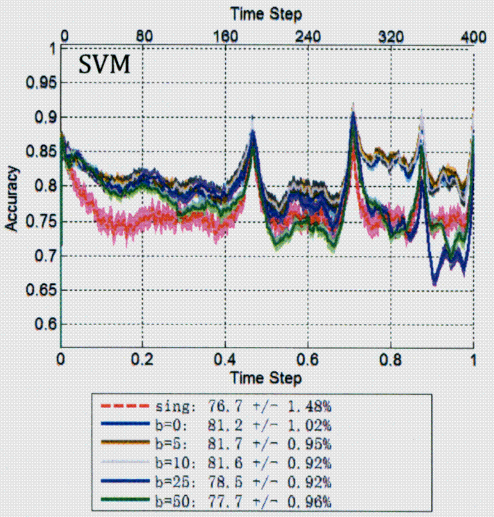


Figure F.6: Sigmoid characterization for checkerboard data (exponential drift)

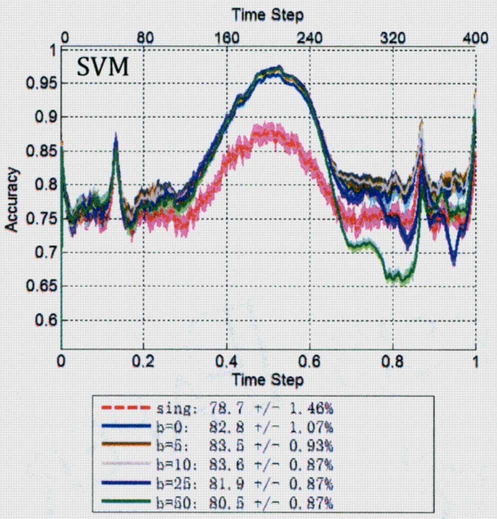


Figure F.7: Sigmoid characterization for checkerboard data (sinusoidal drift)

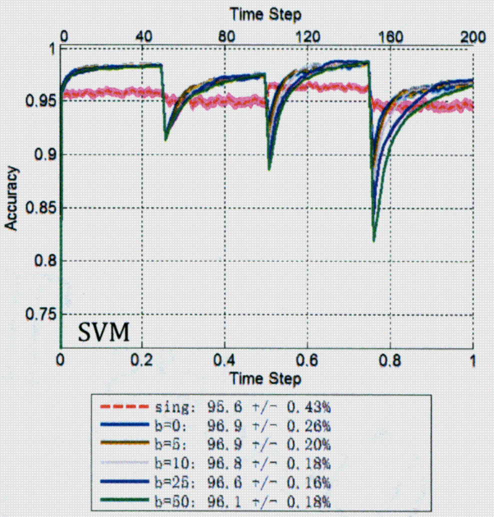


Figure F.8: Sigmoid characterization for SEA concept data