University of Exeter
Department of Computer Science

Learning from Data Streams with Randomized Forests

Timothy Joseph Hoay-Chi Seah

September, 2021

Supervised by Professor Richard Everson &
Professor Jonathan Fieldsend

Abstract

Non-stationary streaming data poses a familiar challenge in machine learning: the need to obtain fast and accurate predictions. A data stream is a continuously generated sequence of data, with data typically arriving rapidly. They are often characterised by a non-stationary generative process, with concept drift occurring as the process changes. Such processes are commonly seen in the real world, such as in advertising, shopping trends, environmental conditions, electricity monitoring and traffic monitoring.

Typical stationary algorithms are ill-suited for use with concept drifting data, thus necessitating more targeted methods. Tree-based methods are a popular approach to this problem, traditionally focussing on the use of the Hoeffding bound in order to guarantee performance relative to a stationary scenario. However, there are limited single learners available for regression scenarios, and those that do exist often struggle to choose between similarly discriminative splits, leading to longer training times and worse performance. This limited pool of single learners in turn hampers the performance of ensemble approaches in which they act as base learners.

In this thesis we seek to remedy this gap in the literature, developing methods which focus on increasing randomization to both improve predictive performance and reduce the training times of tree-based ensemble methods. In particular, we have chosen to investigate the use of randomization as it is known to be able to improve generalization error in ensembles, and is also expected to lead to fast training times, thus being a natural method of handling the problems typically experienced by single learners.

We begin in a regression scenario, introducing the Adaptive Trees for Streaming with Extreme Randomization (ATSER) algorithm; a partially randomized approach based on the concept of Extremely Randomized (extra) trees. The ATSER algorithm incrementally trains trees, using the Hoeffding bound to select the best of a random selection of splits. Simultaneously, the trees also detect and adapt to changes in the data stream. Unlike many traditional streaming algorithms ATSER trees can easily be extended to include nominal features. We find that compared to other contemporary methods ensembles of ATSER trees lead to improved predictive performance whilst also reducing run times.

We then demonstrate the Adaptive Categorisation Trees for Streaming with Extreme

Randomization (ACTSER) algorithm, an adaption of the ATSER algorithm to the more traditional categorization scenario, again showing improved predictive performance and reduced runtimes. The inclusion of nominal features is particularly novel in this setting since typical categorization approaches struggle to handle them.

Finally we examine a completely randomized scenario, where an ensemble of trees is generated prior to having access to the data stream, while also considering multivariate splits in addition to the traditional axis-aligned approach. We find that through the combination of a forgetting mechanism in linear models and dynamic weighting for ensemble members, we are able to avoid explicitly testing for concept drift. This leads to fast ensembles with strong predictive performance, whilst also requiring fewer parameters than other contemporary methods.

For each of the proposed methods in this thesis, we demonstrate empirically that they are effective over a variety of different non-stationary data streams, including on multiple types of concept drift. Furthermore, in comparison to other contemporary data streaming algorithms, we find the biggest improvements in performance are on noisy data streams.

To my parents

Acknowledgements

Throughout my PhD journey, I have been blessed with the presence of many people along the way. While it would be challenging to thank them all, I would like to express my gratitude to some of those who have guided, supported and encouraged me here.

I would like to begin by thanking my supervisors Professor Richard Everson and Professor Jonathan Fieldsend. Their ongoing guidance has been invaluable throughout my studies and I cannot thank them enough for the numerous hours that they have invested into patiently advising me. I would particularly like to thank Richard for the continuous inspiration and passion he has provided throughout. I would also like to thank Jonathan for his insightful advice and feedback on various topics, especially regarding java implementations.

None of this work would have been possible without the funding provided by the team at EG, and I would like to express my appreciation to Steve and Tom for their interest and involvement. Their unique observations and suggestions have repeatedly helped to develop and improve my research.

During my studies, I had the pleasure of sharing an office with a wonderful group of fellow PhD students, and would like to thank Carina, Dan, Finley and John for the various discussions we had, on topics ranging from statistics to stargazing.

Since its inception, the members of the Belmont PhD group have been a constant source of support and friendship and I am grateful for the advice and encouragement they have offered, whether it be during a study session or a picnic in the park.

I would also like to thank my parents, Joe and Sandra, and my brothers Tobias and Theo, for the love and encouragement they have shown throughout my studies. Furthermore, thank you Kay for your patience and understanding throughout, being there to hear me out and support me at any time, be it 3am or 3pm.

Finally, thank you God for keeping me going through all the struggles a PhD entails.

Contents

List of Figures

Nomenclature and Abbreviations

Acronyms

ABORF	Approximately Bayesian Online Randomized Forest *(p. 100)*
ACTSER	Adaptive Classification Trees for Streaming with Extreme Randomization *(p. 5)*
ADWIN	Adaptive Windowing Algorithm *(p. 26)*
ARF	Adaptive Random Forest *(p. 37)*
ARF-Reg	Adaptive Random Forest for Regression *(p. 37)*
ATSER	Adaptive Trees for Streaming with Extreme Randomization *(p. 4)*
CPU	Central Processing Unit *(p. 40)*
CUSUM	Cumulative Sum *(p. 28)*
CVFDT	Concept-Adapting Very Fast Decision Tree *(p. 3)*
DDM	Drift Detection Method *(p. 27)*
EFDT	Extremely Fast Decision Tree *(p. 33)*
EWMA	Exponentially Weighted Moving Average *(p. 28)*
FIMT-DD	Fast Incremental Model Tree with Drift Detection *(p. 3)*
FLORA	Floating Rough Approximation *(p. 29)*
KL	Kullback-Leibler *(p. 17)*
MAE	Mean Absolute Error *(p. 38)*
MOA	Massive Online Analysis *(p. 40)*
MSE	Mean Squared Error *(p. 35)*
OAUE	Online Accuracy Updated Ensemble *(p. 36)*
PH	Page-Hinckley *(p. 28)*
RAM	Random Access Memory *(p. 40)*
RMSE	Root Mean Squared Error *(p. 38)*
SVM	Support Vector Machine *(p. 30)*
VFDT	Very Fast Decision Tree *(p. 2)*
VR	Variance Reduction *(p. 48)*

Nomenclature

$\boldsymbol{\beta}$	A vector of linear model weights for each feature, $\boldsymbol{\beta} = \{\beta_0, ..., \beta_D\}$ *(p. 51)*
$\mathbf{x}$	$\mathbf{x} = (x_1, ..., x_D)$, A vector of feature observations, with each element in the observation a corresponding to a particular feature d, $d \in \{1, ..., D\}$. Individual features may be numerical or nominal. *(p. 7)*
$\mathbf{y}$	The values we are trying to predict, known as the target values. Can be numerical (regression problems) or nominal (categorisation problems). Each element in the $\mathbf{y} = (y_1, ..., y_M)$ observation corresponds to a particular target m, $m \in \{1, ..., M\}$ *(p. 7)*
γ	A smoothing factor. *(p. 52)*
$\hat{y}_t$	The estimate of the target value at time t *(p. 38)*
$\mathbb{E}[\,]$	The expected value of the quantity in brackets *(p. 52)*
$\mathcal{T}$	A decision tree *(p. 47)*
ν	A node in the tree *(p. 102)*
$i.i.d.$	independent and identically distributed *(p. 104)*
$p_t(\mathbf{y}, \mathbf{x})$	The joint probability distribution of $\mathbf{y}$ and $\mathbf{x}$ at time t *(p. 7)*
t	A specific point in time *(p. 7)*
D	The dimensionality of the data. Each dimension corresponds to a specific feature j. *(p. 51)*

Publications

Some of the materials presented in Chapter 4 have been submitted for publication:

Seah, Timothy J., Everson, Richard M., and Fieldsend, Jonathan E. (2021). Adaptive Trees for Streaming with Extreme Randomiztion. *Machine Learning*

Chapter 1

Introduction

One of the hallmarks of the modern era is the rapid development of machine learning. Since the advent of computers, humanity has turned to them to perform tasks on our behalf, leading to revolutionary improvements in the quality of our lives. Machine learning is a particularly important area in this regard, with practitioners constantly seeking to create algorithms that allow computers to learn and adapt to "experiences" without explicit intervention. This "experience" is passed to computers in the form of data, with algorithms designed to learn by recognising patterns in the data presented. In this manner machines are able to learn to perform a wide variety of tasks, many of which we are inefficient at or even incapable of performing ourselves, including speech and image recognition, medical diagnoses, natural language processing, email filtering, and robotics, all of which have become embedded in modern society as ways to improve and develop our daily lives.

1.1 Contextual Setting

Machine learning is broadly comprised of three major categories: unsupervised, supervised and reinforcement learning. In the context of this thesis, we focus on supervised learning. In this setting data arrives in pairs of input and target output values, which are used to learn a mapping between the provided input and target output values. After the mapping is learnt, a provided input can be used to predict an unknown target output in a task known as predictive modelling.

Predictive modelling has applications across many fields. Given a sufficiently accurate mapping, it is able to answer questions we may not have an obvious answer to, such as *"Which disease do these symptoms suggest a patient has?"*, *"Which consumer would be most interested in these products?"*, and *"What will the traffic be like in Exeter tomorrow?"*. However, it is clear that training an accurate model for these circumstances requires a nuanced approach. For example, consumer tendencies change over time, meaning that any mapping learnt will only be accurate for a short period. This leads us to the next consideration of this thesis: dataset shift.

A common assumption in supervised learning is that the data used to train a model follow the same distribution as the data the model will be used to make predictions on: the testing data. Intuitively, this makes sense as a model is learnt based on the training data. Consequently, it would be reasonable to expect more accurate predictions on similar data since the information it received when learning is more relevant to such data. However, this assumption of the training and testing data following identical distributions is often violated in real-world problems. While this disparity between the training and testing data can take a number of different forms, the general case may be termed as “dataset shift” [Quiñonero-Candela et al., 2009].

In this thesis, we focus on a specific form of dataset shift, in which the incoming data are projected to arrive sequentially over time as a non-stationary data stream [Aggarwal, 2006]. Under this framework, the effect of dataset shift, which can occur over a number of examples and at multiple points in the stream, is a phenomenon termed as “concept drift” [Widmer and Kubat, 1996]. In comparison to the stationary or batch learning scenarios in which dataset shift occurs, the streaming setting means working with concept drifting data is considerably more challenging as the underlying generative distribution of the data cannot be assumed to be constant over a training set or batch of data, making it harder to identify and correct for drifts.

This sets up learning under concept drift as a specific form of online learning, in which the incoming data are are not assumed to be stationary over time. Unlike traditional time series forecasting, no complete training set is assumed to be available at any given time, and older data may be related to previous concepts and thus unsuitable for learning from. On the other hand, while data streams may be infinitely long and lead to a continual learning process, unlike lifelong learning the data stream provides a fixed domain on which tasks are required to be performed.

Perhaps the most well known foundational algorithm for learning in a streaming data scenario is the Very Fast Decision Tree (VFDT), or Hoeffding Tree [Domingos and Hulten, 2000]. The VFDT provides a method of learning decision trees from a data stream, using the Hoeffding Bound [Hoeffding, 1963] to provide a theoretical guarantee of converging to the stationary result, providing the data stream is stationary. On the other hand, it is widely recognised that the VFDT typically performs poorly in the presence of concept drift. This has led to a number of approaches being introduced for use in this scenario, with most revolving around the idea of adaption: adaptive algorithms are able to update the mapping they learn in reaction to drifts in the data. One of the first adaptive algorithms is a modification of the VFDT known as the Concept-Adapting Very Fast Decision Tree

(CVFDT). Other decision tree based algorithms have also been developed around the VFDT, often focussing on either single learners [Pfahringer et al., 2007; Bifet and Gavald, 2009; Manapragada et al., 2018], or on ensembles in which decision trees act as the base learners [Kolter and Maloof, 2003; Bifet et al., 2012].

Most research has, however, been targeted at classification problems. Perhaps the first major work on concept drift in a regression is the Fast Incremental Decision Tree with Drift Detection (FIMTDD) algorithm [Ikonomovska et al., 2011b], which trains an online regression tree that adapts in response to changes in the data stream. While following works have developed the regression approach using ensembles [Gomes et al., 2018] and extended it to handle other scenarios such as multi-target regression [Osojnik et al., 2018], there has been limited attention given to alternative strategies for training single learners. This is problematic since it has limited diversity in ensemble approaches in which FIMT-DD acts as a base learner. An example of this can be seen in areas in which FIMT-DD struggles, such as choosing between similarly discriminative splits, in turn being a common problem across ensemble strategies.

One strategy which may be able to overcome difficulties in the FIMT-DD algorithm is the use of randomization, since randomly choosing a split would remove the need to choose between similarly discriminative ones. Furthermore, randomization has been shown to be able to improve generalization error in ensembles trained for use with concept-drifting data, [Minku et al., 2010; Minku and Yao, 2012] . Another benefit of randomization-based approaches is that they typically train multiple weak learners, with each having lower data requirements than non-randomized approaches. Consequently, we also expect randomization to lead to to reduced training times, thus being a natural method of handling many of the the problems experienced by models designed for concept drifting regression streams.

This thesis contains works for training predictive models in the presence of concept drift. It primarily looks toward a regression scenario, in which it seeks to address the limited pool of single learners and problems caused by dependence on the FIMT-DD algorithm in many ensembles. It does this through the development of additional tree-based approaches, offering alternative strategies for training single learners, and presenting ensembles of them for comparison with other contemporary algorithms. In particular, it explores the use of randomization in the splitting process when growing decision trees, and also considers the idea of fully randomized decision trees which can be pre-grown before observing data. It further examines the effectiveness of applying some of the methods in the better-addressed classification scenario.

1.2 Research goals

The main goal of this thesis is to study decision tree based methods for predictive modelling on data streams. As data streams are characterized by high volumes and frequencies of examples, they necessitate a secondary goal of simultaneously minimizing the required training and testing times. To these ends, we aim to examine the impact of randomization in the structure of decision trees on their predictive performance. Randomization has previously been shown to improve the generalization ability of models with concept drifting data [Minku et al., 2010; Minku and Yao, 2012], leading to improved predictive performance. In our setting, we further hope to take advantage of randomization to reduce the time complexity of algorithms. Thus, throughout this thesis, we focus on the following research questions:

- Does randomization in the structure of decision trees impact their predictive performance on nonstationary data streams?
- If so, to what extent is randomization in the structure of decision trees helpful for improving performance?
- Can randomization in the structure of decision trees be used to reduce their run times whilst simultaneously maintaining similar levels of predictive performance.

1.3 Contributions

This thesis focusses on the problem of predictive modelling with non-stationary data streams. It aims to provide rapidly training incremental algorithms for making accurate predictions on an unknown and likely non-stationary data stream. Our main contributions are listed as follows:

1. We have designed and implemented the Adaptive Trees for Streaming with Extreme Randomization algorithm (ATSER) for use with regression data streams. We introduce an adaptive modelling approach in the leaves, allowing them to better adapt in the presence of localized concept drift, while the novel splitting mechanism we suggest allows us to leverage the use of randomization during the splitting process. Together, this method leads to shorter splitting times and improved predictive accuracy compared to previously suggested approaches.
2. We have adapted the ATSER algorithm to a categorization scenario and implemented it. We show that our novel splitting mechanism allows the inclusion of nominal variables - a previously prohibitively expensive task for other streaming classification algorithms.

3. We have taken a Bayesian approach to design a model based on an ensemble of pre-grown regression trees with tree weights found using importance weighting. We show that this allows us to decouple the tree construction from the data, reducing initial training times whilst still retaining the ability to adapt in the presence of concept drift. We introduce a novel technique to handling concept drift in this Bayesian framework, and show that this principled approach leads to accurate predictions, but with slow runtimes. We further introduce a novel approximation of this method for use in a data streaming scenario, leading to a faster algorithm while retaining a strong predictive performance in comparison to other contemporary methods.

1.4 Layout

This introduction is designed to begin by providing the reader with a broad overview of the topic of this thesis, highlighting the main goals motivating the work undertaken, and the primary contributions of the thesis.

Chapter 2 gives a breakdown of concept drift classifications, as well as its stationary counterpart, dataset shift. It offers a number of methods for classifying different types of concept drift and dataset shift. Most of these methods can be directly applied in a streaming scenario and include a theoretically motivated exhaustive breakdown of different types of shift. It follows with a review of common approaches to predictive modelling under a theoretical classification of dataset shift.

Chapter 3 gives a more detailed review of the literature pertaining to concept drift. It provides an overview of the setting which they concept drift occurs, before then reviewing common approaches to both detecting and predictive modelling with concept drifting data. The review focusses in particular on methods based on both decision trees and ensembles of decision trees. Finally, a discussion of the methodologies used to evaluate predictive models designed for concept drifting streaming data is presented.

Chapter 4 introduces the Adaptive Trees for Streaming with Extreme Randomization (ATSER) algorithm. It begins with a discussion of the related literature, before introducing the ATSER algorithm and illustrating how it seeks to build on previous concept adapting regression tree techniques. It then empirically demonstrates the performance, showing how the approach leads to improved predictive accuracy and reduces training times relative to other contemporary algorithms.

Chapter 5 introduces the Adaptive Classification Trees for Streaming with Extreme Randomization (ACTSER) algorithm. It begins by providing an overview of the related literature in a classification scenario, before introducing an adaption of ATSER to a classification setting. It shows that the novel splitting mechanism provides an easy way to include nominal features in the tree construction process, which is prohibitively costly under other contemporary approaches, before providing a comparison of the performance with other contemporary algorithms.

Chapter 6 begins by introducing a principled Bayesian method to modelling on non-stationary streaming data, based on the use of a random forest to partition data. It shows how the trees within the model can be grown before data arrives, allowing the training time after each example arrives to be reduced. While the approach we present is robust, it is also slow, a common flaw among Bayesian techniques. To supplement this, Chapter 6 also presents an approximation the model which is better suited for use in a streaming scenario due to having a lower time complexity. Finally, empirical results from both the Bayesian approach and the approximation are presented.

Finally, we conclude in chapter 7. We present a summary of the findings and contributions of this thesis, providing possible directions for further work.

Chapter 2

Characterizing Concept Drift and Dataset Shift

Problems on concept drift have been studied for a long period, with the phrase "concept drift" first being noted by Schlimmer et al. [1986] and Widmer and Kubat [1996] amongst streaming literature. It has also arisen independently in a stationary and batch learning scenario, under the term "dataset shift". This term was first introduced in the book 'Dataset Shift in Machine Learning' [Quiñonero-Candela et al., 2009]. According to the definition given there, 'Dataset shift is a common problem in predictive modelling that occurs when the joint distribution of inputs and outputs differs between training and test stages'. Dataset shift is the stationary counterpart to concept drift, although mathematically, both follow the same definition:

$$p_{tr}(\mathbf{y}, \mathbf{x}) \neq p_{te}(\mathbf{y}, \mathbf{x}) \tag{2.1}$$

where $p_{tr}(\mathbf{y}, \mathbf{x})$ and $p_{te}(\mathbf{y}, \mathbf{x})$ refer to the joint distributions of the examples, $(\mathbf{y}, \mathbf{x})$, in the training and testing datasets respectively, whilst the examples are comprised of the labels, $\mathbf{y}$, and covariates, $\mathbf{x}$. Consequently, the difference between concept drift and dataset shift is based purely on the scenarios in which a drift or shift occurs. Typically, it is assumed in dataset shift scenarios that a set or batch of training data is available, for which $p_{tr}(\mathbf{y}, \mathbf{x})$ is fixed. However, for concept drifting scenarios there is a temporal aspect to the data distribution as drifts may occur as new examples arrive, meaning there is no guarantee that $p_{tr}(\mathbf{y}, \mathbf{x})$ is fixed for the data used for training.

From the definition of concept drift in Equation 2.1, it is obvious that it occurs to some degree in almost all real-world problems, as both the training and testing sets, and the distributions from which they are drawn, are seldom identical. While the magnitude of these drifts may differ between problems, they are often non-negligible and, consequently, concept drift is an important factor to consider when building predictive models.

2.1 Types of Concept Drift

Over recent years, different forms of dataset shift have garnered increasing amounts of attention. However, it is challenging to make a collective analysis as the broad relevance of the topic means it has received attention in both stationary and streaming fields, each with a different focus. This has led to the literature being fractured, an issue that continues to persist despite a number of unifying attempts.

The split in the literature is especially prevalent due to a difference in terminology between the stationary and streaming scenarios. Gama et al. [2014] formally define the term "concept drift" in the same way as Equation 2.1, the definition [Quiñonero-Candela et al., 2009] used for dataset shift. However, further breakdowns of types concept drift are challenging to discuss due to the number of ways a drift can occur, leading to many ways of classifying them, each with their own advantages.

2.1.1 A Theoretical Classification

One way of categorising concept drift is a purely theoretical decomposition, as suggested by Moreno-Torres et al. [2012], who propose a theoretically motivated method of splitting dataset shift into 3 main types based on the effects of the shift, namely; simple covariate shift, prior probability shift, and concept shift. While originally presented in a dataset shift framework, these definitions also hold for concept drift.

With features $\mathbf{x}$ and targets $\mathbf{y}$, we denote by $\mathbf{x} \rightarrow \mathbf{y}$ problems where $\mathbf{x}$ causes $\mathbf{y}$. Then, the types of dataset shift suggested by Moreno-Torres et al. [2012] shifts can be seen from the definition of dataset shift given in Equation 2.1, which can be written as:

$$p_{tr}(\mathbf{y}|\mathbf{x})p_{tr}(\mathbf{x}) \neq p_{te}(\mathbf{y}|\mathbf{x})p_{te}(\mathbf{x}) \tag{2.2}$$

in the case of $\mathbf{x} \rightarrow \mathbf{y}$ problems, or as:

$$p_{tr}(\mathbf{x}|\mathbf{y})p_{tr}(\mathbf{y}) \neq p_{te}(\mathbf{x}|\mathbf{y})p_{te}(\mathbf{y}) \tag{2.3}$$

in the case of $\mathbf{y} \rightarrow \mathbf{x}$ problems. From Equation 2.2, it can then be seen that dataset shift in the $\mathbf{x} \rightarrow \mathbf{y}$ case must be due to either a difference in the covariate distribution:

$$p_{tr}(\mathbf{x}) \neq p_{te}(\mathbf{x}) \tag{2.4}$$

or a difference in the mappings:

$$p_{tr}(\mathbf{y}|\mathbf{x}) \neq p_{te}(\mathbf{y}|\mathbf{x}) \tag{2.5}$$

or a combination of these. These two cases are known as simple covariate shift and concept shift respectively. Furthermore, in the $\mathbf{y} \rightarrow \mathbf{x}$ case given in Equation 2.3, a shift can arise from either a difference in the prior probability:

$$p_{tr}(\mathbf{y}) \neq p_{te}(\mathbf{y}) \tag{2.6}$$

or a difference in the mappings:

$$p_{tr}(\mathbf{x}|\mathbf{y}) \neq p_{te}(\mathbf{x}|\mathbf{y}). \tag{2.7}$$

Consequently, the shift shown in Equation 2.6 is known as prior probability shift, while Equation 2.7 details another form of concept shift. Altogether this leads to a total of 3 main types of dataset shift.

For completeness, Moreno-Torres et al. [2012] also note that it is theoretically possible to have a different type of shift which would take the form $p_{tr}(\mathbf{x}|\mathbf{y}) \neq p_{te}(\mathbf{x}|\mathbf{y})$ in $\mathbf{x} \rightarrow \mathbf{y}$ problems or $p_{tr}(\mathbf{y}|\mathbf{x}) \neq p_{te}(\mathbf{y}|\mathbf{x})$ when $\mathbf{y} \rightarrow \mathbf{x}$. However, they also point out that such shifts are uncommon and are considered too difficult to handle, and as such they will not be discussed in this thesis.

This breakdown is appealing since the theoretical motivation it follows is exhaustive, although Webb et al. [2016] argue that some of the constraints in the definitions are too strong. A similar classification to this arising in the data streaming field has been suggested by Gama et al. [2014], who provide a similar breakdown, referring to concept shift as "real concept drift" and using the term "virtual drift" to collectively refer to both covariate and prior probability shift. Further works on concept drift have also adopted this theoretical approach [Khamassi et al., 2018].

2.1.2 A Characteristic Classification

One of the earliest classifications of types of concept drift arising from streaming data is an approach based on the properties of the drift. In most streaming literature, concept drift has traditionally been split into four different categories: abrupt, gradual, incremental and recurring concepts [Kuncheva, 2004; Narasimhamurthy and Kuncheva, 2007; Žliobait, 2010; Gama et al., 2014]. Furthermore, while not actually a drift, it is important to consider that when viewing an outlier in a streaming scenario, it may appear to be a drift. This breakdown is a qualitative breakdown, based on the perceived properties of the drift, and is often helpful when considering algorithm design. An illustration of these types of drifts can be seen in Figure 2.1, in which the probability of incoming data being generated under the initial concept regime (here labelled C_0) is shown to change over time.

Each type of real concept drift possesses different characteristics, as detailed in the follow-

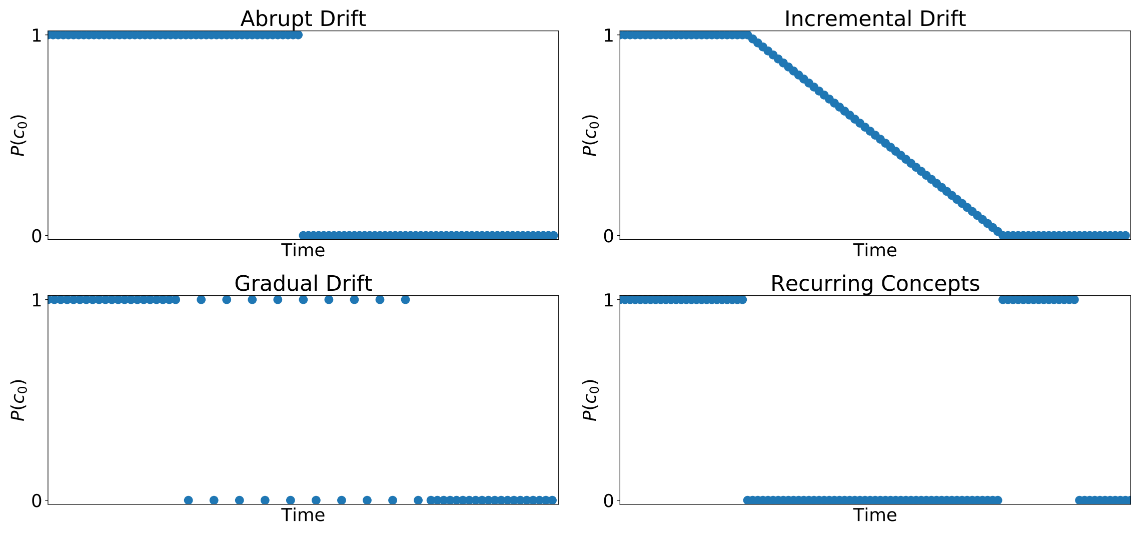


Figure 2.1. Illustrations of different types of concept drift. The points shown represent the probability of each incoming example following the initial concept regime, termed C_0.

ing descriptions:

Abrupt: In an abrupt drift, the concepts change over a very short time frame relative to the frequency of incoming data, effectively instantly switching from one concept regime to another. Consider electricity consumption for lighting at different times of day. An abrupt drift occurs at the daylight saving time switch-over, as the times instantly correspond to different consumption amounts.

Incremental: In an incremental drift, the concepts change much more slowly. One manner in which this may occur is with incoming data being split between multiple concept regimes during a transition period. Electricity consumption for lighting at different times of day also experiences incremental drifts as the seasons change across the year.

Gradual: In a gradual drift, each incoming set of data at a specific time all follows the same concept, but there is a period of time where the data switch between two concepts. This might occur with electricity bills, where if a competing electricity provider lowers prices to attract customers, eventually consumers will transition towards the competitor.

Recurring: In a recurring drift, the concepts recur, with concepts that leave the data being reintroduced at later times. Again, electricity consumption for lighting experiences this type of drift, with a recurring drift from year-to-year.

Outlier: While not actually a form of concept drift, outliers may be mistaken for one and thus are an important consideration when working with data expected to exhibit concept drift.

Table 2.1. The concept drift categorization proposed by Minku et al. [2010].

Criteria			Categories	
Drift in Isolation	Severity	Class	Severe	
			Intersected	
		Feature	Severe	
			Intersected	
	Speed		Abrupt	
			Gradual	Proabilistic
				Continuous
Drift in Sequences	Predictability		Predictable	
			Non Predictable	
	Frequency		Periodic	
			Non-Periodic	
	Recurrence		Recurrent	Cyclic
				Unordered
			Non-Recurrent	

It is clear that this concept drift classification is simple, and mainly based on two main criteria: the speed of a drift, and its recurrence. However, Minku et al. [2010] argue that this approach is too simplistic and leads to drift with different behaviour having the same classification. An example of this can be seen by considering two cases of an incremental drift, both of which result in the quality of goods produced by a company (the target) improving over time, despite having no change in the raw materials available (the features). If the company owns two factories, this drift could be because one factory produces higher quality goods than the other, and the proportion produced at each factory has changed. On the other hand, it could also be that the standard of production at one factory has improved.

Minku et al. [2010] argue that because of situations where drifts of the same type occur very differently, as illustrated in the above example, a further classification is necessary. They propose a detailed manner of classifying drifts through a number of different criteria, extending considerations from speed and recurrence to also include frequency, predictability and severity, as shown in Table 2.1. This approach provides a clearer classification of concept drift and allows a deeper study of the topic. However, it also requires more properties about the data to be known, which can be challenging to estimate beforehand in practice.

2.1.3 A Causal Classification

While the concept drift classifications introduced so far seek to heterogeneously classify the type of a drift, multiple types of drift often occur simultaneously in practice. Consequently, in a dataset shift framework, Storkey [2008] classifies based on the cause of the shift, as opposed to a theoretically motivated classification, arguing that this is a natural way to classify shifts since it allows for the construction of targeted models to handle them. However, due to the vast number of possible causes of shifts, as well as cases in which the cause of a shift is unidentifiable, such a framework is hard to evaluate comprehensively and challenging to develop generally applicable approaches with. As such, Storkey focuses on six common types of shift: simple covariate shift, prior probability shift, sample selection bias, imbalanced data, domain shift, and source component shift.

Storkey's classification of simple covariate and prior probability shift are much the same as the classification by Moreno-Torres et al. [2012]. Furthermore, Storkey's use of domain shift is to indicate a change in the meaning of the numbers, and represents a similar idea to concept shift. On the other hand, the other shifts are notably different and have gained more attention in other fields. Sample selection bias has long been of interest in the field of economics [Heckman, 1979; Vella, 1998], and has only more recently gained attention as a form of dataset shift [Zadrozny, 2004; Cortes et al., 2008]. In most situations, sample selection bias has been considered using a binary selection variable, s, such that:

$$p_{tr}(\mathbf{y}, \mathbf{x}) = p(\mathbf{y}, \mathbf{x}|s = -1) = p(\mathbf{y}|\mathbf{x}, s = -1)p(\mathbf{x}|s = -1) \tag{2.8}$$

$$p_{te}(\mathbf{y}, \mathbf{x}) = p(\mathbf{y}, \mathbf{x}|s = 1) = p(\mathbf{y}|\mathbf{x}, s = 1)p(\mathbf{x}|s = 1). \tag{2.9}$$

In this way, case by case simplifications can be made using Bayes theorem and knowledge about the type of bias introduced.

Imbalanced data is commonly introduced by design, but is known to have consequences on classification accuracy [Latinne et al., 2001]. As noted by Storkey, it is a special case of sample selection bias in which data are excluded in a classification problem based on their output class. This exclusion is presumed to occur due to the primary focus of the model being the prediction of rare events. As such, data points are often excluded with the objective of balancing the number of points in each class. The implicit assumption here is that the excluded data points have negative value when predicting the rare events. The methodology for correcting for this form of dataset shift is fairly straightforward, and simply requires re-weighting and re-normalisation of the output probabilities of being in each class based on the bias introduced by excluding certain points. However, even under these circumstances it can be practically challenging to implement corrections, especially when there are few examples in the minority class, since without many examples any re-weighting is likely to be unstable.

The final type of dataset shift identified is source component shift. As its name suggests, source component shift can arise when a dataset is comprised of examples from different sources. Storkey suggests three different ways this type of shift can occur and, depending on the origin, provides a different model for each.

This approach is insightful since it provides a method of specifically targeting shifts based on their cause. While additional complications arise in a streaming scenario, it is often possible to know before data arrives whether certain causes of drift will occur, allowing this dataset shift classification to also provide a foundation for classifying concept drift. However, practically, streaming data also frequently contains concept drift from unknown causes. Despite this, separating corrections for some known forms of concept drift whist simultaneously correcting for an unknown drift may lead to improvements over treating all of the drift as unknown. While the topic has been largely unaddressed, class imbalance has received some attention [Hoens et al., 2012; Wang et al., 2013]. In particular, Wang et al. [2018] have recently shown that in the case of a concept drifting data stream with class imbalance, adaptively correcting for class imbalance and an unknown concept drift simultaneously leads to stronger predictive performance.

2.1.4 A Quantitative Classification

Both the magnitude of a drift and the time it takes are key parts of assessing its effects. In the case of a small drift, very little effect may be felt in a predictive model, while a large abrupt drift might lead to a sudden large drop in predictive performance. Accordingly, one method of measuring drifts is based on their their speed and magnitude. However, in practice, these are challenging to quantify as the speed and magnitude of a drift are based on the distance of the joint distribution after the drift, $p_t(\mathbf{x}, \mathbf{y})$, from the joint distribution before the drift, $p_{t-1}(\mathbf{x}, \mathbf{y})$. Both of these quantities are typically not explicitly known and hard to estimate, especially since the times $t-1$ and t at which the drift begins and ends are often unknown. Consequently, there has been limited investigation into a quantitative assessment of drifts.

One of the first proposals to classify a drift by its magnitude is the severity measure suggested by Minku et al. [2010]. The authors propose two metrics for estimating the severity of a drift based on the percentage of the input space which has its target class changed, while stating the time taken for a drift can be determined as the number of steps it takes for the new concept to completely replace the old concept. While this classification is effective, it is somewhat limited in scenarios where a drift occurs in the features, i.e. $p_t(\mathbf{x}) \neq p_{t-1}(\mathbf{x})$, since the target class may remain unaffected.

An alternative definition of magnitude suggested by Webb et al. [2016] is based on the distance between the old and new concept regimes. The authors also suggest the use of duration, path length, and rate, with the combined use of all four metrics providing a comprehensive method of quantifying drifts.

2.1.5 Other Classifications

The approaches to classifying concept drift laid out in the previous sections have received a lot of attention, with both offering different insights into how to handle cases of concept drift. The classification suggested by Moreno-Torres et al. [2012] is exhaustive, accounting for all possible cases of dataset shift and offering some direction in each situation. However, although the classifications by Minku et al. [2010] and Storkey [2008] are not similarly exhaustive, the approach of predicting with concept drifting data by identifying its properties or cause is highly applicable. Furthermore, in many situations a data stream will experience multiple forms of concept drift, making it a very situational task. Consequently, having the variety of classification regimes we have discussed is very useful, creating the option to model for drifts from multiple viewpoints, each of which can offer helpful insights. This utility has consequently led to more detailed shift classifications [Kull and Flach, 2014; Webb et al., 2016], which focus on developing and combining the previous approaches. In particular, Webb et al. [2016] attempt to provide a unified approach to concept drift, offering a detailed taxonomy of types of concept drift.

It is also worth noting that while the framework laid out by Moreno-Torres et al. [2012] provides a natural and comprehensive outlook which can be applied to either concept drift or dataset shift, the different environments in which they arise has led to the different terminology. The name *dataset shift* arose in a static environment since a shift occurs in the concepts between two distinct datasets; the training and testing set. On the other hand, in a streaming scenario the training and testing sets are dynamic, with incoming data typically belonging to the testing set before transitioning to the training set. Consequently, changes in concept appear as a "drift" over examples, leading to the name *concept drift*. Furthermore, the research focus in both dataset shift and concept drift literature is largely distinct, with dataset shift research tending to focus on applying theoretically motivated corrections to known shifts in the data, whilst concept drift literature tends to be around the design of algorithms which are robust to, or can take reactive approaches to, an unknown shift.

2.2 Corrections for Dataset Shift in a Theoretical Framework

Having seen a number of different classifications of concept drift, we now return to the comprehensive theoretical framework provided by Moreno-Torres et al. [2012]. We visit corrections for dataset shift in this setting, seeing that concept drift algorithms have largely targeted "real concept drift", overlooking the forms corresponding to "virtual concept drift"; covariate shift and prior probability shift. While the corrections we examine are not

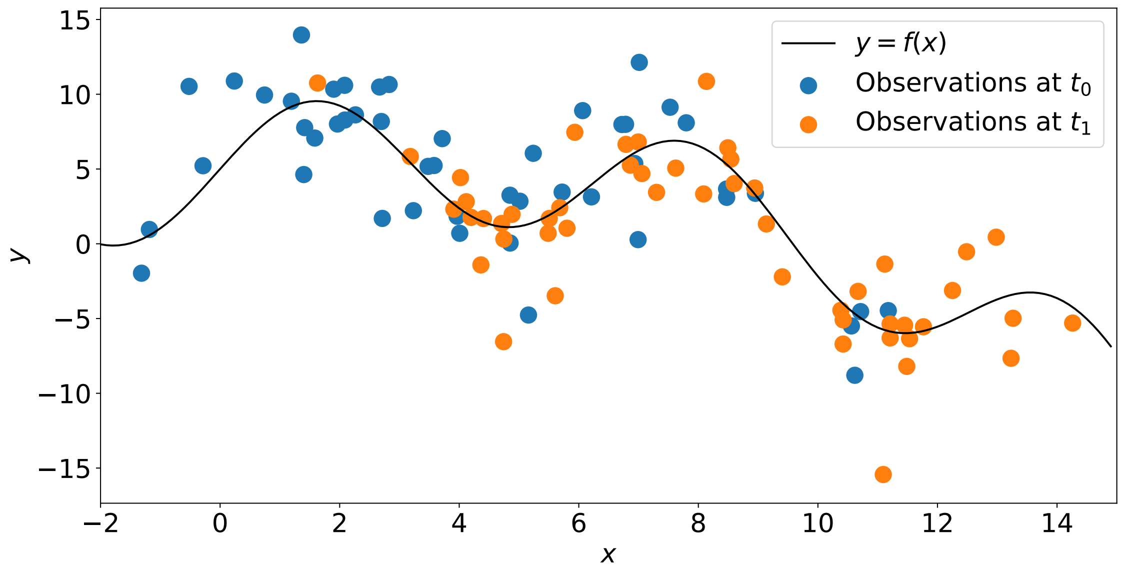


Figure 2.2. A graph of observations from 2 time points, t_0 and t_1, which have undergone covariate shift. While the underlying function $y = f(x) + \epsilon$ has not changed, observations at t_1 tend to have a higher value for x than those at t_0, leading to covariate shift.

directly applicable to scenarios exhibiting concept drift, they provide intuition about the form of different drifts and a direction for future works in a concept drift setting.

2.2.1 Covariate Shift

Covariate shift is the simplest, and perhaps the most common, type of dataset shift, which has led to it gathering large amounts of attention from researchers. In particular, it is often problematic in spam email filtering where the distribution of "spammy" words, used to identify spam emails, tends to differ between training and testing sets. When considering covariate shift, the mapping between $\mathbf{x}$ and $\mathbf{y}$ in the training and testing domains is assumed to remain constant while the distribution of covariates changes:

$$p_{tr}(\mathbf{x}) \neq p_{te}(\mathbf{x}) \tag{2.10}$$

$$p_{tr}(\mathbf{y}|\mathbf{x}) = p_{te}(\mathbf{y}|\mathbf{x}). \tag{2.11}$$

An illustration of this can be seen in Figure 2.2.

The term "covariate shift" was first introduced by Shimodaira [2000], who suggested a method of accounting for covariate shift by re-weighting the training covariate distribution, $p_{tr}(\mathbf{x})$, to that of the testing distribution, $p_{te}(\mathbf{x})$, finding the optimal weights to be $w(\mathbf{x}) = \frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})}$. An example of the desired re-weighting can be seen in Figure 2.3.

However, while the ratio $\frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})}$ is optimal, the method proposed Shimodaira [2000] of directly estimating the distributions $p_{tr}(\mathbf{x})$ and $p_{te}(\mathbf{x})$ is very difficult in practice. If n

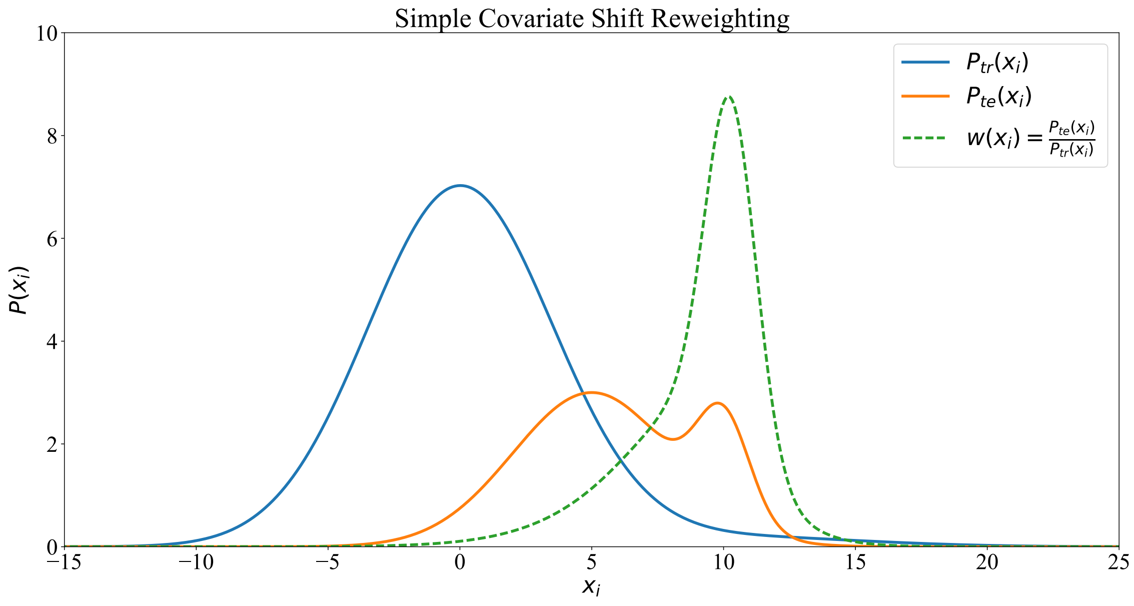


Figure 2.3. Illustrations of a potential training and testing distribution (unnormalised) in one dimension, as well as the desired re-weighting distribution.

examples are necessary to obtain a sufficiently good estimate of the probability distribution in one dimension, roughly n^d examples are needed to obtain an equally good estimation in d dimensions. While this may be possible for lower dimensional data, obtaining enough data for a good estimate of the joint covariate distribution clearly becomes impractical very quickly as d increases. This is important as small errors in the covariate distribution can create large errors in the weights, especially in less populated parts of the training domain where a slight underestimation of the probability can lead to highly inflated weights.

As such, a number of other methods have been proposed to either directly estimate the ratio $w(\mathbf{x})$, or to estimate it implicitly. The earliest proposed methods include Kernel Mean Matching (KMM), Kullback-Leibler Importance Estimation Procedure (KLIEP) and an integrated model [Huang et al., 2006b; Sugiyama et al., 2008; Bickel et al., 2007].

KMM, proposed by Huang et al. [2006b], involves choosing $w(\mathbf{x})$ to minimise the distance between the testing and the weighted training distributions in a Reproducing Kernel Hilbert Space (RKHS) with a universal kernel:

$$\min_{w(\mathbf{x})} \left|\left| \int K_\sigma(\mathbf{x}, \cdot) p_{te}(\mathbf{x}) d\mathbf{x} - \int K_\sigma(\mathbf{x}, \cdot) w(\mathbf{x}) p_{tr}(\mathbf{x}) d\mathbf{x} \right|\right| . \tag{2.12}$$

Subject to $\int w(\mathbf{x}) p_{tr}(\mathbf{x}) d\mathbf{x} = 1$ and $w(\mathbf{x}) \geq 0$, where $K_\sigma(\mathbf{x}, \cdot)$ represents a kernel (with width σ) between $\mathbf{x}$ and any point. These constraints arise as the weighted training covariate distribution is also a distribution so must sum to 1, and as the weights must be non-negative.

The minimum distance solution then gives the ratio $w(\mathbf{x}) = \frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})}$. A common criticism of this method is that KMM suffers from having to define the kernel width arbitrarily, since the training data are biased and thus not suitable for optimising over as a way to estimate σ. This is also problematic in an empirical situation when it comes to estimating an upper bound for $w(\mathbf{x})$ and a margin of error for the difference between the sum of the weights and the number of testing points, ϵ.

The KLIEP method proposed by Sugiyama et al. [2008] avoids this issue. KLIEP works by finding the weights that minimise the Kullback-Leibler (KL) divergence of the weighted training distribution from that of the testing distribution according to:

$$\min_{w(\mathbf{x})} KL\left[p_{te}(\mathbf{x})|w(\mathbf{x})p_{tr}(\mathbf{x})\right] = \min_{w(\mathbf{x})} \int p_{te}(\mathbf{x}) \log\left(\frac{p_{te}(\mathbf{x})}{w(\mathbf{x})p_{tr}(\mathbf{x})}\right) d\mathbf{x}. \tag{2.13}$$

Separating the log and considering only the part dependant on the weights reduces the problem to:

$$\max_{w(\mathbf{x})} \int p_{te}(\mathbf{x}) \log\left(w(\mathbf{x})\right) d\mathbf{x} \tag{2.14}$$

subject to $\int w(\mathbf{x})p_{tr}(\mathbf{x})d\mathbf{x} = 1$ and $w(\mathbf{x}) \geq 0$.

These constraints arise the same way as in KMM and, as previously noted, KLIEP overcomes the difficulty of KMM since the training data is not required in the maximisation procedure, allowing the use of importance weighted cross-validation [Sugiyama et al., 2007] for optimising parameters.

Others have proposed integrated approaches to covariate shift [Storkey and Sugiyama, 2007; Bickel et al., 2007]. These models differ to KMM and KLIEP since instead of estimating weights explicitly, they integrate the estimation with the training of a predictive model. Bickel et al. [2007] consider a binary variable s denoting whether an observation is selected into the training or testing set such that:

$$p_{tr}(\mathbf{x}) = p(\mathbf{x}|s=-1) \qquad \text{and} \qquad p_{te}(\mathbf{x}) = p(\mathbf{x}|s=1). \tag{2.15}$$

Then it can be seen from Bayes theorem that:

$$w(\mathbf{x}) = \frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})} = \frac{p(\mathbf{x}|s=1)}{p(\mathbf{x}|s=-1)} = \frac{p(s=1|\mathbf{x})}{p(s=-1|\mathbf{x})}\frac{p(s=-1)}{p(s=1)}. \tag{2.16}$$

The ratio $\frac{p(s=-1)}{p(s=1)}$ can easily be estimated as the ratio of the number of examples in each set, $\frac{n_{tr}}{n_{te}}$, giving:

$$w(\mathbf{x}) = \frac{n_{tr}}{n_{te}}\frac{p(s=1|\mathbf{x})}{p(s=-1|\mathbf{x})}. \tag{2.17}$$

Consequently, it is possible to estimate the weights using a binary classifier to predict which set an example is in based on its covariates, and then taking the ratio of the probabilities of being in each class. However, as a two-step process, making predictions from a second model using these weights may not provide the optimal solution. Instead, Bickel et al.

[2007] further designed a model which integrates both steps, solving:

$$\min_{\mathbf{v},\mathbf{w}} w(\mathbf{x},\mathbf{v})L(\mathbf{x},\mathbf{w}) + L(\mathbf{x},\mathbf{v}) + \lambda||\mathbf{v}||^2 + \mu||\mathbf{w}||^2 \tag{2.18}$$

where $\mathbf{v}$ and $\mathbf{w}$ are the parameters of the individual weights and predictive models, $L(\mathbf{x},\mathbf{v})$ and $L(\mathbf{x},\mathbf{w})$ are the respective loss functions of both individual models and λ and μ are regularisation parameters. Initially having found that this problem is only locally convex under a log-loss for $L(\mathbf{x},\mathbf{w})$, a later extension found the solution to be globally convex under an exponential loss function [Bickel et al., 2009].

These initial models have provided a basis for further development of covariate shift models and links have been drawn between them. In particular, Tsuboi et al. [2009] study a number of extensions to KLIEP, showing a specific variant with a log-linear model for the weights can be regarded as an extension to KMM.

On the other hand, a notable issue with all of the techniques discussed is that they are heavily dependent on having large amounts of testing data available. While this means they are well suited to situations where models can be trained in an offline manner, it can be problematic in cases with little testing data. This weakness is particularly notable with regards to time series data, in which models constantly need updating and there is often little testing data relative to the amount of training data.

2.2.2 Prior Probability Shift

Prior probability shift occurs in $\mathbf{y} \to \mathbf{x}$ problems, where the causal direction is such that the covariates are caused by the variable of interest. In these situations it is assumed that:

$$p_{tr}(\mathbf{y}) \neq p_{te}(\mathbf{y}) \tag{2.19}$$

$$p_{tr}(\mathbf{x}|\mathbf{y}) = p_{te}(\mathbf{x}|\mathbf{y}). \tag{2.20}$$

An illustration of this prior probability shift be seen in Figure 2.4.

Cases of prior probability shift are very important, especially in situations of medical diagnosis. An extreme example can be seen by considering outbreaks of fever symptoms in both Sweden and Africa. If a training set is taken from Africa, where most cases are identified as being due to typhoid, a prior probability shift will occur when using a testing set from Sweden, as the cause there is far more likely to be a virus instead.

In cases where the shift is known, a number of methods have been proposed to correct for prior probability shift. When specifically considering classification problems, Japkowicz and Stephen [2002] use the term “class imbalance” to denote the same scenario termed

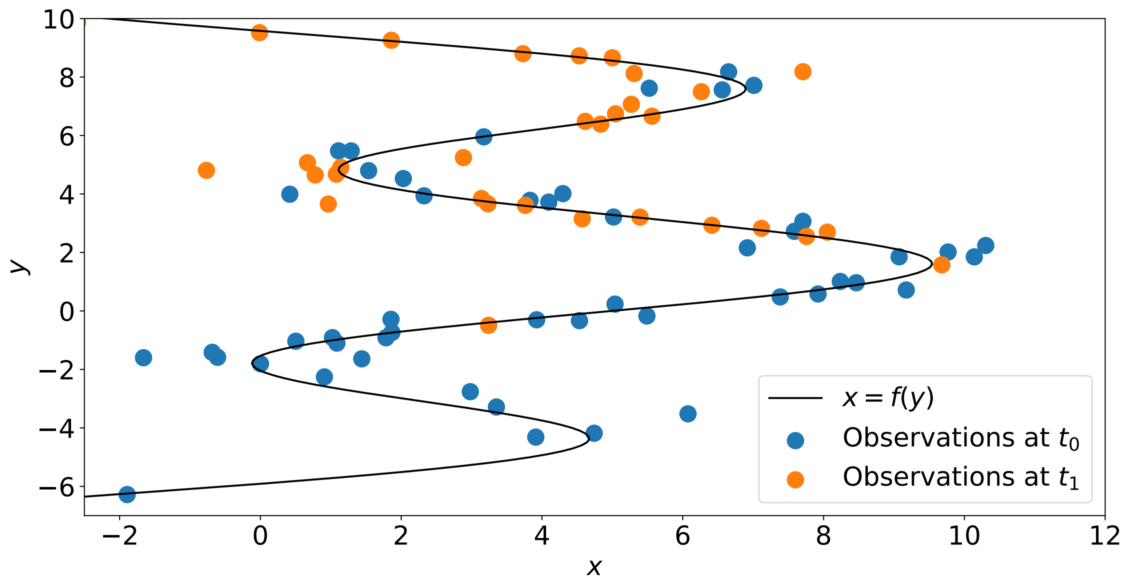


Figure 2.4. A graph of observations from 2 time points, t_0 and t_1, which have undergone prior probability shift. While the underlying function $x = f(y) + \epsilon$ has not changed, observations at t_1 tend to have a higher value for y than those at t_0, leading to prior probability shift.

as "imbalanced data" by Storkey [2008]. Japkowicz and Stephen [2002] then consider three methods to model for class imbalance; over-sampling, under-sampling and cost-modifying. The two sampling methods work similarly, by over-sampling the rare class or under-sampling the common class respectively. The cost-sampling method works by re-weighting the cost of misclassification so that misclassifying a common class as a rare class incurs a higher cost. Other methods for handling class imbalance have also been developed, such as multiple resampling [Estabrooks et al., 2004] or the use of support vector machines [Akbani et al., 2004]. Storkey [2008] suggests a more general approach, applicable in both the classification and regression case: correcting for prior probability shift may be done via re-weighting the training examples according to $w(\mathbf{y}) = \frac{p_{te}(\mathbf{y})}{p_{tr}(\mathbf{y})}$, analogous to the methods used in handling simple covariate shift.

However, in many situations no information about the shift is known and the testing data is unlabelled, making prior probability shift much harder to identify or correct for. In this case, as $p_{te}(\mathbf{y})$ unknown, Storkey [2008] suggests using the fact that $p_{te}(\mathbf{x}|\mathbf{y})$ can be estimated from the testing data and specifying a prior distribution for $p_{te}(\mathbf{y})$ based on the modeller's understanding of the problem. This allows the posterior distribution for $p_{te}(\mathbf{y})$ to be estimated from the testing covariates, and the posterior can then be used to re-weight predictions obtained from a model fitted using the training data. It should be noted that this method also comes with a few caveats: it relies on the modeller having enough information to estimate a prior distribution, and may fall through if it is not possible to perform the integrals required to estimate the posterior.

Another method, suggested by Zhang et al. [2013], introduces the assumption that there is only one possible distribution for $p_{te}(\mathbf{y})$ that together with $p_{tr}(\mathbf{x}|\mathbf{y})$ leads to $p_{te}(\mathbf{x})$. Along with Equation 2.20, this means that a set of weights for re-weighting, $w(\mathbf{y}) = \frac{p_{te}(\mathbf{y})}{p_{tr}(\mathbf{y})}$, can be found by minimising:

$$\min_{w(\mathbf{x})} \left|\left|\mathbf{D}\left(p_{te}(\mathbf{x}) - \int p_{tr}(\mathbf{y})w(\mathbf{y})p_{tr}(\mathbf{x}|\mathbf{y})d\mathbf{y}\right)\right|\right| \tag{2.21}$$

where $\mathbf{D}$ is a distance measure. Adapting the kernel mean matching approach given by Gretton et al. [2006] then allows Equation 2.21 to be solved for $w(\mathbf{y})$ in much the same way as with the KMM in Section 2.2.1.

2.2.3 Concept Shift

Concept shift, or real concept drift, is usually the hardest type of drift to work with. In both simple covariate and prior probability shift, methods focus on re-weighting examples to account for the shift. On the other hand, in the case of real concept drift, a much more flexible approach is needed due to the broad variety of ways a drift can occur.

As previously shown, real concept drift is defined to occur in one of the following two situations:

$$p_{tr}(\mathbf{y}|\mathbf{x}) \neq p_{te}(\mathbf{y}|\mathbf{x}) \tag{2.22}$$

$$p_{tr}(\mathbf{x}|\mathbf{y}) \neq p_{te}(\mathbf{x}|\mathbf{y}) \tag{2.23}$$

in $\mathbf{x} \rightarrow \mathbf{y}$ and $\mathbf{y} \rightarrow \mathbf{x}$ problems respectively. In this, the term “concept” is taken to mean the distribution of a single target and feature, $p(y|x)$, while the entire set of concepts or “concept regime” is taken as referring to the joint distribution $p(\mathbf{y}|\mathbf{x})$. An illustration of concept shift can be seen in Figure 2.5.

There are two main scenarios in which real concept drift is encountered; that in which information is known about the drift in the relationship, and that in which there is no knowledge. In the case where the drift in the relationship is known, the problem generally becomes much simpler and merely involves updating the mapping to account for the drift. However, in the case where nothing is known, the new relationship must be learnt. This situation is far more common and occurs in many modern day scenarios, such as with the weather forecasting or in predicting consumer trends. In fact, real concept drift arises in most settings with dynamic environments, as data is generated over time, which has led to it attracting a lot of attention in the streaming data community.

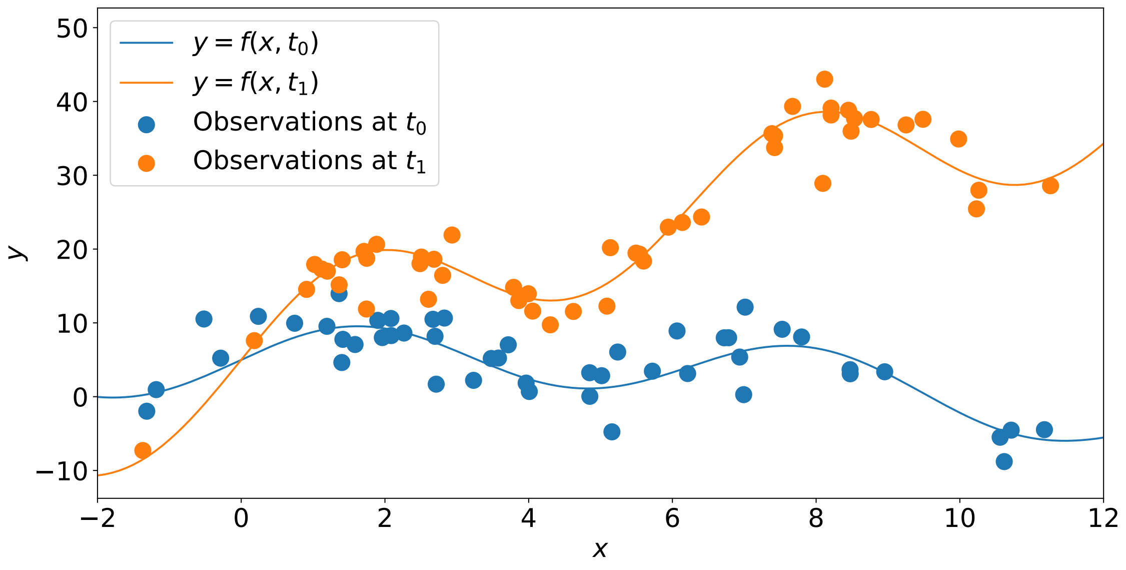


Figure 2.5. A graph of observations from 2 time points, t_0 and t_1, which have undergone concept shift. The underlying function $y = f(x) + \epsilon$ has changed between the time points, leading to concept shift.

2.3 Summary

In this section we have introduced a number of established methods for classifying types of dataset shift and concept drift, providing a discussion on the advantages and disadvantages of each. We have seen that the complexity of the topic makes it had to choose a uniform classification for types of concept drift, but that the insights of each different classification can be useful when constructing a predictive model. We have also given an overview of methods for handling different types of dataset shift based on their theoretical classification. We have placed the main topic of interest in this thesis, real concept drift, within this framework and discussed its importance.

Chapter 3

Modelling with Concept Drifting Data

In Chapter 2, we have introduced a number of different classifications of concept drift which have arisen from either a “dataset shift” or a “concept drift” framework. For the remainder of this thesis, we now work in a streaming scenario, focussing entirely on concept drift. As many scenarios in which concept drift occurs are considered to be data streams, we now provide a brief description of the streaming setting in which we will be working.

Streaming data is considered to arrive sequentially in time, where at each time point t a vector of input data, $\mathbf{x}_t = (x_{t,1}, ..., x_{t,D})$, arrives and must be used to predict targets $\mathbf{y}_t = (y_{t,1}, ..., y_{t,M})$, which are observed later. The form of the input data may be numerical, categorical, or ordinal, while the target can be either numeric or categorical, corresponding to regression or categorization problems respectively. In this setting, concept drift can be said to occur if we have a time-dependant joint distribution $p_t(\mathbf{y}|\mathbf{x})$, since with $\mathcal{X} \in \mathbb{R}^D$ and $\mathcal{Y} \in \mathbb{R}^M$ we will be learning a mapping $\mathcal{X} \to \mathcal{Y}$ on data up to time $t-1$, yet aiming to predict at time t.

In this streaming setup, it is typically assumed that nothing is known about the form of any drift, or even if a drift is occurring. This necessitates reactive approaches to concept drift, where algorithms must be able to first identify and secondly adapt to changes in the data stream. This can be a complicated process and we now detail some of the major challenges and common approaches to modelling with concept drifting data.

3.1 Concept Drift Setting and Major Challenges

Due to its broad nature, there are a number of issues which must be addressed when creating a model in the presence of concept drift:

- Dynamic concepts are associated with temporal data, which adds an additional dimension for consideration.
- Detecting a drift can be challenging, especially amongst noisy data.
- Model memory means that using a model trained on data pertaining to outdated concepts will impede the ability of a model to make predictions in the current concept regime.
- Due to the large fast paced nature of streaming data, modelling for any drift must also account for time and computational constraints.

Time-based data often needs careful consideration. In a concept drift setting, data is normally assumed to be received in a consecutive and continuous stream over time, with each example, $(\mathbf{x}_t, \mathbf{y}_t)$, possibly following a different concept regime to the last. This dynamic environment necessitates modelling in an online situation, and the frequency of data arriving must be taken into consideration. Data may arrive in batches (multiple examples at once) or incrementally (one example at a time). Furthermore, as there is a continuous stream of incoming data, computational constraints may arise when there is too much to hold in RAM. Consequently, many models aim to operate in a single-pass fashion, where each example is only seen once and then discarded. In actuality, this goal very difficult to satisfy, and examples are often kept for a certain period of time and used for validating a model or for the training of an alternative option if a drift is detected.

The biggest challenge in concept drift is that of model memory, and has been noted for a long time, with Maloof and Michalski observing the issue in a concept drift setting in 1995. All concept drift models must account for this issue, and most methods revolve around finding ways to forget parts of the model trained on data generated under outdated concept regimes, thus effectively keeping a model trained only on data from the current concept regime. However, this can often prove troublesome since it is hard to identify outdated aspects of a model. Furthermore, forgetting can also be detrimental in certain circumstances, such as cases of recurring concepts, where previously redundant data can, sometimes suddenly, become relevant again.

3.2 Learning in the Presence of Concept Drift

Due to the differing nature of the types of concept drift and the diverse challenges a model must overcome, a number of different approaches have been developed over time. Many methods revolve around example selection, example weighting or ensemble methods [Tsymbal, 2004]. Other approaches have also been suggested, such as online algorithms. Furthermore, all of the previous methods have also been combined, with online algorithms

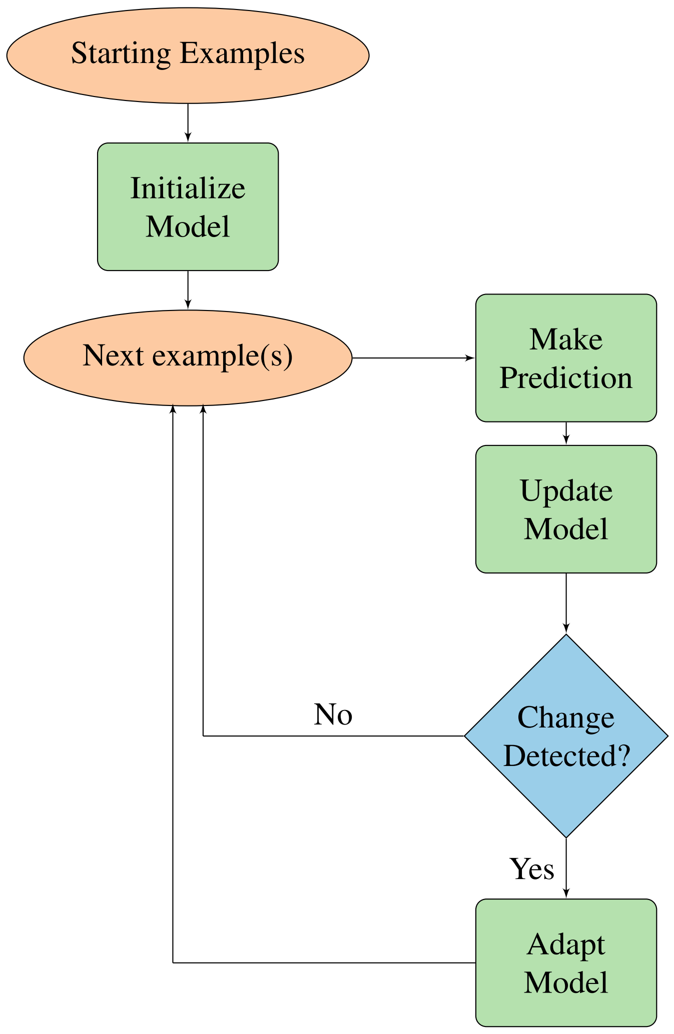


Figure 3.1. The modelling process used in cases of concept drift.

and windowing commonly being used for training the base classifiers in an ensemble [Brzezinski and Stefanowski, 2014b].

In general, concepts drift algorithms follow a similar format. Initially, starting examples are assumed to be stationary and are used to build a model. As new example(s) arrive, they are passed to the model and a prediction for each is made, after which the example trained on to update the model. A change detector then looks for concept drift and, upon detection of a drift, the model is adapted accordingly. The process then continues with the next example(s) being passed to the model. This process continues until the end of the stream is reached. A flow chart detailing this approach can be seen in Figure 3.1. It is worth noting that while the role of change detection has been depicted here as a distinct step for clarity, it is often integrated with other modelling steps. Furthermore, rather than depending on a dedicated change detection method, approaches also exist in which drifts are handled implicitly, without explicit detection.

One statistic of particular importance throughout concept drift literature is the Hoeffding bound [Hoeffding, 1963]. The Hoeffding bound is able to provide a probability, and thus confidence intervals, on the deviation of the sum of independent random variables from the expected value. It is typically used to state that, with probability $1 - \delta$, the confidence interval ϵ for the mean of N random variables with range R, is given by:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2N}}. \tag{3.1}$$

We will see in the rest of this chapter that the Hoeffding bound has been applied in a number of ways throughout the literature, at multiple different stages of the modelling process we have just described. For example, it has been used in the change detection step to detect whether the mean of incoming data has shifted, and thus whether drift has occurred; and also in the model update step, to provide a guarantee on the splitting performance of incrementally trained trees relative to those trained with all the data available.

One assumption made in all concept drift models, which is usually not made explicit, is the assumption of periods of approximate stationarity in the concept regime. Almost all algorithms for concept drift benchmark against periods of data, or models trained on periods of data, when detecting and correcting for drifts, and the data over these period is assumed to be effectively stationary. In practice, this is not a major constraint and makes sense, since without this assumption there is no signal to be found in the data. This can be seen by considering a data stream which exhibits different and new concept regimes for each subsequent observation; the data is effectively just noise.

3.2.1 Change Detection

Many studies have focused on change detectors, which play an important role when dealing with concept drift. While they do not directly tackle the prediction problem, they are often components in concept drift prediction algorithms, since in order to appropriately adapt a model it is still necessary to explicitly know whether concept drift occurs and when. This can be a challenging task and has motivated research into change detectors, which aim to detect whether there is any evidence of concept drift in a dataset.

A number of different styles of change detector have been developed, primarily divided between detecting changes in the underlying data and changes relating to the learner. However, regardless of the style, change detection typically occurs in 3 stages:

1 Initially, new data is retrieved from the stream, which may then be used to update a model.

2 A test statistic is calculated on the new data/updated model and compared with the same statistic calculated on older data/non-updated model.

3 If the new and old test statistic are sufficiently distinct, a change is determined to have taken place.

Data-Based Change Detectors

Data based change detectors attempt to detect concept drift by establishing a distance measure to quantify the dissimilarity between new and old data. If the distance is sufficient, the detector will flag a change as having occurred. This approach enables concept drift to be detected in the data, directly attempting to establish if there are differences between $p_{t-1}(\mathbf{y}|\mathbf{x})$ and $p_t(\mathbf{y}|\mathbf{x})$.

Kifer et al. [2004] pioneered work on detecting changes in the underlying data through the use of two windows of data, termed a "reference window" and a "current window". They compare the distribution of the data in the windows using a distance measure, and identify a drift if the distance exceeds a threshold which is determined based on Chernoff bounds. This method is attractive as it allows for both detection and estimation of the magnitude of a drift.

Other approaches have also focussed on the use of windows of data for change detection. Both Dasu et al. [2006] and Sebastião and Gama [2007] similarly compare between two windows of data, detecting change based on the Kullback-Leibler (KL) divergence, an entropy based measure of distance.

Following this, further window based methods have been developed. As part of the Adaptive Windowing Algorithm (ADWIN), Bifet et al. [2007] compute the window size based on a change detector. Their algorithm splits the base window, W, into two sub-windows, W_0 and W_1, at all possible points and compares the means of each window. If the windows are "large enough" and the means of the data in the windows are "distinct enough", it is classed as a drift and the older window is discarded. Whether the windows are large and distinct enough is decided based on an application of the Hoeffding bound. The threshold for change, ϵ_{cut}, is then given by:

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4n}{\delta}} \tag{3.2}$$

where m is defined the harmonic mean of n_0 and n_1: $\frac{1}{1/n_0+1/n_1}$, n, n_0 and n_1 are the number of examples in W, W_0 and W_1 respectively, and δ is a bound on the false positive rate. ADWIN is a popular approach, and has been used in a number of subsequent algorithms [Bifet et al., 2009b,a; Gomes et al., 2017b]. However, despite the effectiveness of these windowing methods for tracking change, they are not suitable for every situation as they can be computationally expensive due to the need to evaluate changes over many possible windows.

While we are working in a supervised learning scenario, it is worth noting that a number of data based change detectors designed for unsupervised scenarios are also available [Spinosa et al., 2007; Faria et al., 2013; Sethi and Kantardzic, 2017]. These methods often rely on novelty detection and clustering the data in order to detect changes.

Learner-Error Based Change Detectors

Instead of directly testing for changes in the data, learner-error based methods test for change by monitoring the learner error. By basing the approach on the learner, the detector is more likely to capture nonlinear and complex drifts than data based detection methods, which typically use straightforward distance metrics such as the data means. However, it also assumes that a model is correct and that if the error is constant after a change, it implies that there is no need to update a model, which may not always be the case.

Unlike data oriented detectors, most error based change detection strategies do not need a window of examples, and focus either on sequential tests or tracking statistical quantities in an online fashion. One of the only exceptions to this is the approach suggested by Harel et al. [2014], who obtain statistics about the loss distribution by resampling from a window of data. They test for change by comparing the loss of the ordered window, with the average loss obtained from different shuffled windows, determining that there has been a change if a hypothesis test suggests that the losses are different.

Another learner-error based change detector that is only applicable to classification problems has been suggested by Gama et al. [2004]. They proposed the Statistical Process Control (SPC) based Drift Detection Method (DDM), which trains a learner and makes a prediction, treating the error as a random variable from a Bernoulli trial. Consequently, the errors should follow a binomial distribution, with the prediction for example i being false with probability p_i and standard deviation:

$$\sigma_i = \sqrt{\frac{p_i(1-p_i)}{i}}. \tag{3.3}$$

As the distribution should be static unless concept drift occurs, it can then be approximated to a normal distribution with confidence bounds on the false positive rate given by $p_i \pm \alpha \cdot s_i$, providing there are enough examples. As many algorithms expect the error to decrease with additional examples, change is then determined to be in the case when:

$$p_i + \sigma_i \geq p_{min} + \alpha \cdot \sigma_i \tag{3.4}$$

where p_{min} is the lowest value of p_i observed so far. α can then be chosen based on the desired confidence level, with the authors suggesting $\alpha = 2$ and $\alpha = 3$ to give 95$\%$ and 99$\%$ confidence of drift occurring.

It is important to note that the DDM requires sufficient data, as to approximate to a normal distribution they need at least 25 to 30 examples. However, a notable failure for DDM is in cases of gradual drift, where the changes may pass without triggering the alarm level. To rectify this, Bifet et al. [2006] proposed an extension of DDM: the Early Drift Detection Method (EDDM). The EDDM is very similar to the DDM, but is based on the distance between errors as opposed to simply a count of them. Other extensions to the DDM have also been suggested. The Learning with Local Drift Detection (LLDD) [Gama and Castillo, 2006] is designed for partition learners and works similarly to the DDM, except errors are monitored separately in each subspace, rather than at the learner level. This allows for local drift detection in each subspace. On the other hand, the recently proposed Drift Detection Method based on Hoeffding's inequality (HDDM) [Frías-Blanco et al., 2015] uses the Hoeffding bound to provide a theoretical guarantee that a drift has occurred.

The Exponentially Weighted Moving Average (EWMA) for Concept Drift Detection (ECDD) algorithm [Ross et al., 2012] takes a similar approach. It calculates an exponentially smoothed mean error E_t at time t by downweighting older errors by a factor of $1 - \lambda$, giving the error:

$$E_t = (1 - \lambda)E_{t-1} + \lambda e_t \tag{3.5}$$

where e_t is the error on the prediction at time t. This error is monitored, and a change is flagged when a change in E occurs such that

$$E_t > E_0 + \alpha\sigma_{E_t} \tag{3.6}$$

where α again represents the limit which E_t can deviate by before a change is determined to have occurred.

Cumulative sum (CUSUM) based algorithms [Page, 1954], are another popular error-based approach to drift detection [Basseville and Nikiforov, 1993; Ahmed et al., 2008]. They are lightweight and memoryless tests that continuously monitor a metric, usually the predictive performance, and signal a change if there is a significant deviation in the mean of the metric. A further interesting sequential test that is actually not memoryless is the Page-Hinckley (PH) test [Mouss et al., 2004]. This test keeps track of the sum of the deviation of the metric M from its mean at each point:

$$m_T = \sum_{t=1}^{T} M_t - \bar{M}_t - \alpha \tag{3.7}$$

where α is an acceptably small deviation to overlook. The test then flags a change if the sum of the deviations, m_t, exceeds the minimum mean of the metric observed by a reference amount λ:

$$m_T > \min\{m_1, ..., m_T\} + \lambda. \tag{3.8}$$

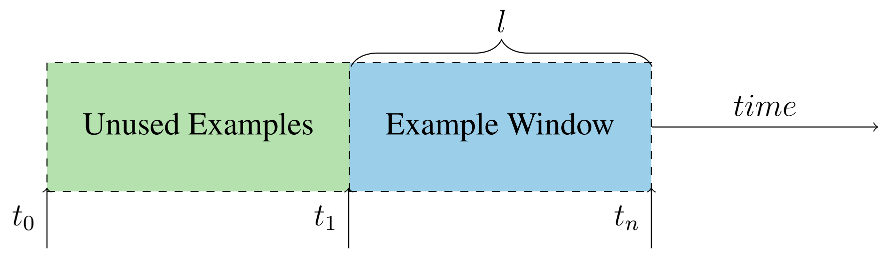


Figure 3.2. An example of a simple windowing procedure. At time t_n, examples from time t_0 until time t_n are available, but a window of examples from time t_1 to t_n is used. The window is of a fixed length l, and as new examples become available, the oldest ones are removed from the window.

3.2.2 Example Modification

One method of accounting for concept drift in the data stream is through example modification. By selecting only the data corresponding to the current concept regime, a stationary model trained on the selected data should be able to make accurate predictions. Among the literature, there have been two main ways to do this explicitly: windowing and weighting.

Windowing

Perhaps the most traditional method for handling concept drift is windowing. As a form of example selection, windowing tries to select a window of examples relevant to the current concepts. One of the oldest algorithms to take this approach was first suggested by Widmer and Kubat [1992], who include windowing as part of the Floating Rough Approximation (FLORA) algorithm. In this algorithm a specific window length, l, is chosen and the most recent l examples are used for training a model. As more examples become available, the oldest ones are then removed from the window. An example of this windowing method can be seen in Figure 3.2.

The idea of windowing appears to address some of the key challenges in dealing with concept drift. Having a window allows older data relating to outdated concepts to be removed and allows for faster modelling due to the reduced number of examples being considered. On the other hand, this gives rise to certain issues: most apparently, there is no obvious choice for the window length, l. In addition to this, older data may still be of value, and in the case of recurring concepts it seems natural to want to use data from the previous time a concept occurred, which may well fall outside the current window.

Further approaches to tackle these issues are also introduced by Widmer and Kubat in later versions of the FLORA algorithm. FLORA2 attempts to heuristically adjust the window length based on whether a drift is suspected, while FLORA3 tracks old concepts, allowing the window to be extended in the case of recurring concepts. Furthermore, a FLORA4 version is also suggested to improve performance in the presence of noise. Other methods of windowing have also been provided with the aim of better covering these issues. One method by Lazarescu et al. [2004] suggests the Competing Windows Algorithm (CWA), which uses 3 different sized and adaptable windows to better account for concept drift by examining the persistence and consistency of any drift.

All of the windowing techniques described so far have a common problem: they depend on heuristics to set the window length. To overcome this, Klinkenberg and Joachims [2000] suggest a method which selects a window size without parametrisation. Their study focuses on Support Vector Machines (SVMs), training SVMs on multiple different length windows of data, and selecting the window which minimises the generalisation error on new examples. Furthermore, as described in Section 3.2.1, Bifet et al. [2007] designed the ADWIN algorithm, which recomputes the window length in an online manner based on changes in the underlying distribution of the data. However, these methods both require models or calculations at multiple different window lengths, which can also be quite computationally costly. As a result, later studies have designed other, less expensive, approaches for certain scenarios, such as Kuncheva and Žliobait [2009] who proposed a theoretically motivated window resizing method designed for handling abrupt drifts.

Weighting

An alternative approach to handling concept drift by example modification is example weighting. It can be viewed as a more general case of windowing, where instead of weights being either 0 or 1, as is the case with windowing, they can take on a continuous range of values. Koychev [2000] presents a method of weighting which gradually forgets previous examples, suggesting that partial memory, as in many windowing situations, may not be the most effective way of reacting to concept drift. This is because under some forms of drift, such as with recurring concepts, older information can be valuable. Consequently, Koychev suggests using a gradual forgetting function, choosing a linear forgetting function for tests. However, while this does retain some older information, linear forgetting still does not effectively account for recurring concepts. Furthermore, the speed of forgetting is defined by a parameter that must still be chosen, and the choice of forgetting function is still arbitrary.

In early tests comparing a number of example selection and example weighting methods, Klinkenberg [2004] trialled both exponential and sigmoidal weighting functions but eventually found example selection methods such as ADWIN to be particularly effective. However, later research has still used example weighting methods, such as Zhang et al. [2008]. In particular, Alippi et al. [2009] recommend a method of choosing weights based on how similar an observation is to those expected under the most recently observed concept regime.

3.2.3 Online Algorithms

The use of online algorithms is another popular method of modelling in the presence of concept drift. Unlike example modification approaches, online algorithms do not explicitly change the set of training examples, instead typically operating in a single-pass fashion and incrementally updating the model. This enables the algorithms to adapt to change by either forgetting or adapting parts of the model which no longer appear to perform as well, giving rise to the term “adaptive base learners” in the literature. These algorithms tend to have limited memory of the data and focus on updating the model using only the most recent example. As a consequence of this style of learning, most online algorithms have the advantage of taking only a short time to update, allowing them to be used in situations where predictions are needed very quickly. Furthermore, the limited memory property allows older examples to be discarded, reducing the computational requirements in situations with large amounts of data.

Tree-Based Learners

Decision trees are a very popular approach to online learning, and offer a number of advantages:

- Decision trees offer a clear and intuitive method of modelling, leading to easy algorithm design.
- Incremental training is possible, allowing trees to be constructed from a single pass of the data stream. This in turn leads to reduced modelling times and memory consumption, compared to example modification.
- As they partition the feature space, local drifts in the data may be able to be isolated to a specific subtree. Subsequently, it is comparatively easy to modify or prune subtrees affected by drift.

Perhaps the most influential algorithm in this category is the Very Fast Decision Tree (VFDT), developed by Domingos and Hulten [2000] for use in mining data streams. The VFDT algorithm uses Hoeffding trees as a base model, named due to their use of the Hoeffding bound to provide a guarantee that the incrementally constructed splits in the trees are sufficiently similar to those that would be obtained in a scenario in which all of the data is available. Domingos and Hulten show that this application of the Hoeffding bound allows decision trees to be grown in an online manner, with the number of examples used to grow the tree providing a level of confidence that the Hoeffding tree will choose the same splitting attribute as a tree grown on all the data.

Although it has the advantage of training very quickly, the original VFDT algorithm is unable to adapt to new concepts quickly as it can only forget older concepts by diluting them out with a large number of examples from newer concepts. Consequently Hulten et al. [2001] extended this work using the Concept-adapting Very Fast Decision Tree (CVFDT). This framework keeps a sliding window of past examples and creates an alternative subtree when the concepts appear to change, which is substituted for the original if it becomes more accurate. This effectively keeps the tree as if it were VFDT trained on a moving window of examples, but maintains a faster learning speed as the tree does not need to be completely regrown when concepts change. Despite this, the CVFDT still faces shortcomings as the length of the window and the number of examples between updates are both arbitrary and must be user defined.

Bifet and Gavald [2009] suggested further extensions to the CVFDT model under the names Hoeffding Window Tree (HWT) and Hoeffding Adaptive Tree (HAT). The HWT is designed to remove the need for a number of arbitrary parameters in the CVFDT; the number of examples between checks for growth of the tree or for concept drift, and the number of examples needed to validate a new tree. This is achieved by checking each example as it arrives and then updating trees instantly if a change is detected. Furthermore, older trees are also replaced instantly if a new tree is proves to be more accurate than the original.

The HAT furthers the idea of eliminating arbitrary parameters by removing the need to specify a specific window length. It uses the ADWIN algorithm described by Bifet et al. [2007] at each node in the Hoeffding tree. The relevant statistics are estimated at each node, thus overcoming the need for a fixed length window. Furthermore, it has been shown that under appropriate assumptions the HAT has a guarantee of growing the same tree as a VFDT only exposed to the new concept regime would. However, the HAT is also more time consuming, and tests found it to be roughly 4 times slower than its predecessor algorithms [Bifet and Gavald, 2009].

Another stream of development involves the concept of Option trees [Kohavi and Kunz, 1997]. Unlike standard decision trees, Option trees differ due to their use of option nodes, which act as multiple different splitting nodes, as shown in Figure 3.3. In an option tree, there are multiple non-exclusive splitting conditions at an option node, and examples follow every path for which they satisfy the splitting condition at the option node, resulting in multiple predictions which are then combined through averaging. Pfahringer et al. [2007] combined Option trees with Hoeffding trees to form Hoeffding Option Trees (HOT) which, in the case that a new split is found to be better than an existing one, allow both splits to kept as an option node. As the HOT structure represents multiple possible trees it can be considered as an ensemble, and an extension from Bifet et al. [2009b] suggests the Adaptive Hoeffding Option Tree (AHOT), which keeps an estimate of the error at each leaf to use in weighting the split at each node.

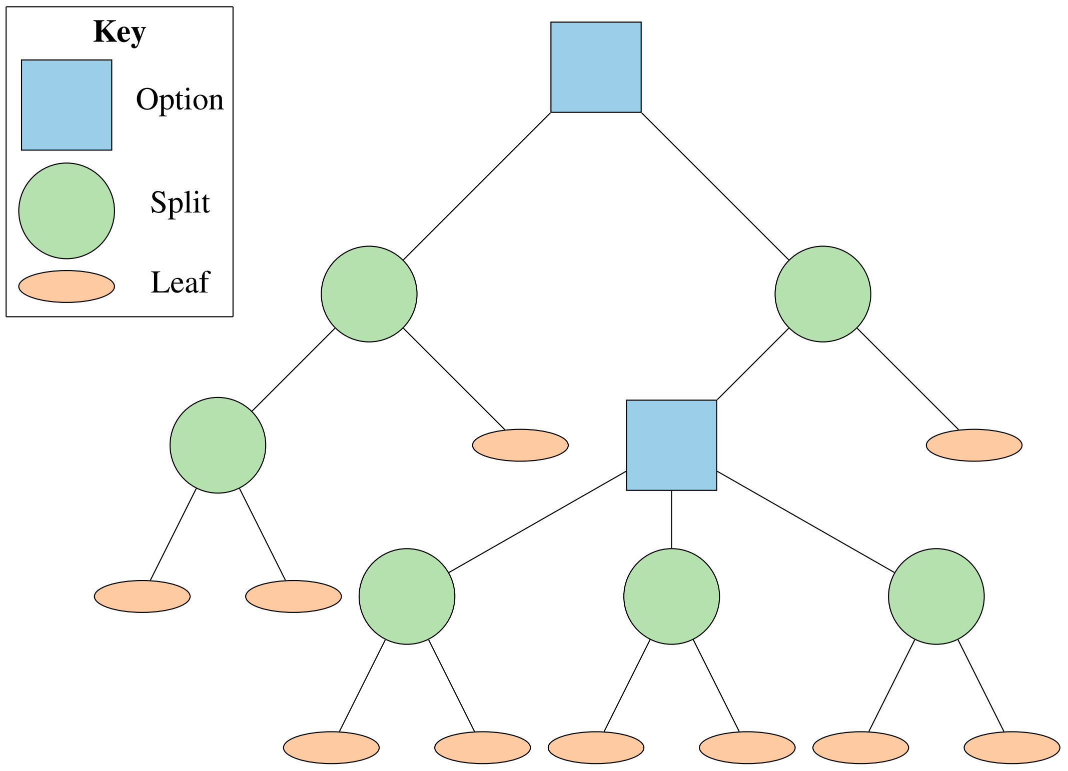


Figure 3.3. An example option tree structure. When an example is passed down the tree, it follows all possible paths for which it satisfies the splitting condition at each option node, resulting in multiple different predictions which are then combined through averaging.

While the trees suggested so far are all based on the VFDT, Manapragada et al. [2018] revisit and adapt the original VFDT splitting mechanism. They suggest the Extremely Fast Decision Tree (EFDT) algorithm which initially splits in a similar fashion to the VFDT, but when the information gain of the best split is greater than that of not splitting, rather than when it is greater than that of the second best split. The EFDT then periodically revisits created splits, and if a different split is detected to outperform the current split, it replaces the current split. The authors suggest that this approach leads to faster learning than the VFDT due to less restrictive initial splitting criteria. Furthermore, they claim the EFDT is also robust to the presence of concept drift, likely due to the replacement of splits.

While most studies focus on classification trees, more recently an effort has been made to address regression problems. Ikonomovska et al. [2011b] pioneered the Fast Incremental Model Trees with Drift Detection (FIMT-DD) algorithm as the first major contribution for use in regression problems with concept drift. This method starts with a fixed number of examples and uses them to grow a tree. With the starting examples, the best split for each attribute is found and the splitting attributes are then ranked. If the splitting criterion is satisfied, the highest ranked attribute is used to make a split, and the process is repeated to create a tree. Similar to the classification case, these splitting decisions are made based on the Hoeffding bound. As further examples come in, they are considered incrementally and passed down the tree, with the results being used to update change detection tests. When a change is detected, a new tree is grown simultaneously and may eventually replace the original tree after meeting the swapping criterion. In a comparison on datasets without concept drift, FIMT-DD does not perform as well as other some other data-stream algorithms [Ikonomovska et al., 2011b]. However, it is faster than any other considered method and in the case of datasets with concept drift, it significantly outperforms other methods [Ikonomovska et al., 2011b].

An extension of FIMT-DD, the On-line Regression Trees With Options (ORTO), has also been proposed [Ikonomovska et al., 2011c]. ORTO trains trees similarly to FIMT-DD, but also introduces option nodes. The tree differs from the FIMT-DD tree when other splitting attributes that are almost equally as discriminative as those currently used at a node are detected. In this scenario an option node is created with both sets of splitting attributes present. As option trees lead to multiple predictions from the tree, and a number of rules for combining the final predictions were considered, with the best being simply averaging the predictions. This makes the option tree somewhat akin to an ensemble approach in which a new tree is added to the ensemble whenever a new similarly discriminative split is detected. In a comparison on multiple datasets, this method, known as ORTO-A, outperformed FIMT-DD in every situation.

Variants for a multi-target scenario have also been presented. The Fast Incremental Trees for Multiple Targets (FIMT-MT) algorithm [Ikonomovska et al., 2011a] and the more recent incremental Structured Output Prediction (iSOUP) Tree [Osojnik et al., 2016] both propose a global approach to multi-target regression. Both approaches replace the original splitting criterion in the FIMT-DD with an inter-cluster criterion, aiming to include information about target dependence in splitting decisions. Furthermore, the iSOUP approach also extends the original problem setting and can be used for multi-label classification.

All of the online algorithms described here take advantage of the Hoeffding bound [Hoeffding, 1963] to guarantee the performance of splits relative to those of that would be found in a stationary scenario. However, in a theoretical analysis, Rutkowski et al. [2013] found that the Hoeffding bound is unsuited to the problem of concept drift and instead offer a tree based on the McDiarmid's bound. Although the implications of this analysis call in to question the suitability many of the current approaches to concept drift, the authors did not make a comparison of their method against others, meaning there is currently no empirical evidence of this.

Other Learners

While many approaches are based on decision trees, other adaptive base learners have also been proposed. The Adaptive Model Rules (AMRules) approach from Duarte et al. [2016] trains a set of concept-adapting rules for a regression problem by choosing splitting rules that minimize the variance reduction, similar to tree-based methods. However, the modularity of rule based methods means that the result is more interpretable relative to a decision tree.

Another recent alternative to decision trees is the Dynamic Extreme Learning Machine for data stream classification (DELM) algorithm proposed by Xu and Wang [2017]. The DELM uses an Extreme Learning Machine (ELM) [Huang et al., 2006a] as a basic learner, while training a double hidden layer structure to improve the performance of the classifier. In the event drift is detected, further hidden layer nodes are added to the ELM. If the effect of a drift persists and the magnitude exceeds acceptable levels, a new classifier is trained. While the authors find this approach is able to improve classification accuracy and swiftly react to drift, it also suffers from a loss of interpretability making it harder to design targeted drift adaption mechanisms.

3.2.4 Ensemble Methods

Ensemble classifiers are a very popular and well-suited approach to concept drift, largely due to their flexibility and effectiveness, which allows them to work well with many different types of drift. They often focus on using other concept drift methods to train base classifiers, allowing them to train quickly and without using too much memory. Typically, ensembles are more robust to noise than single classifiers due to averaging in the voting procedure, which leads to better predictive accuracy.

One of the first ensembles designed for concept drifting data is the Streaming Ensemble Algorithm (SEA) proposed by Street and Kim [2001]. SEA divides the incoming data into sequential batches and trains an individual classifier on each batch, effectively taking a windowing approach. The classifiers are then added as members of a fixed-size ensemble. In the event that the ensemble is already full, the worst performing classifier on the newest batch of data is dropped. The class prediction is then obtained from a majority vote of the individual classifiers in the ensemble. Variations such as accuracy-weighted voting were also considered, but were not found to lead to improvements. The SEA algorithm performs similarly to a single tree, but is useful in cases of concept drift as it recovers from drifts faster.

A similar approach to SEA, developed by Wang et al. [2003], is the Accuracy Weighted Ensemble (AWE). Like SEA, AWE divides incoming data into batches and trains a classifier on each. The weight of each classifier is then found using an estimate of the expected prediction error on test examples. The most recent batch of training data is expected to be most similar to the testing data, so the mean squared error (MSE) of each classifier's predictions is evaluated on that batch and compared to the MSE of a random prediction. Weights are then based on the improvement of the classifier over the random case, and classifiers with no improvement are discarded. Furthermore, to prevent infinite growth of the ensemble, a limit is placed on the maximum number of classifiers in the ensemble, and if it is reached the classifier with the lowest MSE is discarded.

While both SEA and AWE are designed for concept drift environments, they also require many parameters which must be arbitrarily defined, such as the maximum number of ensemble members and the size of each batch. Kolter and Maloof [2003] note specifically that as SEA can only remove older concepts by gradually removing ensemble members, it can be very slow to adapt to new concepts if many members of the ensemble were trained on a previous concept regime. To rectify this, they propose the Dynamic Weighted Majority algorithm (DWM). The DWM is based on the Weighted Majority algorithm suggested by Littlestone and Warmuth [1994], and removes the need for data batches by considering each example incrementally. It dynamically weights the predictions of individual classifiers based on their accuracy, with weights initially set to 1 when a new classifier is introduced, and then reduced over time if the classifier makes incorrect predictions. Furthermore, the number of ensemble members in the DWM is flexible. If the entire ensemble makes an incorrect prediction for an example, a new classifier is trained and added to the ensemble, while if a classifier's weight falls below a threshold, it is removed. The DWM was found to perform better than SEA, and is a popular method among the literature.

The Learn^{++}.NSE algorithm developed by Elwell and Polikar [2011] builds upon the DWM algorithm. Learn^{++}.NSE incrementally learns a new classifier on each incoming batch of data and uses DWM-style voting for predictions. However, unlike the basic DWM, the classifiers are evaluated on previous batches as well as the latest one, and the weights then calculated according to a time-adjusted accuracy. This gives Learn^{++}.NSE an advantage over other ensembles as it is able to incorporate historical concept regimes, allowing it to perform well under all types of drift, including recurring concepts which other methods struggle with. On the other hand, the algorithm does not contain any explicit forgetting mechanism for classifiers. Elwell and Polikar [2009] found error-based forgetting superior to time-based forgetting for all types of drift, but also note than both types of pruning are detrimental in cases of recurring concepts.

Brzeziński and Stefanowski [2011] proposed a hybrid method combining a batch and an incremental approach: the Accuracy Updated Ensemble algorithm (AUE). They then followed this with an improved version: AUE2 [Brzezinski and Stefanowski, 2014b]. The AUE2 algorithm was developed based on AWE and processes data in batches, training a new incremental classifier on each batch, which is presumed to be the best classifier at the time. The MSE is then calculated and used to find a weight for each component classifier. The classifier weights are then used to update the ensemble members in the same way as in AWE. After updating the ensemble, the component classifiers are also updated by being incrementally trained on the latest batch of data. In a comparison with other models, AUE2 was found to be among the most accurate. Furthermore, as the base classifiers are able to incrementally update as new data comes in, AUE2 is able to handle all types of concept drift.

In subsequent work on developing batch-based ensemble algorithms towards incremental use, Brzezinski and Stefanowski [2014a] developed the Online Accuracy Updated Ensemble (OAUE), which follows a very similar approach to AUE2, but training and updating the classifiers incrementally as data arrive, as opposed to over batches. However, to preserve similar reactions to drift as would occur in the block based AUE2 ensemble, OAUE learns the ensemble weights over batches of data. The authors find that this approach leads to strong empirical performance compared ot other incremental ensemble approaches, while retaining the ability to effectively handle all types of concept drift.

Most of the ensembles described so far have been in relation to the more commonly studied classification problem. However, Ikonomovska et al. [2015] discuss a collection of ensemble methods for use in regression problems. These approaches are primarily based around two methods; online bagging of Hoeffding-based trees for regression (OBag) and an online RandomForest for any-time regression (ORF). Both of these methods use FIMT-DD as the base classifier and generally outperformed a single tree based on FIMT-DD in a number of tests. Despite this, the ORTO-A algorithm discussed in Section 3.2.3 outperformed both of these methods overall and returned much more stable and accurate predictions.

Ensemble methods involving multiple adaptive mechanisms to mitigate concept drift have also been suggested. Soares and Araújo [2015] classify three types of learning and three types of adaption mechanism which may be included in ensemble approaches to concept drift, before suggesting the On-line Weighted Ensemble (OWE) for use in a regression context. The OWE adapts to change by both updating the weights of ensemble members according to the inverse of their predictive error, and also by modifying the set of active ensemble members contributing to the final prediction. The authors also consider adapting individual ensemble members in response to change, but do not do so, stating that this may lead to redundant models.

Another ensemble designed for regression is the Adaptive Random Forest for Regression (ARF-Reg) algorithm, a modification of the ARF algorithm which was originally designed for classification [Gomes et al., 2017b, 2018]. Both ARF and ARF-Reg attempt to leverage the use of random forests for streaming data, randomly selecting only a subset of features to be considered at each split. This encourages the growth of more diverse ensembles and in turn improved predictive performance, although the use of bagging in the ensemble means it is slow compared to many of the other methods available.

More recently, other approaches applicable for both classification and regression have been devised. Gouk et al. [2019] introduce the stochastic gradient trees (SGT) algorithm which trains an ensemble of decision trees based on gradient information. While a novel approach, the authors find the SGT approach is comparable to other contemporary methods in terms of both predictive performance and time taken, although it generally trains a more lightweight model and thus has lower memory requirements.

3.3 Evaluation

Having seen the format in which concept drifting data streams are presented, and exploring a number of methods for predictive modelling on a data stream, we now progress to explaining how the performance of different predictive models can be evaluated and compared.

3.3.1 Evaluation Metrics

Evaluating a predictive model on a concept drifting data stream requires a multifaceted approach. Based on the criteria provided in Section 3.1, we can see there main aspects of importance to consider:

- The predictive performance of the model.
- The time taken to update and make predictions from the model.
- The memory consumption of the model and any stored data.

Predictive Performance

The most important metric for evaluating the performance of a model is its ability to make correct predictions. Due to differences between regression and categorization scenarios, different methods have been used in each scenario.

For regression, predictive performance is usually measured via the Mean Absolute Error (MAE) [Abramowitz and Stegun, 1974] or the Root Mean Squared Error (RMSE) [Armstrong and Collopy, 1992]. For a single target, they are calculated according to:

$$\text{MAE} = \frac{1}{T}\sum_{t=1}^{T} |y_t - \hat{y}_t| \tag{3.9}$$

and

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T} (y_t - \hat{y}_t)^2}. \tag{3.10}$$

Both of these measures are based on the deviation of the target predictions $\hat{y}_t$ from the true values y_t at each time. The choice of measurement statistic between the two is indicative of the expected distribution of the errors. In general, we expect that the errors in the regression models described in this thesis, particularly in Chapter 6, ought to have normally distributed errors due to our choice of linear models in the leaves, and thus that the RMSE is an appropriate choice of metric. However, for comparison with previous works we choose to use the MAE, which we note is less affected by the presence of large outliers.

On the other hand, for categorization problems, the classification error, or accuracy, is more commonly used as a measure of predictive performance, and is simply given by

$$\begin{aligned}\text{Classification error} &= 1 - \text{Accuracy} &\quad (3.11)\\ &= \frac{\text{Number of incorrect predicitons}}{\text{Total predictions}}. &\quad (3.12)\end{aligned}$$

The performance measures given so far are error based, and models with comparatively lower values for these metrics can be said to have better predictive performance. However, in some scenarios, it is useful to be able to benchmark the performance of an algorithm without having to explicitly compare with an alternative. For regression problems, this can be done using the Relative Mean Absolute Error (RMAE) or the Root Relative Mean Squared Error (RRSE), while for classification the kappa statistic is often used.

The RMAE and RRSE are given by:

$$\text{RMAE} = \frac{\sum_{t=1}^{T} |y_t - \hat{y}_t|}{\sum_{t=1}^{T} |y_t - \bar{y}_T|} \tag{3.13}$$

and

$$\text{RRSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T} \frac{(y_t - \hat{y}_t)^2}{(y_t - \bar{y}_T)^2}} \tag{3.14}$$

respectively, where $\bar{y}_T = \frac{1}{T}\sum_{t=1}^{T} y_t$ is the mean of y_t at time T. These statistics aim to provide an evaluation of the model against a benchmark scenario. The RMAE and RRSE benchmark the model predictions against those of a mean predictor, and lower values indicate better performance. 0 indicates there are no errors in the model, while a value of 1 indicates the performance is equal to that of a mean predictor. Consequently, values in the range $0 <$ RMAE < 1 and $0 <$ RRSE < 1 indicate a model's performance is above that of the benchmark, with values closer to 0 indicating better performing models.

On the other hand, for classification, the kappa statistic is:

$$\kappa = \frac{p_0 - p_{random}}{1 - p_{random}} \tag{3.15}$$

where p_0 is the classification accuracy of the model in question and p_{random} is the classification accuracy of a random classifier. In the case of perfect predictions $\kappa = 1$, while if the classification accuracy is equal to that of the random classifier $\kappa = 0$. This means values in the range $0 < \kappa < 1$ indicate the model performance is above that of the benchmark random classifier, with higher values representing better predictive performance.

Run Times

Although predictive performance is often the most important aspect of predictive modelling, when working with streaming data, the run time can also be an important constraint, especially if data in the stream arrives rapidly. To this end time is considered an important indicator of the performance of a streaming algorithm.

The run time is typically obtained by measuring CPU time taken by the algorithm, with faster run times being desirable. However, it can be tricky to obtain comparable statistics, as run times are dependant on the efficiency of the implementation of an algorithm and the hardware on which it is run. Furthermore, in practice it is possible to parallelise many ensemble algorithms, which also makes it hard to compare the time. Some frameworks, such as the Massive Online Analysis (MOA) project [Bifet et al., 2010a] aim to provide a large scale implementation of many streaming algorithms which can be used to provide a more uniform basis for comparison. However, since MOA is an aggregation of works from many contributors, different methods are still not always comparable, with algorithms from different contributors having different efficiencies. Furthermore, many practical applications will opt for more bespoke implementations which will change the relative run times of different methods.

Memory

The final aspect to consider when evaluating performance is memory. Typically, this refers to RAM consumption. With large amounts of data available in the stream, models can grow extremely large, such as decision trees growing becoming very deep, and windowing algorithms may end up storing a lot of data. This in turn can be expensive, and has traditionally led RAM consumption to being an important consideration when evaluating a model. However, with recent advances in computing, RAM consumption is no longer such an important constraint since the user can always buy more RAM if desired.

On the other hand, as data streams may be theoretically infinite, it is still important to ensure that data streaming algorithms are bounded on the amount of storage required. This is important since if aspects such as the number of parameters involved or the number of examples stored are unbounded, and thus able to grow infinitely, algorithms will eventually run out of memory with an infinite stream.

3.3.2 Evaluation Methodologies

It is important to note that while the predictive performance measures given are used to evaluate the metric, due to the nature of streaming data they can be applied in different ways. Typically, evaluation is performed using either a *holdout* or a *prequential* method.

Holdout Evaluation

Holdout evaluation involves holding out from training the model on a set of examples received in the stream, which are used to form a testing set for the model. Evaluating the model on this testing set enables us to obtain an estimate of the generalization error of the model. However, this approach assumes that the testing set entirely corresponds to the current concept regime on which the model has been trained to obtain an accurate error. Furthermore, it has the drawback of needing to store the testing data in memory.

Prequential Evaluation

Instead of representing the performance of the current model, prequential evaluation [Dawid, 1984] represents the ability of the set of models existing at each time point to make accurate predictions on the stream. The prequential error is calculated using a test-then-train approach, effectively evaluating the performance of the model on each example as it is received, before then using the same example to update the model.

This evaluation methodology is much more robust to concept drift than a holdout evaluation, since the model and the example it is evaluated on are only ever a single example apart in the stream, meaning they likely always have similar concept regimes. In a detailed framework for evaluating stream learning methods provided by Gama et al. [2013], the authors find that both holdout and prequential evaluation converge to the Bayes error, with a prequential evaluation being the optimal approach for data streams.

One point worth noting is the inclusion of forgetting mechanisms when evaluating the performance over datasets. Gama et al. [2013] find that since online models generally improve over time, evaluating the prequential error over all time is pessimistic and leads to an over-representation of the error in the model. Consequently, they suggest the use of forgetting mechanisms to show the current error more accurately, either through the use of sliding windows, or with forgetting factor.

3.4 Summary

This chapter has given an overview of how to account for concept drift when working with streaming data. It has explored the challenges presented by concept drift, before giving an overview of methods for drift detection and predictive modelling in data drifts are suspected. Finally, it has presented some of the evaluation methodologies used when working with concept drifting data.

Methods for predictive modelling with data exhibiting concept drift have been explored in particular detail. These approaches cover traditional methods such as windowing and weighting, as well as more sophisticated online learning algorithms and ensemble approaches. As regression problems have been overlooked for a long time, the majority of these approaches have focused on classification problems, while only more recent aspects of the literature have addressed regression problems. Due to the size of the topic, although most areas of concept drift have been addressed, the chapter is not exhaustive in terms of the methods available. For a deeper exploration of stream learning and concept drift, the interested reader is directed toward the many survey papers in the field [Gama et al., 2014; Krawczyk et al., 2017; Khamassi et al., 2018; Lu et al., 2019; Gomes et al., 2019].

Chapter 4

Stream Forecasting with Extreme Randomization

Non-stationary streaming data poses a familiar challenge to data scientists: the need to obtain fast and accurate predictions. However, the changing characteristics of the data stream exacerbate this challenge. Hoeffding trees are a popular method to confront this problem. Nevertheless, they often struggle to discriminate between similarly performing attributes when making a split decision, which can be detrimental to predictive performance.

In this chapter, we propose the Adaptive Trees for Streaming with Extreme Randomization (ATSER) algorithm which introduces a new concept to streaming data: the idea of Extremely Randomized (Extra) trees [Geurts et al., 2006]. The ATSER algorithm is loosely based on the principle of Extra Trees, although adapted to an online setting, and is able to trivially overcome the problem of similarly discriminatory splits. Furthermore, while ensembles of Extra Trees have empirically been shown to improve predictions by taking advantage of random splitting choices, in the case of streaming data their efficient splitting mechanism offers a second benefit by reducing the computational complexity. In addition, the ATSER algorithm extends its coverage beyond the range of other contemporary methods by directly handling many-category nominal variables. To fully utilize the benefits of Extra Trees for improving predictive performance, an ensemble is necessary. To this end, we also propose three different ensemble algorithms in which ATSER trees act as base learners. The principal contributions of this chapter are:

- The use of extreme randomisation in single decision trees for regression in streaming environments.
- A novel splitting mechanism which leads to reduced training times and improved performance.

- The ATSER algorithm naturally handles nominal attributes allowing it to be applied to a wide range of commonly encountered data.
- An improved approach to learning linear models in the leaves.
- The use of extremely randomized trees in different ensembles designed for concept drifting streaming data.
- Extensive empirical evaluation of ATSER trees and ensembles based on ATSER trees shows that single trees outperform state-of-the-art methods, with ATSER ensembles offering increased prediction accuracy.

The layout of this chapter is as follows. We begin in Section 4.1 by providing a review of adaptive learning trees and ensembles for regression streaming data, before also introducing the principles of Extra Trees in a stationary scenario. We then detail the ATSER algorithm in Section 4.2, following with an introduction of ensemble methods based on it in Section 4.3. After describing the evaluation methodology in Section 4.4, we then compare the performance of the ATSER algorithm with the popular FIMT-DD and recent ORTO and ARF algorithms in Section 4.5. We show empirically on real world datasets that, out of the adaptive base learners considered, a single ATSER tree has the best performance in terms of both run-time and predictive ability. We then show that a single ATSER tree is itself outperformed by ensembles in which ATSER trees act as base learners. Finally, we summarise the chapter in Section 4.6.

4.1 Related Work

The ATSER algorithm trains an incrementally updating regression tree. It is designed for use with streaming data and combines the typical interwoven train and adapt approach of many tree-based streaming algorithms with the use of the largely random splitting decisions introduced in Extra Trees. Compared to other contemporary streaming algorithms, this results in an increase in both computational efficiency and predictive performance. Consequently, in this section we re-review the related literature on online regression which we previously discussed in Chapter 3, and also provide an overview of Extra Trees in a stationary setting.

4.1.1 Online Regression Trees

As large datasets become increasingly available, the biggest challenge is often how to make use of them. In the case of streaming data, many examples can arrive over a short period of time, raising the problem of how to train an effective predictive model within this inherent time constraint, especially considering that such data are rarely stationary and therefore any predictive relations may only persist for a short period. We have seen in the previous chapters that changing relationships in this type of data are known as ‘concept drift’ [Schlimmer et al., 1986]. As concept-drifting data has become more prevalent, a number of approaches to handling it have arisen, with the most prominent methods stemming from the Very Fast Decision Tree (VFDT) algorithm developed by Domingos and Hulten [2000]. As described in Section 3.2.3, the VFDT incrementally trains a decision tree, using the Hoeffding bound to guarantee the performance of splits relative to those that would be obtained in a stationary scenario.

However, for a long period most approaches were focused on classification problems, leaving regression largely unaddressed. The first major attempt to tackle concept drift in a regression scenario is the Fast Incremental Model Trees with Drift Detection (FIMT-DD) algorithm from Ikonomovska et al. [2011b]. Similar to other Hoeffding-based tree methods, the FIMT-DD algorithm has three main aspects: training a tree, testing for change, and adapting the tree. The training is a continuous incremental process where, as examples are observed, they are passed to the leaf nodes, and used to update a linear model and split statistics. When the split statistics are deemed sufficient to determine the best possible split (usually using the Hoeffding bound) a split is created and the linear model from the node passed to the child nodes. Meanwhile, at every split node in the tree, tests for change occur, incrementally updating and providing an alert when change is detected. Upon detection of a change, the training of an alternative tree is triggered which, after a training period, is then swapped for the original tree if it exhibits better performance.

While generally effective, if at least two attributes are similarly discriminative the greedy splitting of FIMT-DD can lead to long delays when selecting a split, because a larger number of examples must be collected to determine the best split. To handle this, Ikonomovska et al. [2011c] suggest the On-line Regression/Model Trees with Options (ORTO) algorithm, which draws on ideas used to extend the original VFDT algorithm for classification through the innovative use of option trees, as originally described by Pfahringer et al. [2007]. ORTO overcomes the slow splitting process in the original FIMT-DD algorithm by creating an option node instead of a binary split node if there is a tie for the best possible split. When examples are passed to an option node, instead of only progressing to one child node, they progress to a relevant child node for every possible split at the node. This effectively creates an ensemble, and the predictions from the tree are able to be combined in numerous ways. Ikonomovska et al. [2011c] find that the best approach is a simple averaging procedure, which they term ORTO-A.

Although this approach does speed up the training process, the authors also note that option trees have a tendency to grow large very quickly. Furthermore, creating an option tree based on similarly performing splits may lead to a common problem with ensembles, in which hindering diversity degrades an ensemble’s ability to generalize, leading to worse predictive performance [Geman et al., 1992].

Other methods for online regression have extended to the idea of applying FIMT-DD trees in bootstrapped ensembles [Ikonomovska et al., 2015; Gomes et al., 2018] and multi-target extensions [Ikonomovska et al., 2011a; Osojnik et al., 2018]. One particularly notable recent approach is the Adaptive Random Forest for Regression (ARF) algorithm [Gomes et al., 2018], which creates a random forest by combining the use of online bootstrapping with a partially randomized choice of splitting feature.

It has also been noted that the performance of online regression tree algorithms may be harmed by their inability to handle nominal attributes. Preliminary attempts to include them have been made by Osojnik et al. [2016, 2017], although these appear to only consider binary categories. While this may be an effective method, converting a many-category variable to a set of binary ones is impractical, handling so many binary variables quickly becomes expensive, and artificially raises the dimensionality of the problem.

Unlike previous attempts, the ATSER algorithm directly handles many-category variables, by generating random nominal splits in an inexpensive online manner. Furthermore, the novel splitting approach, forming splits in a similar way to Extra Trees, is a fast and simple method which allows deeper trees to be generated quickly, and naturally lends itself to generating diverse ensembles.

4.1.2 Extra Trees

Extremely Randomised (Extra) Trees were introduced by Geurts et al. [2006] as a natural extension to random forests [Breiman, 2001] and are primarily designed for use in ensembles. Random forests select attributes on which to split randomly, but then use greedy splitting to determine the best split among the attributes. This is done to obtain various uncorrelated weak learners which ideally give accurate group predictions.

However, Extra Trees go one step further, again drawing random splitting attributes, but also drawing one random split from each attribute before finally using greedy splitting to determine the best of those remaining. This additional randomness leads to even less correlated individual learners, reducing the variance in predictions relative to a random forest. Furthermore, the second randomization step when splitting drastically lowers the number of examples necessary to form a split, thus reducing the time taken to train an Extra Trees ensemble compared to a random forest, a point of particular importance when working with streaming data.

Algorithm 4.1 Overview of the ATSER Algorithm

```
1: $\mathcal{T} := \emptyset$                                        ▷ Initialise empty tree
2: for $t := 1, 2, \ldots$ do                                      ▷ Iterate over data stream
3:     input: $\mathbf{x}_t$
4:     $\hat{y}_t :=$ MakePrediction($\mathcal{T}, \mathbf{x}_t$)
5:     yield: $\hat{y}_t$                                            ▷ Return prediction
6:     input: $y_t$                                                ▷ Observe $y_t$ for training
7:     $\mathcal{N} :=$ GetRoot($\mathcal{T}$)                                      ▷ Start at root node
8:     while IsNotLeaf($\mathcal{N}$) do                                    ▷ Descend to leaf
9:         $\mathcal{N}$, changed := UpdateNodeStatistics($\mathcal{N}, y_t$)
10:                                                   ▷ Boolean indicator changed
11:         if changed = False then
12:             changed, $\mathcal{N} :=$ TestForChange($\mathcal{N}, y_t, \hat{y}_t$)
13:         else                                                ▷ Adapt if change seen
14:             changed, $\mathcal{T}, \mathcal{N} :=$ Adapt($\mathcal{T}, \mathcal{N}, \mathbf{x_t}, y_t, \hat{y}_t$)
15:         end if
16:         $\mathcal{N} :=$ GetChild($\mathcal{N}, \mathbf{x_t}$)                            ▷ Descend one node
17:     end while
18:     $\mathcal{N} :=$ UpdateLeaf($\mathcal{N}, \mathbf{x_t}, y_t, \hat{y}_t$)
19:     if GetSeen($\mathcal{N}$) mod $n_{min} = 0$ then    ▷ GetSeen returns count of examples seen
20:         $\mathcal{T} :=$ AttemptToSplit($\mathcal{T}, \mathcal{N}$)
21:     end if
22: end for
```

4.2 The ATSER Algorithm

The ATSER algorithm is designed for regression problems and incrementally trains a decision tree. The nodes in the tree are trained sequentially and the bound [Hoeffding, 1963] is used to guarantee the performance of splitting decisions relative to those of a non-incrementally trained Extra Tree. We denote by $\mathbf{x}_t$, $t = 1, \ldots$ the stream of features and by y_t the accompanying regression targets; the true value of the target becomes available *after* the prediction $\hat{y}_t$ has been made. As outlined in Algorithm 4.1, a target prediction is made by the tree as each example (or batch of examples) arrives (line 4). At each split node, predictions from any leaves in the subtree are monitored by a change detector (line 12) and, in the event of concept drift, the affected subtree is adapted (line 14). Finally, each example is used to update the tree (line 18), and a new split is attempted if the number of examples seen is sufficient (line 20).

4.2.1 Training the Tree

As examples arrive, statistics are updated in the root node of the tree until there is enough data available to form a split. The required amount is determined using the Hoeffding

Algorithm 4.2 The `UpdateLeaf` function

```
1: input: $\mathcal{N}, \mathbf{x_t}, y_t, \hat{y}_t$        ▷ Input leaf, features, target and prediction
2: $\mathcal{N}$ := UpdateNodeStatistics($\mathcal{N}, y_t$)
3: $\mathcal{S}$ := GetCurrentSplits($\mathcal{N}$)
4: for $x_t$ in $\mathbf{x_t}$ do        ▷ Loop over features
5:     if GetSeen($\mathcal{N}$) $\leq m_{min}$ then
6:         $\mathcal{N}$ := UpdatePossibleSplits($\mathcal{N}, x_t, y_t$)
7:     end if
8:     $\mathcal{N}$ := UpdateSplitStatistics($\mathcal{N}, x_t, y_t$)
9:     if GetSeen($\mathcal{N}$) $= m_{min}$ then
10:        $\mathcal{S} := \mathcal{S} \cup \{$GetRandomSplit($\mathcal{N}$)$\}$
11:    end if
12:    $\mathcal{N}$ := UpdateLinearModel($\mathcal{N}, x_t, y_t, \hat{y}_t$)
13: end for
14: $\mathcal{N}$ := KeepPossibleSplits($\mathcal{N}, \mathcal{S}$)
15: yield: $\mathcal{N}$
```

bound, which can also be used to indicate the best split. After a split is selected, the process is repeated in the child nodes. With the root node being of depth zero, and every subsequent node in the path down the tree being of a depth 1 greater than its predecessor, the splitting process is repeated until the desired depth is reached.

Choosing a Split

Splits are selected via a two-stage process. First, as in Algorithm 4.2, possible splits and their relevant statistics are collected until m_{min} examples have been seen at a node (lines 6, 8). k splits are then randomly selected from the possible splits (line 10), with a maximum of one split for each feature.

After possible splits have been chosen, subsequent examples are used to update their split statistics until a total of n_{min} examples have been seen. As in Algorithm 4.3, the best of the randomly selected possible splits is then chosen with the aim of reducing heterogeneity between the targets in the child nodes, thereby allowing more accurate predictions to be made. The variance of the targets, σ^2, is used as a measure of this, leading to splits being chosen to maximize the reduction in variance between the parent and child nodes (line 6). That is, the split s maximizing the variance reduction

$$\mathrm{VR}(s) = \sigma_P^2 - \frac{N_L}{N_P}\sigma_L^2 - \frac{N_R}{N_P}\sigma_R^2 \tag{4.1}$$

is selected, where $\sigma_{\cdot}^2$ and $N_{\cdot}$ denote the variances and number of targets at the parent (P), left child (L) and right child (R) nodes respectively. In the interest of reducing the required statistics we need to store and obtaining more random splits, we choose $m_{min} = 10$. On the other hand, following previous works, we choose $n_{min} = 200$.

Examples are received incrementally, meaning the best split may change as more examples are received. Consequently, the Hoeffding bound is used to determine when there are enough examples at the parent node to be confident that the current split with the best VR will remain the same as more examples arrive.

As described in Chapter 3, the Hoeffding bound may be used to provide confidence intervals for the mean of a random variable. It states that with probability $1 - \delta$, the empirical mean of N examples of a random variable with range R is within ϵ of the true mean, where:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2N}}. \tag{4.2}$$

Therefore, the Hoeffding bound can be used to choose the best split at a node by recording the ratio of the VR of the second best split relative to the best as each new example arrives at time t:

$$r_t = \frac{\mathrm{VR}_t(\text{second best split})}{\mathrm{VR}_t(\text{best split})}. \tag{4.3}$$

By treating the observed ratios $r_1, r_2, ..., r_T$ as a random variable which ranges from 0 to 1 and providing the desired confidence level, chosen here as $\delta = 10^{-6}$, a split can then be chosen as the best when $1 - \epsilon > r_t$. Details on how to apply the Hoeffding bound can be seen in Algorithm 4.3.

While this approach appear to differ from the traditional classification approach in a number of ways, it is actually equivalent. The ratio r_t is chosen, rather than a difference, in order to guarantee the range of the series is bounded in $[0, 1]$. Furthermore, this ratio is then checked to be smaller than $1 - \epsilon$ since this is equivalent to checking if $\mathrm{VR}_t(\text{best split}) - \mathrm{VR}_t(\text{second best split}) > \epsilon$ after normalising both ϵ and VR_t by the variance of the best split.

Perhaps the most appealing property of the Hoeffding bound is that the $1/\sqrt{N}$ term means that the bound invariably decreases as more examples arrive, meaning it is eventually always possible to find a single best split. Practically, this occurs by either accepting a best split if one is determined after seeing n_{min} examples or, if it is not possible to determine a best split after n_{min} examples are observed, by waiting for an additional n_{min} examples before again checking if a single split is determined to be the best. This process is repeated until, eventually, a single best split is obtained.

However, if splits are similarly performing, it may be impractical to wait for enough examples to choose the best split. ATSER solves this by treating all splits which reach a certain threshold for the Hoeffding bound (here $\epsilon < 0.05$) as equally discriminative. Thus, when the threshold is reached, if no single best split is observed, a split if randomly chosen from among the tied best splits.

Algorithm 4.3 The `AttemptToSplit` function

```
1:  input: $\mathcal{T}, \mathcal{N}$                                  ▷ Input tree and leaf node
2:  VR := $\emptyset$                        ▷ Empty set of variance reductions (VRs) from splits
3:  $\epsilon$ := CalulateHoeffdingBound($\mathcal{N}$)
4:  $\mathcal{S}$ := GetCurrentSplits($\mathcal{N}$)
5:  for $s$ in $\mathcal{S}$ do                                          ▷ Get VR of each split
6:      VR := VR ∪ CalculateVarianceReduction($s, \mathcal{N}$)      ▷ Equation (4.1)
7:  end for
8:  best := max(VR)
9:  second := max(VR \ {best})
10: if $(1 - \epsilon) > \frac{\text{second}}{\text{best}}$ then            ▷ Compare using Hoeffding Bound
11:     $s$ := GetBestSplit($S$)                              ▷ Split with the largest VR
12:     $\mathcal{T}$ := CreateNewSplit($\mathcal{T}, \mathcal{N}, s$)
13: end if
14: yield: $\mathcal{T}$                                             ▷ Return updated tree
```

Numerical Attributes

With large quantities of data it is impractical to store each example in memory. Therefore, only information necessary for generating the VR of each split is kept. As Equation (4.1) shows, for each possible split it is only necessary to store: the number, the sum and sum of the squares of the targets at the parent and potential child nodes. In Algorithms 4.1 and 4.2, the parent statistics are stored in the `UpdateNodeStatistics` function (Algorithm 4.1 line 10, Algorithm 4.2 line 2), while the child statistics are stored in the `UpdateSplitStatistics` function (Algorithm 4.2 line 8). An demonstration of some incoming points and he splitting statistics stored can be seen in Figure 4.1.

While not as expensive as storing all the data, maintaining these statistics is still the most costly aspect of the ATSER algorithm, as every time an example reaches a leaf all of the possible split statistics need to be updated. Similar to [Ikonomovska et al., 2011b], we use an extended binary search tree (eBST) to sort and maintain the possible splits and statistics. The computational efficiency of the ATSER algorithm is also visible here, since as soon as the potential splits have been randomly chosen, there is no need to update the split statistics to consider new potential split points.

Nominal Attributes

As with numerical attributes, y, y^2 and N for both the parent and possible child nodes are necessary to find the VR of a split. However, for a nominal feature x with c possible categories $x_j \in \{x_1, ..., x_c\}$, there are $2^{(c-1)}$ possible splits. As this number scales exponentially with the number of categories, it is too expensive to be considered in a streaming scenario, thus empirically leading to the exclusion of nominal splits in most regression streaming algorithms [Bifet et al., 2010a].

t	X	Y
1	1.38	2.1
2	2.57	3.26
3	1.2	1.56

STATISTIC	1	2	3
x_s	1.2	1.38	2.57
$\sum y_L$	1.56	3.66	6.92
$\sum y_L^2$	2.43	6.86	17.46
N_L	1	2	3
$\sum y_R$	5.36	3.26	0
$\sum y_R^2$	15.03	10.6	0
N_R	2	1	0

Figure 4.1. Left: Three examples for data with a single feature, x and target y. Right: Accumulated split statistics. Statistics are sorted on x and for each pre-existing split x_s, new examples with $x_i \leq x_s$ are denoted L while those with $x_i > x_s$ are denoted R.

Since they cannot be handled directly, nominal variables are often incorporated though the use of dummy variables. However, this can be problematic if the number of categories is large, as a feature with c possible categories is encoded into $c - 1$ different dummy variables, which can drastically increase the number of features and, subsequently, training times. Additionally, splits on dummy variables result in a 1 vs $c - 1$ class division. This necessitates the construction of many nodes to pick out splits between groups of categories, and also results in weakly discriminative splits, since the majority of examples flow down a single branch. Furthermore, practical challenges also exist, since categories may not all be known beforehand, which in turn means new dummy variables must be incorporated into the modelling process as new categories are observed.

On the other hand, the partially randomized split choice in the ATSER algorithm means only 1 split needs to be considered per feature. Furthermore, the nature of nominal variables means that splits on them can be constructed online by assigning newly observed categories to a random branch.

Modelling in the Leaves

A linear model is gradually trained in each leaf of the tree, giving predictions according to:

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{D} \hat{\beta}_j x_j \tag{4.4}$$

where weights w_f are initially set to 0. As examples arrive in the leaves, they are first preprocessed. As in [Ikonomovska et al., 2011b], numeric feature values are standardized according to $x'_t = \frac{x_t - \bar{x}}{3\sigma_t}$, where $\bar{x}$ and σ_t are the mean and standard deviation of examples of x_f in the leaf. Each category of a nominal feature is treated as a one-hot encoded variable. The weights in the linear model are then updated using RMSProp-style stochastic gradient descent [Hinton et al., 2012]. For each individual feature, taking $g_t = (\hat{y}_t - y_t)x_t$, the update rule for each individual weight is given by:

$$\hat{\beta}_{t+1} = \hat{\beta}_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \mu}} g_t, \quad t \neq 0 \tag{4.5}$$

where

$$\mathbb{E}[g^2]_t = \gamma_{\beta} \mathbb{E}[g^2]_{t-1} + (1 - \gamma_{\beta}) g_t^2 \tag{4.6}$$

is the exponentially smoothed mean of g^2. Following preliminary experiments, we choose $\eta = 0.01$, $\gamma_{\beta} = 0.9$, and $\mu = 10^{-8}$.

The use of RMSProp gradient descent enables localized predictions to be made for each feature subspace of the tree, and the cheap regularization allows for a faster initial learning rate to be set, giving better predictions early whilst not overtraining later. Furthermore, as the regularization of the step size is dependant on recent gradients, it allows the linear model to be robust to outliers and to avoid overtraining when adapting in the presence of concept drift. The entire process for training the leaves can be found in Algorithm 4.2.

After a split is made at a leaf, the model used to make predictions is no longer updated at the split node. However, the model is passed down to the child nodes, allowing them to avoid having to train a new model and effectively allowing the model to be refined to suit the specific feature subspace.

4.2.2 Detecting Change

The ATSER algorithm is designed for concept-drifting data. Consequently, it is expected that parts or all of the tree will need updating as the relationship between the targets and features in the data changes. The first step to adapting to this change in relationship lies in detecting the change. Examples are tested for change as they arrive, using the Page-Hinckley (PH) test [Mouss et al., 2004], which works by monitoring the accuracy of the predictions made by the tree, flagging potential change if the accuracy suddenly decreases.

The accuracy of the tree is gauged by monitoring the difference, $e_t = |y_t - \hat{y}_t|$, between the normalized target y_t and its predicted value $\hat{y}_t$. By comparing the difference in each period to the mean difference across every period,

$$\bar{e}_T = \frac{1}{T} \sum_{t=1}^{T} e_t \tag{4.7}$$

the relative accuracy in a period can be obtained. With a minimum detection threshold, α,

Algorithm 4.4 The `TestForChange` Function

```
 1: input: $\mathcal{N}, y_t, \hat{y}_t$                          ▷ Input split node, target and prediction
 2: $\alpha, \lambda :=$ GetParameters($\mathcal{N}$)               ▷ Get user defined parameters
 3: $\bar{e}_{t-1}, m_{t-1}, M_{t-1} :=$ GetPastStatistics($\mathcal{N}$)
 4: $e_t := |y_t - \hat{y}_t|$                                    ▷ Find the prediction error
 5: $\bar{e}_t := \frac{(t-1)\bar{e}_{t-1} + e_t}{t}$              ▷ Update mean prediction error
 6: $m_t := m_{t-1} + e_t - \bar{e}_t - \alpha$                   ▷ Update relative accuracy
 7: $M_t := M_{t-1}$
 8: changed := False
 9: if $m_t < M_t$ : then
10:     $M_t := m_t$
11: else if $(m_t - M_t) > \lambda$ then
12:     changed := True
13: end if
14: $\mathcal{N} :=$ StoreStatistics($\mathcal{N}, \bar{e}_t, m_t, M_t$)
15: yield: changed, $\mathcal{N}$
```

the relative accuracy is given by:

$$m_T = \sum_{t=1}^{T} \left(e_t - \bar{e}_t - \alpha\right). \tag{4.8}$$

The PH test remembers the minimum relative accuracy seen,

$$M_T = \min_t \{m_1, m_2, \ldots, m_T\} \tag{4.9}$$

and compares it to the relative accuracy in each period to generate the PH statistic,

$$PH_t = m_t - M_t. \tag{4.10}$$

If the PH statistic exceeds a threshold, $PH_T > \lambda$, the test determines that there has been a change at time T. The incremental method to apply the test is shown in Algorithm 4.4. Following [Ikonomovska et al., 2011b], we choose $\lambda = 50$, and $\alpha = 0.005$.

As the PH test requires little information to be stored and can be performed quickly, it is very suitable for use with streaming data. Furthermore, the parameters α and λ are independent of the data, meaning that the need for manual adjustment is minimized. However, their optimal values will still vary across datasets, likely depending on the amount of noise and the presence of any drifts in the data. On the other hand, a common failing of the PH test is that gradual changes may be missed because a slow increase in d_t will also increase $\bar{e}_t$, which can keep m_t below the detection level, despite the (slow) change. To offset this, the value of the standard deviation of the targets when the change detector is initialized is kept and used for normalizing subsequent values of d_t. This helps reduce the effect of drift in $\bar{e}_t$ by keeping the standard deviation relative to the data distribution when the test is initialized, rather than the higher value that would occur after gradual drift.

Change detection is performed at each node, giving two possible ways to calculate $\hat{y}_t$. The *top-down* way is for the prediction to be generated at each node as the example passes down the tree, while the *bottom-up* way is to use the prediction made at the leaf for all nodes in the path. While both approaches are viable, Ikonomovska et al. [2011b] argue that the bottom-up approach is superior, having found fewer false positives are generated. We therefore use a bottom-up approach for generating predictions here.

4.2.3 Adapting the Tree

When change is detected in the tree, a natural response would be to simply prune the affected subtree and continue training, allowing the tree to be regrown as if the data were still stationary. However, the detected changes may be a false alarm, and even if they are not, the predictions from the affected subtree may still be more accurate than those of the pruned node.

As shown in Algorithm 4.5, ATSER responds to this by training an alternative subtree, $\mathcal{T}'$, in parallel with the original subtree, $\mathcal{T}$, for a period of time, and switching to the alternative subtree if it has better performance after seeing N_Q examples. If the original subtree has better performance, the training continues and the test is repeated every N_Q examples. If the alternative subtree appears to be getting worse after a total of $10n_{min}$ examples have been seen, it is discarded and the original subtree retained.

As suggested by Gama et al. [2009], each tree is monitored by tracking the mean squared error (MSE) of the predictions it makes as it sees each example. The MSEs are then used to calculate a smoothed loss metric, Q_t^{tree} according to

$$Q_t^{tree} = \gamma_Q MSE_t + (1 - \gamma_Q) Q_{t-1}^{tree}. \tag{4.11}$$

This is calculated for both trees and compared using the the relative log-loss,

$$Q_t = \log(Q_t^{Org} / Q_t^{Alt}) \tag{4.12}$$

where *Org* and *Alt* denote the original and alternative trees respectively; $Q_t > 0$ indicates that the alternative tree is outperforming the original tree. The value for the fade factor, β, as well as the growth time, N_Q, will vary across datasets and should be determined based on the rate at which changes are expected to occur. As the training and retention of an alternative subtree is expensive, at most one alternative subtree is grown at each node of the original subtree. Furthermore, to stop an infinitely recurring process, changes are not considered in the alternative subtrees until they have been adopted. For comparability we

Algorithm 4.5 The `Adapt` function

```
1: input: $\mathcal{T}, \mathcal{N}, \mathbf{x}_t, y_t, \hat{y}_t$        ▷ tree, split node, features, target, prediction
2: $\mathcal{T}'$ := GetAltSubtree($\mathcal{N}$)        ▷ Alternative tree stored at node
3: changed := True
4: $\hat{y}'_t$ := MakePrediction($\mathcal{T}', \mathbf{x}_t$)
5: $\mathcal{T}'$ := TrainAltSubtree($\mathcal{T}', \mathbf{x}_t, y_t$)
6: $Q_t, \mu_Q$ := GetQStatistics($\mathcal{N}$)
7: $Q_t$ := UpdateQStatistic($Q_t, y_t, y'_t, y''_t$)
8: if GetSeen($\mathcal{T}'$) $\bmod N_Q = 0$ then
9:     if $Q_t > 0$ then        ▷ If Alternative tree is better
10:        $\mathcal{T}, \mathcal{N}$ := UseAltSubtree($\mathcal{T}, \mathcal{T}', \mathcal{N}$)
11:        changed := False        ▷ Restart change detection
12:    else
13:        $\mu_Q$ := UpdateMeanQStatistic($Q_t, \mu_Q$)
14:    end if
15: end if
16: if GetSeen($\mathcal{T}'$) $\geq 10 n_{min}$ and $\mu_Q < Q_t$ then
17:     $\mathcal{T}, \mathcal{N}$ := DiscardAltSubtree($\mathcal{T}, \mathcal{T}', \mathcal{N}$)
18:     changed := False
19: end if
20: $\mathcal{N}$ := KeepStatistics($\mathcal{T}', \mathcal{N}, Q_t, \mu_Q$)
21: yield: changed, $\mathcal{T}, \mathcal{N}$
```

follow Ikonomovska et al. [2011b] in choosing the fade factor $\gamma_Q = 0.995$ and growth time $N_Q = 150$, leading to changes being detected on a very short timescale relative to the number of examples in most of our datasets.

4.2.4 Time Complexity

We now discuss the manner in which the mechanism for choosing splits in ATSER leads to lower computational complexity relative to other algorithms such as FIMT-DD. While the complexity of the entire algorithm also depends upon that of the leaf models, these are interchangeable across algorithms and we here focus solely on the tree construction complexity.

On average, the time taken to insert a new value into the eBST structure, which handles the possible splits, is $O(\log(n))$ for an eBST constructed from n examples, while the time taken to attempt to split is $O(n)$. The FIMT-DD algorithm requires a value to be inserted each time a new example is seen, until a split is attempted after seeing n_{min} examples. This leads to an expected time complexity of:

$$O\left(\sum_{k=1}^{n_{min}} \log(k)\right) + O\left(n_{min}\right) = O\left(\log(n_{min}!) + n_{min}\right) \tag{4.13}$$

$$\approx O\left(n_{min}\log(n_{min})\right) \tag{4.14}$$

using Stirling's approximation. On the other hand the ATSER algorithm chooses which splits to consider after $m_{min} < n_{min}$ examples have been seen, before also attempting to split after seeing n_{min} total examples. Similar to the FIMT-DD algorithm, this leads to a time complexity of $O(m_{min} \log(m_{min}))$ over the first m_{min} examples. After a split has been chosen, the eBST is reduced to a single node, requiring O(1) for each of the remaining $a = n_{min} - m_{min}$ examples, thus having a complexity of

$$O(m_{min} \log(m_{min}) + a). \tag{4.15}$$

We then find the difference in times to be:

$$\begin{aligned} &O(n_{min} \log(n_{min}) - m_{min} \log(m_{min}) - a) && (4.16)\\ &\qquad = O((a + m_{min}) \log(n_{min}) - m_{min} \log(m_{min}) - a) && (4.17)\\ &\qquad = O\left(a(\log(n_{min}) - 1) + m_{min}\left(\log\left(1 + \frac{a}{m_{min}}\right)\right)\right). && (4.18)\end{aligned}$$

As all of $a, n_{min}, m_{min} > 1$, we can see that providing $\log(n_{min}) > 1$, this difference must be positive, suggesting that the ATSER algorithm should generally be faster than the FIMT-DD algorithm.

4.3 Ensembles

Ensembles are often able to increase predictive performance compared to that of a base learner [Dietterich, 2000]. They are especially effective with Extra Trees, which are designed to reduce variance by leveraging the combination of both randomized split points for the individual trees and the ensemble averaging procedure [Geurts et al., 2006].

Here we describe three ensemble approaches which use ATSER trees as base learners.

The **ATSER-Seeds** ensemble is constructed simply by using a different random seed for each base learner, allowing us to offset the stochastic aspect of individual ATSER trees. We randomly choose the seeds, under the constraint that no two members can have the same seed.

ATSER-OBag is a way of replicating the results that would be obtained by training models on data obtained by performing bootstrap sampling on a training dataset where all examples are always available. Online bagging was introduced by Oza and Russell [2001], who showed that as the size of the dataset tends to infinity, the number of times each individual example appears in the bootstrap sample tends to a Poisson(1). This enables us to replicate offline bagging with M replications by training on each member of the datastream m times where $M \approx \frac{e^{-1}}{m!}$. Furthermore, Bifet et al. [2010b] have shown that drawing from a

Poisson(θ), with $\theta > 1$, modifies the ensemble input space and allows a larger portion of the data to be utilised. In light of this, we proceed using a Poisson(3) in the ATSER-OBag ensemble.

In the ASTSER-Seeds and ATSER-OBag algorithms we choose the number of randomly generated splits, k, for each ATSER tree such that one split is considered for each available feature, leading to $k = F$ which Geurts et al. [2006] find empirically to be most effective. By choosing $k = \sqrt{F}$, we create the **ATSER-RForest** ensemble. This choice of k gives a specific case of the original Extra Trees algorithm which acts similarly to the random forest algorithm suggested by Breiman [2001], albeit without the bagging approach used in random forests. As such, our third ensemble approach is to train each base learner in the ensemble on all F features, but to randomly select a subset of $\sqrt{F}$ features for each splitting decision.

In the experiments described below each of the ensembles has 10 members and the ensemble prediction is found as the arithmetic mean of the individual predictions. We use a larger learning rate $\eta = 0.025$ for the linear models of each member, taking advantage of the ensemble averaging to counter the effects of any overtraining ($\eta = 0.01$ for single ATSER trees).

4.4 Evaluation

Having described the model used in the ATSER algorithm, we now detail the datasets and methodology we have used for evaluating the ATSER algorithm.

4.4.1 Datasets

As summarised in Table 4.1, we use 11 stationary and 4 non-stationary timeseries datasets for evaluation.

Stationary Datasets

Following Ikonomovska et al. [2011b], we use 10 well known regression datasets available from: the UCI Machine Learning Repository, the Delve Repository and the StatLib System's site. They are presumed stationary and can be used to benchmark the performance of online algorithms in stationary scenarios.

We also use the larger **YEAR MSD DATASET**, allowing for a comparison over a longer time horizon. Here the task is to use audio features to predict the release year of a selection of songs from 1922-2011.

Table 4.1. Overview of the datasets used. N gives the number of records, μ_y and σ_y are the mean and standard deviation of the targets, NUM and CAT list the number of numerical and categorical variables, while N_{CAT} gives the total number of categories in the dataset.

DATASET	SYNTHETIC	N	μ_y	σ_y	NUM	CAT	N_{CAT}
ABALONE	NO	4.98E3	10.07	3.325	8	1	3
AILERONS	NO	1.38E4	-8.7E-4	4.1E-4	41	0	0
CAL_HOUSING	NO	2.05E4	2.07E5	1.15E5	9	0	0
ELEVATORS	NO	1.66E4	0.0216	0.0067	19	0	0
HOUSE_8L	NO	2.28E4	5.01E4	5.28E4	9	0	0
HOUSE_16H	NO	2.28E4	5.01E4	5.28E4	17	0	0
MV_DELVE	NO	4.10E4	-8.85	10.417	8	3	7
POL	NO	1.56E4	29.07	41.795	49	0	0
WIND	NO	6.57E3	15.60	6.698	13	2	43
WINEQUALITY	NO	5.30E3	5.87	0.89	12	0	0
YEAR MSD	NO	5.15E5	2.00E3	10.93	90	0	0
AIRLINE 08	NO	5.81E6	7.95	37.91	5	7	670
FRIEDMAN	YES	1.00E6	14.41	4.98	10	0	0
HYPERPLANE	YES	1.00E6	0.815	1.142	5	0	0
HOUSING	NO	2.25E7	1.70E5	1.5E5	0	8	1785
ZURICH	NO	5.47E6	53.34	201.30	7	7	1662

Timeseries Datasets

The synthetic **FRIEDMAN DATASET** was introduced by Friedman [1991]. We generate 1 million examples, each consisting of 10 uniformly and independently distributed features in the range $(0, 1)$, which are used to predict a single target, y. The target is created from a nonlinear combination of the features and a noise term $\nu \sim N(0, 1)$:

$$y^{(1)} = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \nu$$

Following Ikonomovska et al. [2011b], three abrupt concept drifts were introduced by permuting the features in the generative equation, resulting in y in the second and last quarters of the dataset being given by:

$$y^{(2)} = 10 \sin(\pi x_4 x_5) + 20(x_2 - 0.5)^2 + 10x_1 + 5x_3 + \nu$$

One notable aspect of using a synthetic dataset is that it is possible to determine the Bayes error. In the case of the mean absolute error, this is given by $\mathbb{E}[|\nu|]$, which gives a Bayes error of 0.798 for the Friedman dataset.

The synthetic **HYPERPLANE DATASET**, suggested by Gomes et al. [2018], is based on the classification dataset from Hulten et al. [2001]. The task is to predict the squared distance of points from a $D - 1$ dimensional hyperplane. We again generate 1 million examples, with 5 uniformly and independently distributed features in the range $(0, 1)$, all of which are used to determine the target y. After generating the features, we also generate weights for the hyperplane. The hyperplane passes through the centre of the unit hypercube in which the features are located, $\mathbf{c} = (0.5, 0.5, 0.5, 0.5, 0.5)$, and begins with a set of uniform and positive weights $\mathbf{w}^{(1)} = (1, 1, 1, 1, 1)$. These are then used to create the target according to:

$$y = 10 \left(\sum_{j=1}^{D} \frac{(x_j - c_j)w_j}{|\mathbf{w}|} \right)^2 .$$

We then introduce three incremental changes by rotating the hyperplane, beginning the changes after 25%, 50% and 75% of the data have been seen respectively. Over the course of 100000 examples, the weights are gradually linearly updated at each change to reach $\mathbf{w}^{(2)} = (-1, 1, -1, 1, -1)$, $\mathbf{w}^{(3)} = (-1, -1, -1, -1-, 1)$ and $\mathbf{w}^{(4)} = (1, -1, 1, -1, 1)$ respectively. We note that due to the square in y, the weights $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(3)}$ are actually equivalent, as are the weights for $\mathbf{w}^{(2)}$ and $\mathbf{w}^{(4)}$. However, as the drift is incremental, the direction of the rotation means that the squared distances during the change will be different.

Finally, we add noise to the dataset, after the changes have been introduced. The noise term, $\nu \sim N(0, \frac{\sigma_y^2}{16})$, gives noise equal to a quarter of the standard deviation of the target. As we know the form of the noise, we can again find the Bayes error, which is 0.221.

The task in the **AIRLINE 08 DATASET** is to predict delays to US domestic flights. The dataset was originally provided for the 2009 Data Expo and the task is to predict flight delays. While the original dataset used by Ikonomovska et al. [2011b] records information about US domestic flights between 1987-2008, we use a shorter version, consisting only of flights in 2008.

The **HOUSING DATASET** originates from HM Land Registry. It contains time ordered records of individual housing transactions from 1995-2017. The goal is to predict transaction values based on multiple nominal attributes, with the most diverse attribute containing 1170 distinct categories.

The **ZURICH DATASET** contains information for predicting public transport delays for 4 weeks in November 2016. The OpenML version was sorted chronologically before use.

4.4.2 Methodology

Due to the evolving nature of streaming data, evaluation methods designed for stationary data will be inaccurate and do not properly capture algorithm performance. Instead, we have discussed two main approaches for evaluating performance on streaming data in Section 3.3.2: prequential and holdout.

We note that, in general, holdout evaluation gives a fairer assessment of the model itself,

Table 4.2. An overview of ATSER hyperparameters

Description	Symbol	Value
Num examples after which splits are randomly selected	m_{min}	10
Num examples after which the best split is chosen	n_{min}	200
Desired significance for a split	δ	10^{-6}
Split tie threshold for the Hoeffding bound	ϵ	0.05
Leaf learning rate	η	0.01
Leaf forgetting factor	γ_β	0.9
Page-Hinckley test threshold	λ	50
Page-Hinckley minimum detection threshold	α	0.005
Alt tree MSE fade factor	γ_Q	0.995
Alt tree comparison frequency	N_Q	150

assuming that the data collected for each holdout set are stationary. On the other hand, although prequential evaluation is negatively biased due to the inclusion of mistakes made early on in the training period, it provides a better assessment of how well predictions can be made at each time. As the main goal of our predictive modelling task is to provide accurate predictions rather than an informative model, we have elected to use a prequential evaluation method, both in this chapter and for the rest of this thesis.

We evaluate the performance itself using the mean absolute error (MAE). However, the size of the data makes it challenging to represent so many predictions coherently. We therefore display the performance evaluated over a sliding window of prequential predictions, as discussed in Section 3.3.2. We use windows of 10^4 predictions for the slightly smaller YearMSD and Friedman datasets, and 10^5 for the other timeseries datasets.

Furthermore, throughout this thesis we present results averaged over 10 runs, to ensure the results are robust to the random elements in both ATSER and the subsequent algorithms presented in Chapters 5 and 6 . Although this is comparatively fewer than would be chosen in many circumstances, we use 10 runs since our datasets are large, meaning each additional run is costly in terms of time taken. However, as we will see in Section 4.5 , the standard error of the results is small, providing a level of confidence in their validity.

The ATSER algorithm and ensembles were implemented in the MOA framework [Bifet et al., 2010a], ensuring comparability with the already present MOA implementations of the reference algorithms. We use the parameters specified across the previous sections for ATSER-based algorithms, and all but one of the default parameters suggested by the authors for comparison algorithms. The single parameter change is that we have increased the learning rate in linear models of the ensemble algorithms used for comparison to 0.025, to match that of the ATSER ensembles. In all cases this led to an improvement in the comparison algorithm relative to the default suggested parameters. A summary of hyperparameters and their chosen values can be seen in Table 4.2.

Another point worth mentioning is that different choices of hyperparameters can lead to threats to experimental validity if ill-suiting values are chosen for some algorithms, while well-suited values are chosen for others. To ensure the results are comparable, we have chosen defaults for ATSER and subsequent algorithms in this thesis to match the parameters of the comparison algorithms where common hyperparameters exist, such as the choice of $n_{min} = 200$. Furthermore, we have evaluated over a number of datasets for which the chosen hyperparameters have been shown to be optimal for the comparison algorithms [Ikonomovska et al., 2011b] , ensuring that the hyperparameters are not biased toward the proposed algorithms. We also consider the performance of the algorithms using the same fixed hyperparameters over all of the datasets, making it unlikely that having better-suited hyperparameters will be the source of an algorithm outperforming the others, since the same hyperparameters are unlikely to be well suited to all of the different datasets.

No parallelization was used in our implementation, and all results were obtained on a laptop with an Intel Core i5-6200U dual core 2.3GHz CPU, L2 cache: 513kb, L3 cache: 3Mb, 16Gb RAM @ 2133MHz.

4.5 Results

The ATSER algorithm and ensemble methods we have proposed need to be relevant in a number of scenarios. In this section we first demonstrate their effectiveness on a number of smaller, stationary datasets, finding that they outperform prior methods in most cases. We then show the performance of the ATSER algorithm in a test scenario, where controlled changes are induced in otherwise stationary data, before proceeding to test the algorithm on multiple real-world datasets which are suspected to be non-stationary.

At all stages we compare with the FIMT-DD algorithm as the de facto benchmark for regression methods, and also with the contemporary ORTO-A and ARF algorithms. ORTO-A is found by Ikonomovska et al. [2015] to have strong performance relative to many other ensemble methods, while Gomes et al. [2018] find ARF to perform well on a selection of real world datasets. All results shown are averaged over 10 runs. Summary results for the larger datasets can be found in Table 4.3.

Table 4.3. Mean Absolute Error of each model on the larger datasets. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the best performance.

	Single Learners			Ensembles			
	FIMT-DD	ORTO-A	ATSER	ATSER-Seeds	ATSER-RForest	ATSER-OBag	ARF-Reg
Airline 08	19.47 (0.10)	20.21 (0.00)	16.89 (0.03)	**14.97 (0.02)**	15.07 (0.02)	16.39 (0.01)	21.00 (0.01)
Friedman	1.79 (0.06)	2.14 (0.00)	1.64 (0.02)	1.36 (0.01)	1.39 (0.01)	1.40 (0.01)	**1.18 (0.01)**
Housing	97724 (0)	92121 (0)	55968 (133)	**55857 (25)**	55933 (21)	59268 (52)	101258 (48)
Hyperplane	0.418 (0.0003)	0.722 (0.0005)	0.39 (0.0002)	0.28 (0.0001)	0.289 (0.0002)	0.293 (0.0002)	**0.263 (0.0001)**
Year MSD	10.10 (0.75)	17.12 (0.00)	9.12 (0.03)	**7.52 (0.01)**	7.71 (0.01)	8.44 (0.01)	13.43 (0.01)[1]
Zurich	64.89 (0.14)[2]	63.49 (0.00)	50.10 (0.21)	39.90 (0.11)	45.20 (0.10)	**38.78 (0.05)**	71.92 (0.03)[3]
Average Rank	5.67	6.17	4.17	1.5	2.5	3.17	4.83

[1] We have removed ≈70k outlying predictions in each run for comparability. Retaining them leads to an MAE of 9.72×10^4

[2] We have removed ≈10k outlying predictions in each run for comparability. Retaining them leads to an MAE of 2.75×10^3

[3] We have discarded 2 outlying runs and removed ≈3k outlying predictions in each remaining run for comparability. Retaining them leads to an MAE of 1.652×10^5

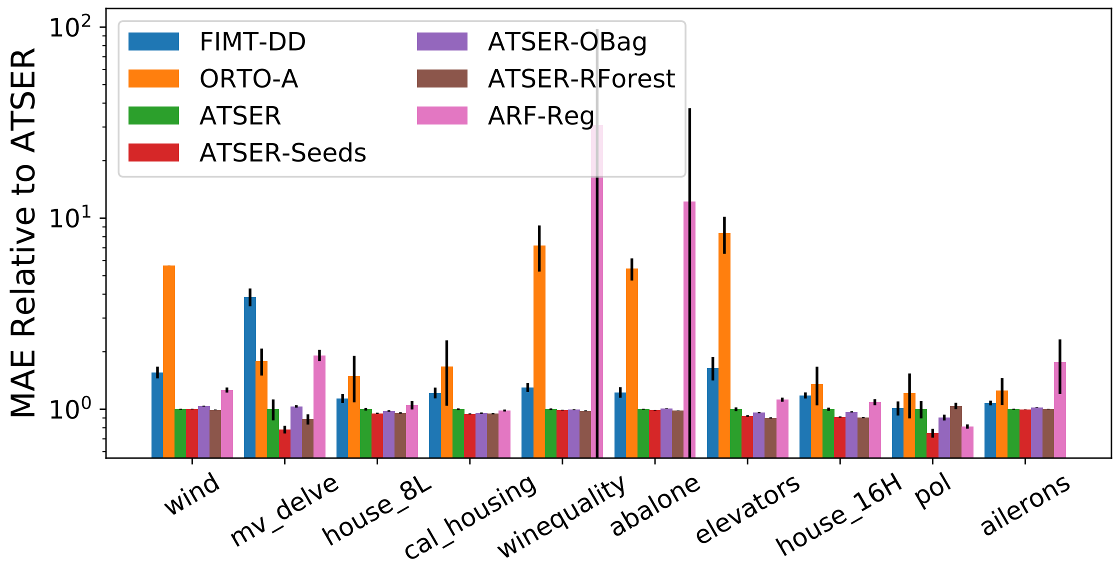


Figure 4.2. FIMT-DD, ORTO-A, ARF and ATSER ensemble performance, relative to ATSER on smaller stationary datasets. The mean absolute error and standard error (over 10 runs) relative to ATSER for each algorithm is shown. We have removed ≈100 and ≈500 outliers from each run of the ARF algorithm on the elevators and ailerons datasets respectively. Retaining them leads to MAEs of 2.92×10^5 and 9.55×10^6 respectively, relative to ATSER.

4.5.1 Performance on Stationary Data

Results from testing on the smaller stationary datasets are shown in Figure 4.2, which shows the ratio of each method's MAE relative to the ATSER result. Note the log scale.

The ATSER-based algorithms all perform better than the FIMT-DD and ORTO-A (ORTO with averaging) approaches on every dataset, although they are still within error on the POL and AILERONS datasets. Meanwhile, the ARF algorithm performs erratically, and while it generally performs worse than the ATSER-based algorithms, it outperforms two of them on the POL dataset. Notably, the ATSER-Seeds ensemble is particularly effective, either outperforming all of the other methods or performing within error of the best algorithm on every dataset. One surprise is that a single ATSER tree still performs well in comparison to other algorithms, since intuitively we would expect it to perform the worst due to the randomness hurting the performance of a single tree. This is likely due to the faster training speed of the ATSER algorithm, which is more noticeable on smaller datasets. The faster training arises from both rapid split construction in the ATSER tree and also from RMSProp SGD allowing larger steps when training the linear models in the leaves.

The performance of each algorithm on the YEAR MSD dataset can be seen in Figure 4.3. The ATSER algorithm performs better than the other single tree algorithms except for a relatively brief period when the FIMT-DD is comparable with it. Indeed the ATSER single tree algorithm outperforms the ensemble-based ARF method. Overall, the ATSER-Seeds approach again performs best, although all the ATSER ensemble methods are perhaps

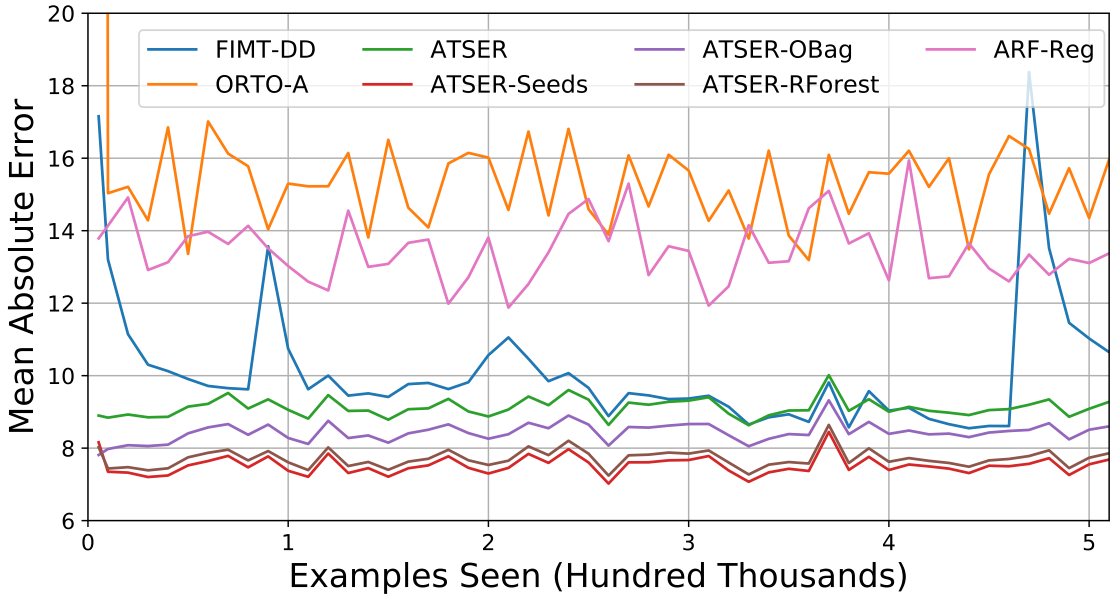


Figure 4.3. Performance on YEARMSD over 10k windows. We have removed ≈70k outlying predictions in each run of ARF-Reg for comparability. Retaining them leads to an MAE of 9.72×10^4

unsurprisingly more effective than the single tree methods. It is interesting to note that all of the ATSER-based algorithms perform more consistently than the FIMT-DD, ORTO-A and ARF algorithms in this scenario.

4.5.2 Effectiveness in Non-stationary Scenarios

While the stationary comparisons show that the ATSER algorithms are effective on datasets of varying sizes, the true purpose of the algorithms is for use in potentially non-stationary timeseries scenarios. We use the FRIEDMAN data to test the performance in a mostly stationary scenario with abrupt, controlled changes, and the other timeseries datasets to test how well the algorithms perform in practice.

Synthetic Data

The FRIEDMAN dataset shows how the algorithms respond to abrupt changes; learning curves can be seen in Figure 4.4. While the algorithms train rapidly at first, none of them appears to have fully converged before the change points.

Unlike the previous datasets, the ATSER-Seeds ensemble only has the second best performance, with the ARF ensemble having the best. Furthermore, contrary to the previous datasets, FIMT-DD outperforms the ATSER algorithm. However, after each of the changes, the peak in the ATSER curve remains shorter than that of the FIMT-DD algorithm, with improved performance over the next few hundred thousand examples. This suggests that

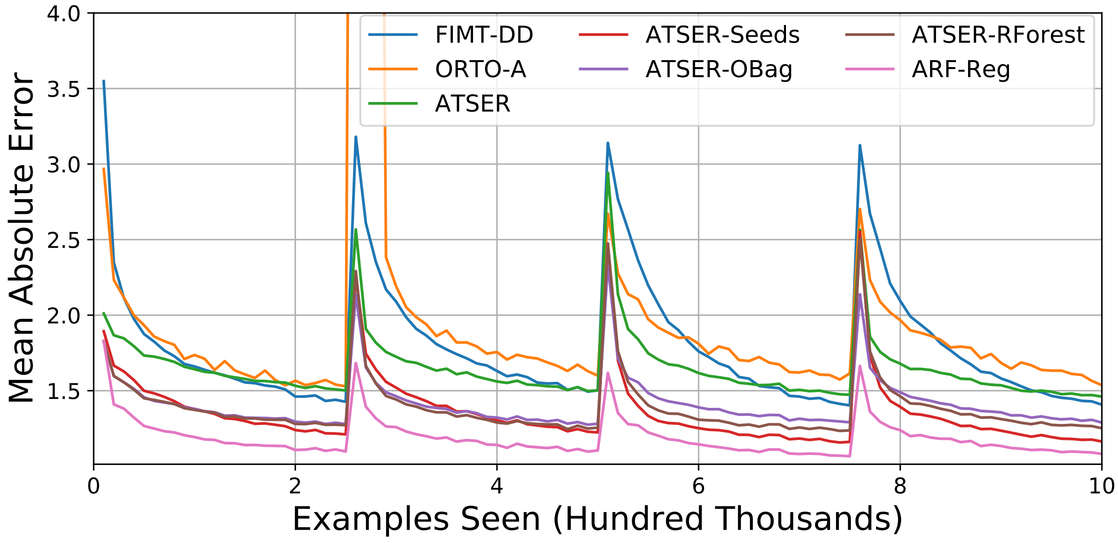


Figure 4.4. Performance on FRIEDMAN over 10k windows. A section of the ORTO-A results which peaks at 14.5 has been omitted for visibility.

ATSER responds faster to change than FIMT-DD, although it may do worse over a long stationary period. We recall, however, ASTER's better performance on more realistic stationary datasets in Section 4.5.1.

On the other hand, the HYPERPLANE dataset shows how the algorithms respond to incremental drifts, which can be seen in Figure 4.5. Overall, the ARF-Reg ensemble has the best performance. Interestingly, however, both FIMT-DD and ARF-Reg appear to react somewhat differently in the face of changes to the ATSER based approaches. As the first change occurs, compared to the ATSER approaches, both FIMT-DD and ARF-Reg face much steeper increases in MAE, before rapidly adapting to the changes. On the other hand, the ATSER methods all adapt more slowly, but have much smoother learning curves. Furthermore, while the ARF-Reg ensemble performs well in the stationary sections, the ATSER-Seeds ensemble appears to perform well in the presence of change. This suggests that the optimal choice of algorithm may depend on the frequency of the drifts in the dataset.

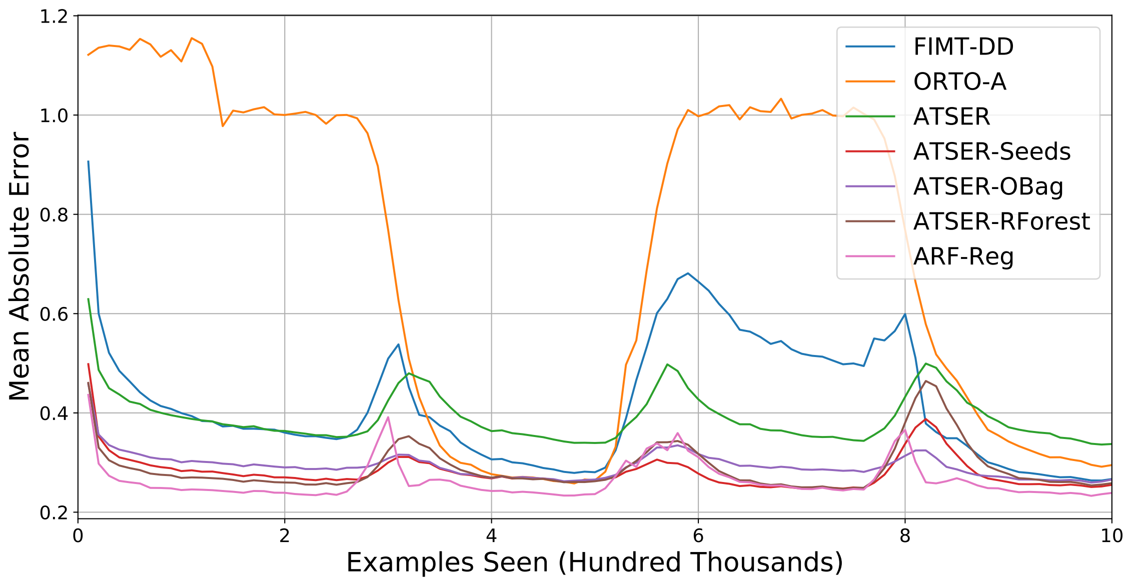


Figure 4.5. Performance on HYPERPLANE over 10k windows.

Real World Data

Figures 4.6 and 4.7 show comparisons of the algorithms on the AIRLINE08 and ZURICH datasets respectively. The jagged learning curves indicate the presence of concept drift as the ability of each algorithm to predict changes over time. This is particularly noticeable in the ZURICH dataset, where there are 4 cycles in performance corresponding to changes across the 4 weeks in the dataset.

Across these datasets we see that ATSER outperforms the FIMT-DD, ORTO-A and ARF methods at all times. Although the highly randomized splits in ATSER should lead to a single ATSER tree performing worse than a single FIMT-DD tree in terms of splits, the inclusion of nominal features and the use of RMSProp gradient descent leads to improved performance. This occurs since including nominal features provides more training data, while RMSProp gradient descent allows the linear models in the leaves to rapidly adapt to concept drift.

On the other hand, all of the ATSER-based ensemble algorithms outperform the other approaches. While the ATSER-OBag ensemble performs best on the ZURICH dataset, we see that the ATSER-Seeds ensemble, which together with ASTER-RForest, performs best over the AIRLINE08 datset, has the most consistent performance.

In the HOUSING dataset (Figure 4.8) only nominal features are available meaning that, as they cannot split, ORTO-A acts as a mean predictor, while FIMT-DD learns the constant in a linear model and ARF learns the same constant via an ensemble. Consequently, the ATSER based algorithms again perform better here.

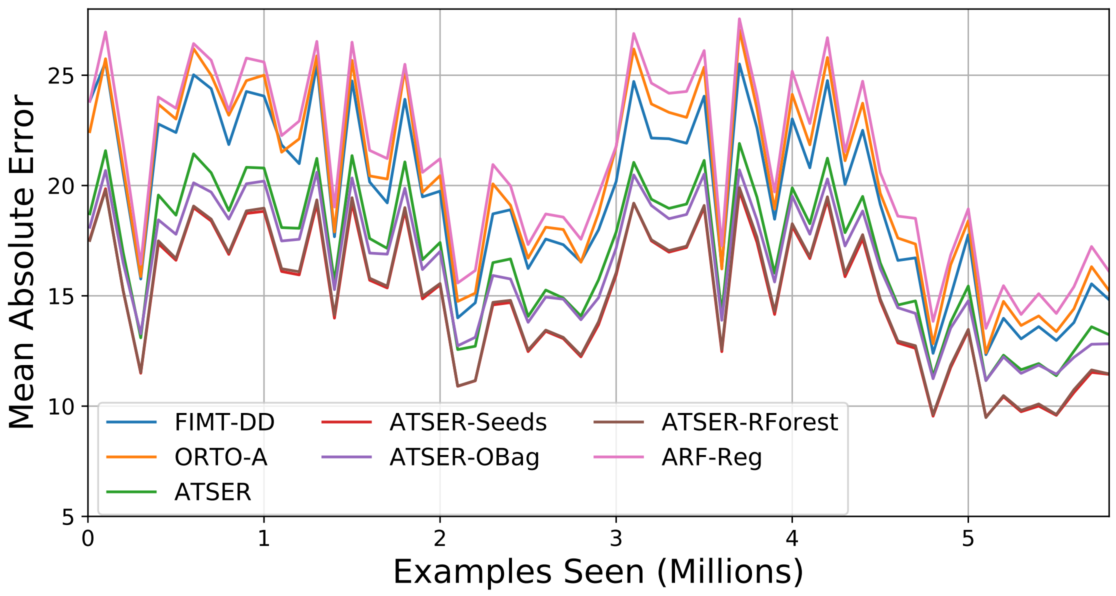


Figure 4.6. Performance on AIRLINE 08 over 100k windows. Note that the performance of ATSER-Seeds (red) is almost obscured by that of ASTER-RForest (brown), which performs very similarly.

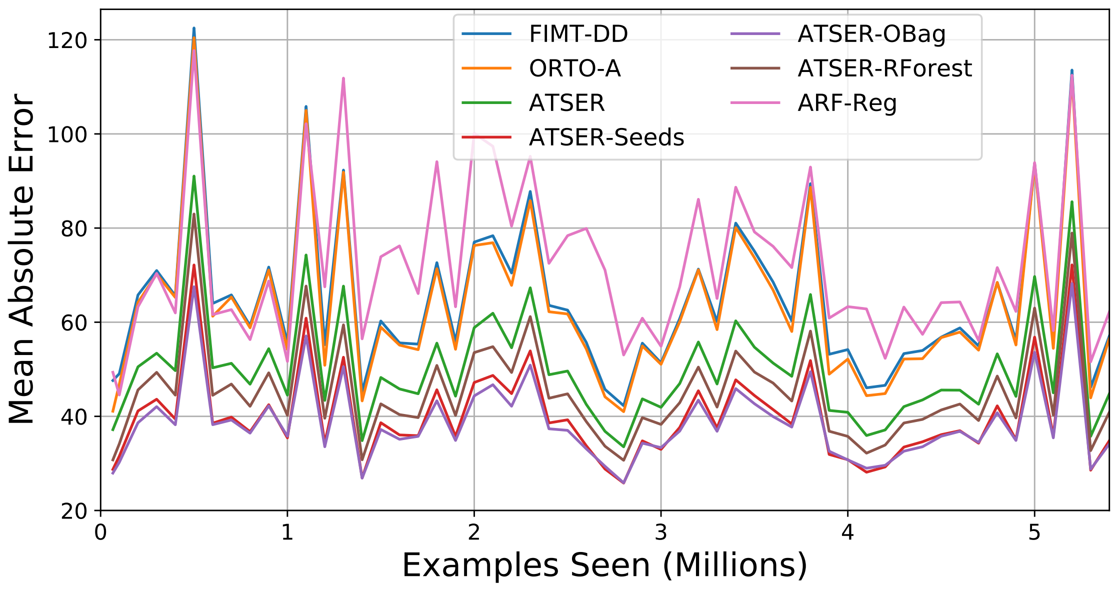


Figure 4.7. Performance on ZURICH over 100k windows. We have removed ≈10k outlying predictions in each run of FIMT-DD for comparability. Retaining them leads to an MAE of 2.75×10^3. We have discarded 2 outlying runs and removed ≈3k outlying predictions in each remaining run of ARF-Reg for comparability. Retaining them leads to an MAE of 1.652×10^5

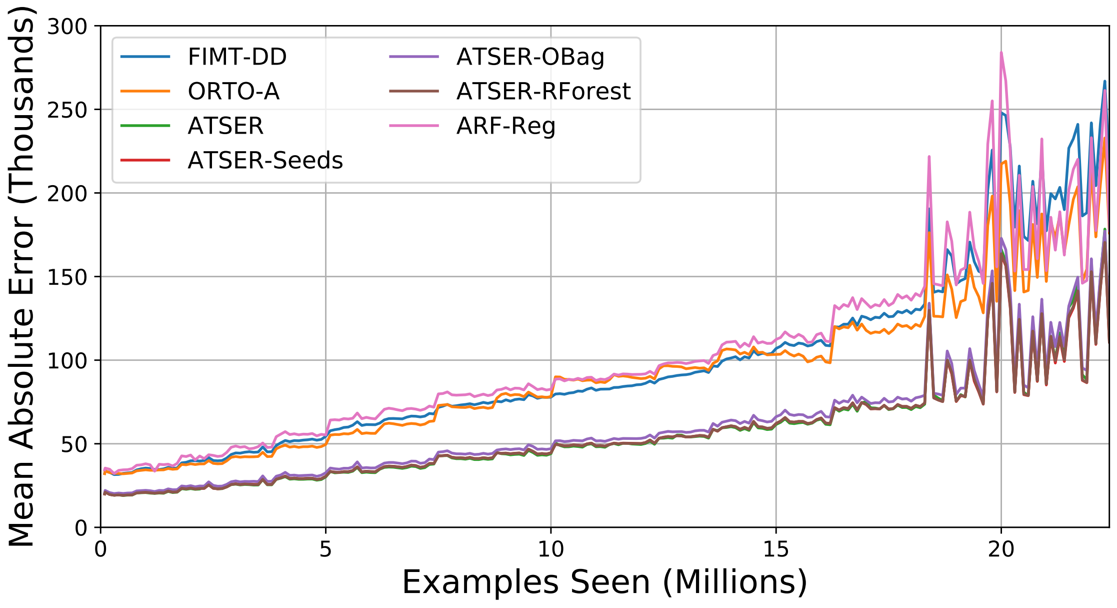


Figure 4.8. Performance on HOUSING over 100k windows. Note that the performance of ATSER (green) and ATSER-Seeds (red) are almost obscured by that of ASTER-RForest (brown), which performs very similarly.

4.5.3 Run time

The average run time for each model and dataset over 10 runs can be found in Table 4.4. On all datasets, the fastest performing algorithm is either ATSER or ORTO-A, and ATSER is faster than or within error of FIMT-DD on all but the HOUSING dataset.

The inconsistency in speed is due to nominal features in the datasets, as ATSER uses them while FIMT-DD and ORTO-A cannot. The fairest time comparisons are therefore on the FRIEDMAN and YEARMSD data, which contain no nominal features. On these, ATSER runs significantly faster, as expected based on the discussion in Section 4.2.4.

On the other hand, the ensembles are slower. However, we expect that parallelizing the ATSER-Seeds and ATSER-RForest ensembles would, in the limiting case, lead them to achieving similar times to the ATSER algorithm. As the ATSER-OBag ensemble trains an average of 3 times on each example, and the ARF ensemble 6 times, they are likely to remain slower even after parallelization.

4.5.4 Effect of Linear Models

Learning in the tree-based algorithms tested occurs in two places: the tree itself, and the linear models. However, this can make it challenging to disentangle how different aspects of the algorithms are performing, especially since we have introduced a different approach for both the splitting procedure in the trees and the method of training the linear models.

Consequently, in this section we repeat part of the previous analysis, but without linear models in the leaves of the trees, allowing for a better comparison of the novel splitting mechanism in ATSER trees relative to previous approaches. The results of this approach for the large datasets are summarised in Table 4.5 and detailed performance graphs are given in Appendix A.1.

Somewhat surprisingly, many of these results do not differ greatly from those with linear models in the leaves (Table 4.3), suggesting the difference in performance on most datasets is largely due to the novel splitting process. However, the MAE of all algorithms on the synthetic FRIEDMAN dataset is noticeably worse without the linear models in the leaves, suggesting that the dataset behaves differently to many real-world datasets, possibly due to the low noise it exhibits.

Another notable difference is that while the ATSER-Seeds ensemble algorithm retains the best overall performance, removing the linear models actually improves performance of the ATSER ensemble algorithms on the YEARMSD dataset. This suggests the inclusion of linear models in the leaves may not always be beneficial, although it is not clear as to why this is the case.

Furthermore, by focusing in particular on the datasets without nominal features, we are able to ascertain the impact of ATSER's splitting mechanism. Looking at the results on the FRIEDMAN, HYPERPLANE and YEARMSD datasets, we see that for all three datasets a single ATSER tree is clearly the fastest algorithm, confirming that the splitting mechanism leads to faster run times. However, in terms of performance, it is less clear. While the ATSER-Seeds ensemble is the best performing algorithm on the FRIEDMAN and YEARMSD datasets, the ARF-Reg ensemble performs the best on the HYPERPLANE data, although by a comparatively smaller amount that in the case with linear models. Furthermore, in comparison to the results with linear models, the drop in performance of the ATSER algorithms is lower in all cases than that of the other algorithms. Altogether, this suggests that much of the improved performance of the ATSER algorithm can be directly attributed to the introduction of randomness in the splitting mechanism.

Table 4.4. CPU time taken (below) for each model on the larger datasets. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the fastest time.

	Single Learners			Ensembles			
	FIMT-DD	ORTO-A	ATSER	ATSER-Seeds	ATSER-RForest	ATSER-OBag	ARF-Reg
Airline 08	124 (1)	**94 (1)**	**94 (2)**	386 (3)	410 (3)	624 (4)	562 (1)
Friedman	36 (1)	32 (1)	**22 (0)**	71 (2)	72 (1)	129 (2)	234 (23)
Housing	314 (2)	**278 (2)**	396 (4)	1696 (9)	1704 (10)	2472 (9)	4138 (19)
Hyperplane	23 (0)	28 (0)	**16 (0)**	51 (0)	56 (1)	89 (1)	137 (5)
Year MSD	95 (1)	78 (0)	**50 (0)**	157 (3)	158 (2)	321 (6)	664 (13)
Zurich	92 (1)	**75 (1)**	93 (1)	352 (3)	408 (4)	682 (9)	718 (26)
Average Rank	2.5	1.67	1.67	4	5	6.17	6.83

Table 4.5. Mean Absolute Error of each model on the larger datasets without linear models in leaves. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the best performance.

	Single Learners			Ensembles			
	FIMT-DD	ORTO-A	ATSER	ATSER-Seeds	ATSER-RForest	ATSER-OBag	ARF-Reg
Airline 08	19.48 (0.01)	20.21 (0.01)	19.88 (0.01)	**15.1 (0.01)**	15.21 (0.01)	16.3 (0.01)	20.73 (0.01)
Friedman	1.81 (0.00)	2.14 (0.00)	1.99 (0.00)	**1.36 (0.00)**	1.39 (0.00)	1.39 (0.00)	1.63 (0.00)
Housing	92122 (50)	92122 (50)	92122 (50)	**55556 (46)**	55603 (46)	58763 (47)	97384 (48)
Hyperplane	0.504 (0.0003)	0.722 (0.0005)	0.445 (0.0003)	0.348 (0.0002)	0.38 (0.0002)	0.316 (0.0002)	**0.314 (0.0002)**
Year MSD	13.49 (0.09)	17.12 (0.12)	16.47 (0.11)	**7.12 (0.01)**	7.27 (0.01)	7.99 (0.01)	42.95 (0.07)
Zurich	63.4 (0.05)	63.49 (0.05)	63.61 (0.05)	40.75 (0.05)	45.7 (0.05)	**39.12 (0.05)**	62.94 (0.06)
Average Rank	4.67	6	5.33	1.5	2.5	2.33	5

Table 4.6. Performance of different ATSER ensembles with varying ensemble sizes. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the best performance.

		ATSER-Seeds	ATSER-RForest	ATSER-OBag
10 Trees	Airline 08	**14.97 (0.02)**	15.07 (0.02)	16.39 (0.01)
	Friedman	**1.36 (0.01)**	1.39 (0.01)	1.40 (0.01)
	Housing	**55857 (25)**	55933 (21)	59268 (52)
	Hyperplane	**0.28 (0.0001)**	0.289 (0.0002)	0.293 (0.0002)
	Year MSD	**7.52 (0.01)**	7.71 (0.01)	8.44 (0.01)
	Zurich	39.90 (0.11)	45.20 (0.10)	**38.78 (0.05)**
	Average Rank	1.17	2.17	2.67
25 Trees	Airline 08	**14.76 (0.008)**	14.85 (0.008)	16.03 (0.007)
	Friedman	**1.32 (0.001)**	1.35 (0.001)	1.33 (0.001)
	Housing	**55720 (46.3)**	55764 (46.4)	58740 (46.6)
	Hyperplane	**0.272 (0.0001)**	0.282 (0.0002)	0.277 (0.0001)
	Year MSD	**6.7 (0.007)**	6.85 (0.007)	7.2 (0.006)
	Zurich	38.66 (0.047)	43.89 (0.048)	**37.29 (0.047)**
	Average Rank	1.17	2.5	2.33
50 Trees	Airline	**14.69 (0.008)**	14.77 (0.008)	15.91 (0.007)
	Friedman	1.31 (0.001)	1.34 (0.001)	**1.3 (0.001)**
	Housing	**55669 (46.3)**	55707 (46.4)	58525 (46.6)
	Hyperplane	**0.27 (0.0001)**	0.279 (0.0001)	0.271 (0.0001)
	Year MSD	**6.39 (0.007)**	6.52 (0.007)	6.72 (0.006)
	Zurich	38.25 (0.047)	43.51 (0.048)	**36.78 (0.047)**
	Average Rank	1.33	2.5	2.17

4.5.5 Impact of Ensemble Size

Thus far, we have been chosen small ensembles of 10 trees, enabling comparison with previous works. However, we would typically expect improved performance with a larger ensemble size than this, especially in cases with a higher degree of randomization, such as the ATSER ensembles. Consequently, in Table 4.6 we present the performance of the ATSER ensembles with 10, 25 and 50 trees for comparison.

We first note that in every case, increasing the ensemble size leads to improved predictive performance, as can be seen in Figures 4.9 to 4.11. Furthermore, the figures suggest that the relative improvement as the ensemble size increases appears to follow a somewhat similar pattern across the different ensemble approaches, with the largest relative improvements occurring on the stationary YearMSD dataset, and the smallest on the purely categorical Housing dataset.

However, it is worth noting that while all of the ensemble approaches benefit from larger ensembles, and that the ATSER-Seeds ensemble remains the overall best performing method, it is the ATSER-OBag method that benefits the most from increasing the ensemble

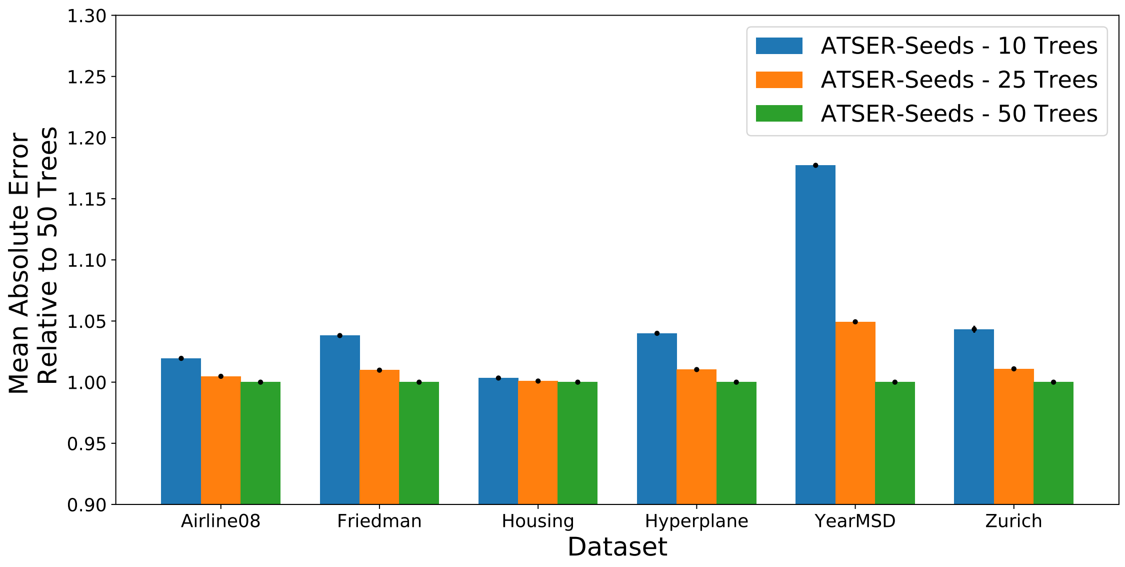


Figure 4.9. Performance of the ATSER-Seeds ensemble at different ensemble sizes. The performance is shown relative to that of a 50 tree ensemble on each dataset.

size. Table 4.6 shows that, relative to the other ensemble methods, ATSER-OBag moves from an average rank of 2.67 to 2.17 as the ensemble size increases from 10 to 50 trees. This improvement relative to the other methods makes sense, since ATSER-OBag has the highest degree of randomization out of the different ensemble approaches, which suggests it ought to benefit the most from increasing the ensemble size.

Overall, we see that all of the ensembles approaches benefit from choosing a larger ensemble, as expected. While ensembles of more than 50 trees may lead to further improvements, the benefit of each additional tree diminishes as the ensemble size increases. This suggests that while ATSER-OBag appears to improve more with ensemble size relative to the other ensemble methods, it may take a long time, or perhaps not even reach, the performance of ATSER-Seeds on many of the datasets, suggesting that ATSER-Seeds is the optimal choice of ensemble method.

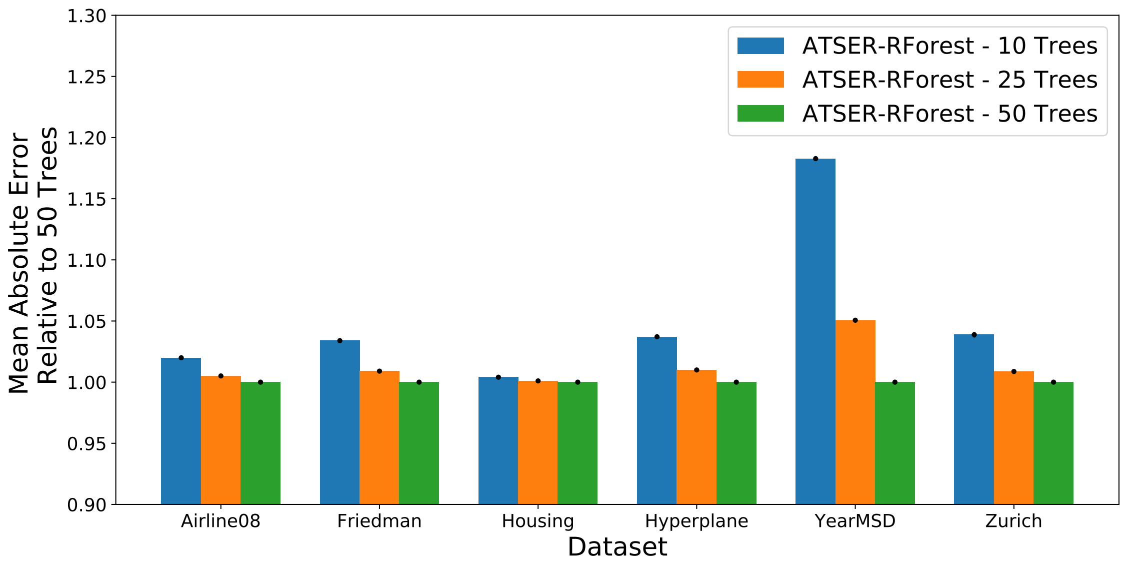


Figure 4.10. Performance of the ATSER-RForest ensemble at different ensemble sizes. The performance is shown relative to that of a 50 tree ensemble on each dataset.

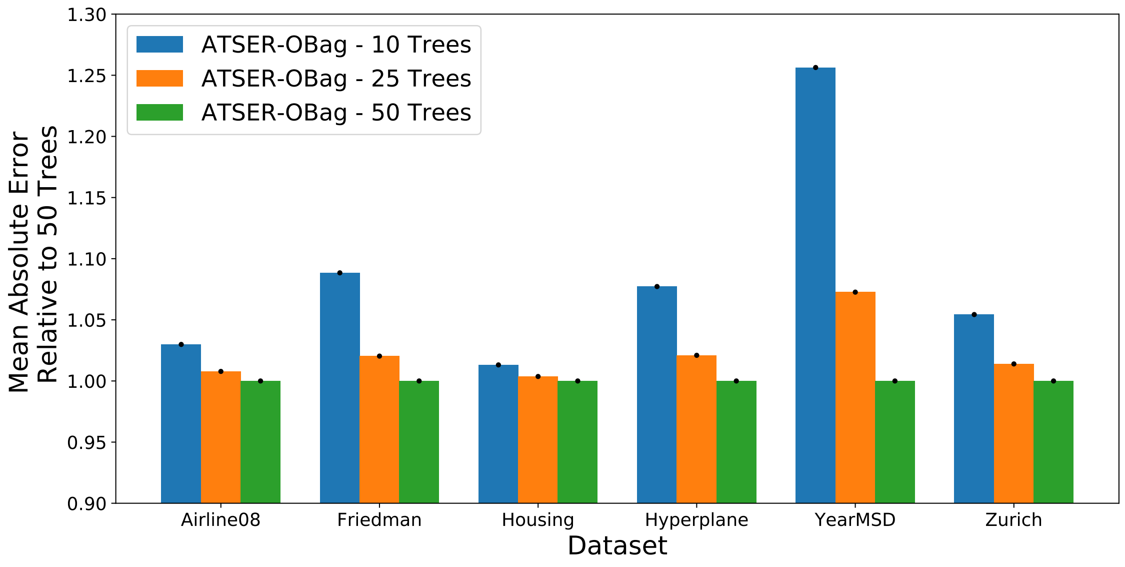
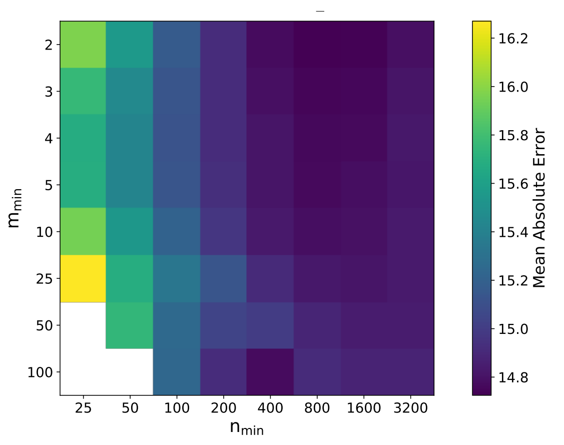
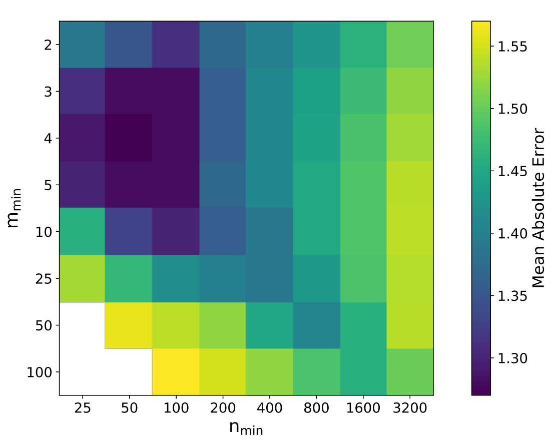


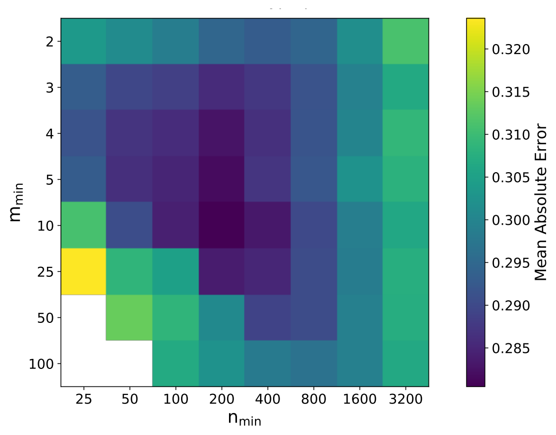
Figure 4.11. Performance of the ATSER-OBag ensemble at different ensemble sizes. The performance is shown relative to that of a 50 tree ensemble on each dataset.



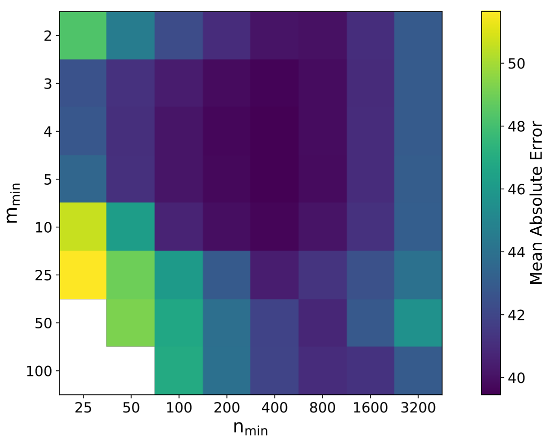
(a) AIRLINE 08



(b) FRIEDMAN



(c) HYPERPLANE



(d) ZURICH

Figure 4.12. Sensitivity of ATSER-Seeds to m_{min} and n_{min} on timeseries datasets.

4.5.6 Sensitivity of Hyperparameters

Although many of the hyperparameters we have chosen are based on those in other works, some hyperparameters, most notably the minimum number of examples before attempting to split at node, m_{min}, and n_{min}, the number of examples to wait until selecting a split, are especially important due to their role in the novel splitting process used in the ATSER algorithm. While we have chosen $n_{min} = 200$ for comparison with previous algorithms, this may not be optimal. We now consider the sensitivity of the results from the best-performing ATSER-Seeds ensemble on the timeseries datasets, as shown in Figure 4.12.

From Figure 4.12, we can see that empirically, the optimal values of m_{min} and n_{min} vary with the dataset. While m_{min} appear to be stable, optimally between 2 to 10, the value of n_{min} is much more sensitive, varying from 50 on the FRIEDMAN dataset (Figure 4.12b) to 800 on the AIRLINE 08 dataset (Figure 4.12a). The surprisingly low optimal value for m_{min} suggests that the performance benefits a lot from the randomness in the splitting procedure. On the other hand, the variation in n_{min} may be due to differences in the level of noise across datasets, with the relatively low-noise FRIEDMAN dataset needing fewer examples to detect a signal and form a meaningful split.

However, it is important to note that despite the optimal values varying across datasets, the variation in performance across different values of m_{min}, and n_{min}, is relatively small, suggesting the ATSER-Seeds algorithm is fairly robust to changes in these parameters. For the chosen values of $n_{min} = 200$ and $m_{min} = 10$ used in this work, the performance is within 10% of the optimum on each dataset.

4.6 Summary

We have introduced the concept of extreme randomisation as embodied in the Extra Trees algorithm for modelling streaming data which, when compared to other contemporary models, results in both improved predictive performance and computational efficiency arising from the simpler splitting procedure. We have incorporated a framework for handling nominal variables with large numbers of categories in streaming data, allowing for a wider range of applications.

Taking our new model as a base learner, we have proposed several ensemble algorithms for regression streams, allowing further improvements in performance, particularly when parallel hardware is available as if often the case on modern machines. While we have worked explicitly in a regression context, we have also adapted ATSER for use in classification scenarios, which will be described in the next chapter.

Chapter 5

Adaptive Categorization Trees for Streaming with Extreme Randomization

When seeking models with strong predictive performance, practitioners concerned with data stream mining face an additional challenge in the volume of data available. As new data is constantly being generated, models must be able to train rapidly, and update in accordance with any changes in the data stream. While many models seek to do this, the most prominent among them is the Very Fast Decision Tree (VFDT), introduced by Domingos and Hulten [2000] which acts as the classifier used for streaming data. However, despite being known to be ill-suited to nonstationary data, the VFDT is still commonly used as the base learner in many ensembles designed for such scenarios. In this chapter, we introduce an adaption of the ATSER algorithm to a categorization scenario: the Adaptive Categorization Trees for Streaming with Extreme Randomization (ACTSER) algorithm. ACTSER is able to act as a base learner for ensembles designed for nonstationary data and offers a fast and effective alternative to the VFDT.

The ACTSER algorithm again applies the concept of extremely randomised (Extra) trees to a streaming scenario, which are known for both their speed and their ability to leverage randomness in the base learners to improve ensemble performance. We show that this allows individual ACTSER trees to include a change adaption mechanism whilst still maintaining similar speeds to the VFDT. Furthermore, we compare the performance of a single ACTSER tree with the VFDT and the contemporary fast alternative EFDT [Manapragada et al., 2018]. We also demonstrate the strong performance of ensembles of ACTSER trees relative to the contemporary Adaptive Random Forest (ARF) ensemble algorithm [Gomes et al., 2017a], finding that it generally has stronger predictive performance, whilst also being faster.

The remainder of this chapter proceeds as follows. In Section 5.1 we briefly re-review the related works designed for data stream classification that we saw in Chapter 3, before specifying the structure of the ACTSER algorithm in Section 5.2 and how to compose ensembles in Section 5.3. We then describe the data used for evaluation in Section 5.4, followed by a discussion of results in Section 5.5 and concluding remarks in Section 5.6.

5.1 Online Categorization

Due to the importance of big data streams in numerous fields, there have been many attempts to create algorithms which can handle concept drifting data. As we have seen in Section 3.2.3, the original VFDT algorithm, introduced by Domingos and Hulten [2000], incrementally trains a classification tree and is a popular approach due to its fast construction through the use of Hoeffding bounds [Hoeffding, 1963].

While the VFDT offers a fast method for constructing accurate trees, it is only applicable to stationary data. To account for concept-drifting data, Hulten et al. [2001] suggested an adaption known as the Concept-Adapting VFDT (CVFDT), which monitors nodes in the tree for change and trains alternative trees if an alternative splitting attribute at a node appears to be better. However, as it maintains trees relative to a current window of examples, it is expensive and slow to update compared to the VFDT.

Due to the time and memory costs of maintaining the CVFDT, a number of other approaches have been suggested for handling concept drifting data. Some methods such as Hoeffding Adaptive Trees (HAT) [Bifet and Gavald, 2009] similarly focus on keeping a VFDT relevant to current data, while other approaches attempt to simply update the tree faster. Hoeffding Option Trees (HOT), proposed by Pfahringer et al. [2007], achieve this through the use option nodes, which allow the tree to be split on multiple features at the same node, effectively turning the tree into an ensemble. A more recent adaption is the Extremely Fast Decision Tree (EFDT) [Manapragada et al., 2018], which differs from the VFDT by splitting when information gain is non-zero rather than when the best split outperforms the second best split. While not the primary motivation for its design, the authors note it has the capability to handle concept drift.

Despite the number of other learners available, VFDTs have remained popular as base learners in ensembles since, despite their inability to handle concept drift, they are simpler and faster than other approaches. Many concept drift ensembles use block-based approaches with VFDTs as base learners, such as the Streaming Ensemble Algorithm (SEA), Accuracy Weighted Ensemble (AWE) and Accuracy Updated Ensemble (AUE) [Street and Kim, 2001; Wang et al., 2003; Brzeziński and Stefanowski, 2011]. These ensembles work by breaking the data into batches and training a VFDT on each. The classifiers are then added to the ensemble, and the results obtained through the ensemble's respective voting scheme. In these block-based approaches, concept drift is usually handled by weighting each ensemble member's vote based on its performance. However, the performance of such ensembles is often disappointing due to the lack of exposure to the data in each of the base classifiers and the inability of each base classifier to react to concept drift independently.

More recently ensembles offering solutions to both of these problems have been proposed. The Adaptive Random Forest algorithm (ARF) [Gomes et al., 2017a] trains each tree on all of the available data in a random forest style approach [Breiman, 2001]. Each base learner is trained following the online bagging approach suggested by Oza and Russell [2001].

In the Adaptive Classification Trees for Streaming with Extreme Randomization (ACTSER) algorithm, we expand on previous works by introducing further randomization through a similar splitting mechanism to extremely randomised (Extra) trees [Geurts et al., 2006]. Extra Trees work by randomly choosing one possible split point for each feature considered, and splitting on the best of them. By adapting this approach for the ACTSER algorithm, we are able to increase the diversity of our ensembles and in turn improve predictive performance. Furthermore, this approach allows the easy inclusion of nominal features in the training process, which is particularly expensive in the traditional VFDT.

Additionally, the ACTSER algorithm includes the use of McNemar's test [McNemar, 1947] for deciding whether to adopt an alternative tree, exploiting the fact that the predictions of a base classifier and an alternative tree are paired nominal data. This provides us a simple and theoretically motivated assurance that any adaptions are appropriate.

5.2 The ACTSER Algorithm

The basic construction of the ACTSER Tree follows that of the VFDT, but with semi-randomized splitting decisions based on Extra trees [Geurts et al., 2006]. Starting from a single root node, the tree is constructed incrementally. Examples are assumed to arrive successively, with each example containing a single target y_t and a set of features $\mathbf{x}_t$. Predictions are made in the leaves using an adaptive Naïve Bayes classifier. After each split node is constructed, a change detector begins monitoring the predictions and, in the event that change is detected, an alternative tree is initiated alongside the original. After a period of growth, the alternative tree either replaces the original, or is discarded, based on how well it performs. An overview of the ACTSER algorithm can be seen in Algorithm 5.1.

Algorithm 5.1 Overview of the ACTSER Algorithm

```
 1: $\mathcal{T} := \emptyset$                                        ▷ Initialise empty tree
 2: for $t := 1, 2, \ldots$ do                                          ▷ Iterate over data stream
 3:     input: $\mathbf{x}_t$
 4:     $y'_t :=$ MakePrediction($\mathcal{T}, \mathbf{x}_t$)
 5:     yield: $y'_t$                                                 ▷ Return prediction
 6:     input: $y_t$                                                  ▷ Observe $y_t$ for training
 7:     $\mathcal{N} :=$ GetRoot($\mathcal{T}$)                                    ▷ Start at root node
 8:     while IsNotLeaf($\mathcal{N}$) do                                  ▷ Descend to leaf
 9:         $\mathcal{N} :=$ UpdateNodeStatistics($\mathcal{N}, y_t$)
10:         if ChangeDetected($\mathcal{N}$) = False then
11:             $\mathcal{N} :=$ TestForChange($\mathcal{N}, y_t, y'_t$)
12:         else
13:             $\mathcal{T}, \mathcal{N} :=$ Adapt($\mathcal{T}, \mathcal{N}, \mathbf{x}_t, y_t, y'_t$)
14:         end if
15:         $\mathcal{N} :=$ GetChild($\mathcal{N}, \mathbf{x_t}$)                          ▷ Descend one node
16:     end while
17:     $\mathcal{N} :=$ UpdateLeaf($\mathcal{N}, \mathbf{x_t}, y_t, y'_t$)
18:     if GetSeen($\mathcal{N}$) $\bmod n_{min} = 0$ then
19:         $\mathcal{T} :=$ AttemptToSplit($\mathcal{T}, \mathcal{N}$)
20:     end if
21: end for
```

5.2.1 Splitting

In the ACTSER algorithm, splits are created in two stages, leading to a semi-randomized splitting process. The randomization is introduced during the first stage, which occurs after the first m_{min} examples at each node arrive. At this point, one possible split point is randomly chosen for each feature. For numeric features, candidate splits drawn from a uniform distribution between the largest and smallest value seen, while for categorical features, every possible binary split between the observed categories is considered with equal probability.

Unlike other algorithms, ACTSER is able to handle many-category nominal attributes easily, since by simply selecting the splits randomly, we eliminate the need for an exhaustive search of all 2^{c-1} possible splits for an attribute with c classes.

The second stage in the splitting process occurs after n_{min} total examples have been seen. The best split among those randomly chosen in the first stage is chosen as the split with the greatest Information Gain (IG):

$$IG(s) = E_P - \frac{N_L}{N_P}E_L - \frac{N_R}{N_P}E_R \tag{5.1}$$

with

$$E_k = -\sum_i p_{ik} \log_2 (p_{ik}) \tag{5.2}$$

where the subscript k indicates whether the quantity relates to a parent (P), left child (L) or right child (R) resulting from split s, N_k and E_k are the number and entropy of examples at node k, and p_{ik} represents the probability of each target class i occurring at node k.

After determining the apparent best split, the Hoeffding bound [Hoeffding, 1963] is used to determine whether, under the assumption that the data stream is stationary, the split will remain the best as more examples are seen. As explained in Chapter 3, the Hoeffding bound is calculated according to:

$$\epsilon = \sqrt{\frac{R^2 \ln (1/\delta)}{2N}}$$

and states that the empirical mean of a random variable with range R and N examples is, with probability $1 - \delta$, within ϵ of the true mean. We follow Domingos and Hulten [2000] by first considering the difference in IG between the best and second best splits: $\Delta IG = IG(s_{best}) - IG(s_{second})$. By treating this quantity as the deviation of a random variable from the true mean, we can determine that s_{best} will remain the best if $\Delta IG > \epsilon$.

If using the Hoeffding bound we are able to obtain assurance that there is a single best split, the split is accepted and the splitting process begins again in the resulting child nodes. Conversely, if we are not able to obtain this guarantee, we wait for another n_{min} examples to arrive before repeating the second stage of the splitting process, until eventually a single best split is obtained. However, it is possible to be stuck with multiple similarly performing splits for a long time. To deal with this, if we are unable to determine a single best split after the Hoeffding bound reaches a threshold τ, a split is randomly chosen from the best splits since it can be assumed they are all equally discriminative.

In our experiments, we follow Domingos and Hulten [2000] in choosing $\tau = 0.05$, $\delta = 10^{-7}$ and $n_{min} = 200$ for the VFDT, EFDT and ACTSER trees. We further choose $m_{min} = 10$ for the ACTSER trees. While we use $n_{min} = 200$ in the ACTSER trees for comparability, we recommend smaller values such as $n_{min} = 50$. This allows the trees to train faster by taking advantage of the first stage of our split selection process which means that, compared to other algorithms such as the VFDT, we generally have fewer splits to compare between and therefore need less information to form a split.

Maintaining Sufficient Statistics

In general, we want to maintain only the minimum sufficient statistics at each stage of the splitting process. Consequently, we only retain a small amount of information from each example at each node, in the form of summary statistics. After the statistics are updated, the original example is instantly discarded.

From Equations (5.1) and (5.2) we can see that to calculate the IG of a split, only the target class frequency distributions in each of the child nodes are required. Consequently, we maintain the information to construct them from the initial creation of a node until a split is made. For numeric features this means we need to maintain all pairs of feature and target values until a random split point is chosen after m_{min} examples have been seen, at which point we create and maintain the target distributions for each of the two child nodes.

On the other hand, for nominal splits we can take advantage of their categorical nature by starting to create the split and child distributions from the first example seen, by randomly assigning categories to either the left or right child when they are first observed.

5.2.2 Modelling in the Leaves

As with many other tree based models, we follow Holmes et al. [2005] in adopting an adaptive Naïve Bayes predictor in the leaves of the tree, since they find it virtually guarantees improved performance relative to a simple Majority Classifier. However, it is also perhaps the most time consuming and expensive part of the algorithm. This is because for each feature at each leaf, adaptive Naïve Bayes classifiers require the feature value distribution to be kept for each target class observed.

The adaptive Naïve Bayes model works by combining of a Majority Class (MC) and a Naïve Bayes (NB) predictor. As each example is seen, both an MC and an NB prediction are made, and the number of correct predictions tracked. The historically best performing prediction is then returned as the adaptive Naïve Bayes prediction.

While the MC prediction works by simply predicting the most frequently observed class at the leaf, the NB prediction requires more information to be stored. Assuming a target, y, is one of k possible classes associated with a set of n features, x_j, $j = 1, \dots, J$ each of which takes on a value i, we can use Bayes' theorem to find the probability of a specific class y_k according to:

$$p(y_k|\mathbf{x}) = \frac{p(\mathbf{x} \,|\, y_k)p(y_k)}{p(\mathbf{x})}.$$

As the $p(\mathbf{x})$ term is independent of y, it is effectively just a normalising factor. Therefore, by rewriting

$$p(\mathbf{x} \,|\, y_k)p(y_k) = p(y_k, \mathbf{x}) = p(y_k) \prod_{j=1}^{J} p(x_j \,|\, y_k),$$

which uses the assumption that features are conditionally independent given y_k, and reintroducing the normalizing factor $Z = \sum_k p(\mathbf{x} \,|\, y_k)p(y_k)$, gives

$$p(y_k \,|\, \mathbf{x}) = \frac{1}{Z} p(y_k) \prod_{j=1}^{J} p(x_j \,|\, y_k).$$

Consequently, to construct the NB predictor the related data needs to be stored. For the $p(y_k)$ terms, a count of each class y_k is stored, while $p(x_j|y_k)$ is estimated differently depending on the feature type. If the feature, x_j is nominal, $p(x_j|y_k)$ can be found by incrementing a count for each time x_j takes a value i and the category and y_k is seen. On the other hand, if x_j is numeric, a Gaussian estimator is used such that $p(x_j \,|\, y_k) = \mathcal{N}(x_j; \mu_{jk}, \sigma^2_{jk})$, meaning that for numeric features the observed mean and variance of x_j when $y = y_k$, denoted μ_{jk} and σ^2_{jk}, are stored.

5.2.3 Reacting to Changes

As with all algorithms designed for nonstationary data, it is important to be able to react to changes in the data stream. In the ACTSER algorithm, as each split is made, a change detector is installed at the split node and proceeds to monitor for change as new examples are seen. If change is detected, a warning will be raised and trigger the training of a possible alternative tree rooted at the node, in parallel with the original tree. During this time, change detection is halted, allowing at most one alternative tree to be trained per node. Predictions from both trees are monitored, and the best performing tree is adopted. Should neither tree prove to be better after an allotted training period, the alternative tree is adopted.

Change Detection

Similarly to in the ATSER algorithm (Section 4.2.2), we detect changes in the tree using an adaption of the Page-Hinkley (PH) test [Mouss et al., 2004], as shown in Algorithm 5.2. The test is designed for numeric data and works by tracking the difference between the normalized prediction error, d_t and its mean over time, $\bar{d}_t = \frac{1}{T}\sum_{t=1}^{T} d_t$. At each time t, the relative error is then designated as $m_t = \sum_{t=1}^{T}(d_t - \bar{d}_t - \alpha)$, where α is a user defined threshold for the sensitivity of the detector to changes. Should the value of the relative error exceed the minimum value seen, M_t, by more than a user defined threshold, λ, the detector will trigger. In our experiments, we follow Ikonomovska et al. [2011b] who find that choosing $\alpha = 0.005$ and $\lambda = 50$ is effective for a variety of different datasets.

As our target, y_t, is a categorical variable, to obtain the normalised prediction error we begin by modelling each prediction error as the outcome of a Bernoulli trial, such that the error is 0 if the prediction, y'_t, is correct, and 1 otherwise. By tracking the number

Algorithm 5.2 The `TestForChange` Function

```
 1: input: $\mathcal{N}, y_t, y'_t$                                  ▷ Input split node, target and prediction
 2: $\alpha, \lambda :=$ GetParameters($\mathcal{N}$)                ▷ Get user defined parameters
 3: $q, n, \sigma, \bar{d}_{t-1}, m_{t-1}, M_{t-1} :=$ GetPastStatistics($\mathcal{N}$)
 4: $n := n + 1$                                                     ▷ Increment count of examples seen
 5: $q := q + 1 - \delta(y_t, y'_t)$                                 ▷ Update count of incorrect predictions
 6: if $n < \gamma$ then
 7:     $\sigma := \sqrt{\frac{q}{n}(1 - \frac{q}{n})}$
 8: end if
 9: $d_t :== \dfrac{1 - \delta(y_t, y'_t) - \frac{q}{n}}{\sigma}$
10: $\bar{d}_t := \dfrac{(t-1)\bar{d}_{t-1} + d_t}{t}$                 ▷ Update mean prediction error
11: $m_t := m_{t-1} + (d_t - \bar{d}_t) - \alpha$                   ▷ Update relative accuracy
12: $M_t := M_{t-1}$
13: if $m_t < M_t$ : then
14:     $M_t := m_t$
15: end if
16: if $(m_t - M_t) > \lambda$ then
17:     SetChangeDetected(N)
18: end if
19: $\mathcal{N} :=$ StoreStatistics($\mathcal{N}, q, n, \sigma, \bar{d}_t, m_t, M_t$)
20: yield: $\mathcal{N}$
```

of correct and incorrect predictions, we can empirically determine the probability, p_t, of being correct at time t. This in turn allows the calculation of the mean, $\mu_t = (1 - p_t)/t$ and standard deviation $\sigma_t = \sqrt{p_t(1 - p_t)}$ of the prediction errors, thus letting us normalise the predictions error according to:

$$d_t = \frac{1 - \delta(y_t, y'_t) - \mu_t}{\sigma_t},$$

where δ is the Kronecker delta symbol.

While it has the advantage of requiring very few calculations, the PH test is prone to err in two main ways; triggering false alarms due to a small number of large outliers, and missing changes induced by gradual drift. While the triggering of false alarms is less problematic with categorical data than regression, since the maximum raw error is 1, it is conversely easier to miss changes induced by gradual drift. This is primarily due to gradual drift leading to σ_t slowly increasing and reducing d_t and in turn m_t. To counter this effect, we use the assumption that the data is stationary to fix the standard deviation after a sufficient number of examples, γ, have been seen at the node, such that

$$\sigma_t = \sigma_\gamma : \quad t > \gamma.$$

In our experiments, we find $\gamma = 250$ to be sufficient.

Table 5.1. The contingency table used in McNemar's test when determining how to adapt the tree.

	ORG CORRECT	ORG INCORRECT
ALT CORRECT	a	b
ALT INORRECT	c	d

Change Adaption

After a change is detected, any further change detection at the node is stopped, and an alternative (Alt) tree is grown (from the node) in parallel with the original (Org) tree for a period of time, T_{min}. In this work, we use $T_{min} = 150$, following Ikonomovska et al. [2011b] who use a similar alternative tree adaption approach, but in a regression scenario.

After T_{min} examples have been seen, the predictive ability of the trees is compared. Should one tree perform better than the other, the better tree is kept and the other tree discarded, at which point change detection at the node is restarted. On the other hand, if there is not enough evidence to suggest that one tree is performing better than the other, both trees are kept for a further T_{min} and the process repeated. If neither tree is found to be better after a total time of $10T_{min}$ since the growth of the Alt tree began, the Alt tree is treated as the better performing tree and kept in place of the Org tree for two reasons. Firstly, it may be more appropriate since we have previously suspected the presence of change while, secondly, the alternative tree has been grown for less time and will therefore likely be smaller, meaning adopting it will reduce time and memory usage. The adaption process is detailed in Algorithm 5.3.

To determine whether one tree is performing better than the other, we begin by treating the incoming examples as stationary. After $T_{min}/2$ examples have been seen, we begin monitoring each tree. For each example, we monitor each tree's accuracy: whether its predictions are correct or incorrect. As both trees are training with and making predictions on the same sample, the results form a set of paired nominal data. Consequently, we can test whether one tree is performing better than the other by applying McNemar's test to the data [McNemar, 1947].

We therefore implement McNemar's test by monitoring the accuracy of each tree on each example, allowing us to draw a 2x2 contingency table as shown in Table 5.1. The following hypotheses are then constructed: $H_0 : p_b = p_c, \quad H_1 : p_b \neq p_c$. b represents the number of correct predictions made by the Alt tree that the Org tree predicted incorrectly, and c number of correct predictions made by the Alt tree that the Org tree predicted incorrectly. p_i represents the probability of the quantity i in Table 5.1 being observed. The test statistic is calculated according to $\chi^2 = (b - c)^2/(b + c)$, and is distributed according to a chi-squared distribution with 1 degree of freedom. The test is two-tailed, and in our experiments, we test whether a tree is better at the 10% level, thus rejecting the null

Algorithm 5.3 The `Adapt` function

```
1: input: $\mathcal{T}, \mathcal{N}, \mathbf{x_t}, y_t, y'_t$    ▷ tree, split node, features, target, pred
2: $b, c :=$ GetCounts($\mathcal{N}$)    ▷ Get counts from node
3: $\mathcal{T}' :=$ GetAltSubtree($\mathcal{N}$)    ▷ Alternative tree stored at node
4: $y''_t :=$ MakePrediction($\mathcal{T}', \mathbf{x_t}$)
5: $\mathcal{T}' :=$ TrainAltSubtree($\mathcal{T}', \mathbf{x_t}, y_t$)
6: if $y'_t \neq y''_t$ and $y_t$ in $\{y'_t, y''_t\}$ then    ▷ If only one tree correct
7:     if $y_t = y''_t$ then    ▷ If alternative tree correct
8:         $b := b + 1$
9:     else    ▷ If original tree correct
10:        $c := c + 1$
11:    end if
12: end if
13: $\chi^2 = \frac{(b-c)^2}{b+c}$    ▷ Calculate test statistic
14: if $\chi^2 > 3.841$ then    ▷ If alternative tree better
15:     $\mathcal{T}, \mathcal{N} :=$ UseAltSubtree($\mathcal{T}, \mathcal{T}', \mathcal{N}$)
16:     UnsetChangeDetected(N)
17:     $b, c := 0, 0$
18: end if
19: if $\chi^2 < -3.841$ then    ▷ If original tree better
20:     $\mathcal{T}, \mathcal{N} :=$ DiscardAltSubtree($\mathcal{T}, \mathcal{T}', \mathcal{N}$)
21:     UnsetChangeDetected(N)
22:     $b, c := 0, 0$
23: end if
24: if GetSeen($\mathcal{T}'$) $\geq 10n_{min}$ then    ▷ If max examples seen
25:     $\mathcal{T}, \mathcal{N} :=$ UseAltSubtree($\mathcal{T}, \mathcal{T}', \mathcal{N}$)
26:     UnsetChangeDetected(N)
27:     $b, c := 0, 0$
28: end if
29: $\mathcal{N} :=$ KeepStatistics($\mathcal{T}', \mathcal{N}, b, c$)
30: yield: $\mathcal{T}, \mathcal{N}$
```

hypothesis if $\chi^2 > 3.841$ or $\chi^2 < -3.841$.

While slower than simply pruning the tree as soon as change is detected, this adaption strategy is beneficial since having an empirical comparison between the alternative and the original tree corrects for false alarms in the detector. This occurs since a new tree will only be adopted if it is not worse than the original. This strategy is also helpful even if a true change is detected, as predictions from a pruned tree may be worse compared to those of the original tree, so continuing to use the original for a time may be beneficial.

5.3 Ensembles

While they may be used as individual learners, ACTSER trees are primarily designed as the base learners in an ensemble. They are intended to function similarly to Extra Trees

Table 5.2. An overview of the datasets used. N represents the number of examples in the dataset. y_n represents the number of target classes, while y_{max} and y_{min} give the population of the most and least populated target classes respectively. NUM and NOM give the number of numeric and nominal features respectively.

DATASET	N	y_n	y_{max}	y_{min}	NUM	NOM
ACTIVITY	33,741,500	7	5,580,900	4,204,346	6	0
FLIGHTDELAY	539,383	2	299,119	240,264	3	4
FOREST_COVER	581,012	7	283,301	2747	54	0
GAS	928,991	3	346,580	276,967	10	0
GMSC	150,000	2	139,974	10,026	10	0
KDDCUP99	4,898,431	23	2,807,886	2	38	0
PAMAP2	2,844,868	13	923,437	47,579	53	0
POKERHAND	829,201	10	415,526	2	5	5
WISDM	1,369,349	6	655,362	2,130	44	0

[Geurts et al., 2006] by introducing randomness into each tree and leveraging the resulting diversity to improve the ensemble prediction.

Unlike many ensembles designed for concept drift [Street and Kim, 2001; Brzeziński and Stefanowski, 2011; Gomes et al., 2017a], we make no attempt to account for drift in the ensemble algorithm itself, instead assuming any changes will be handled by the base learners. Consequently we propose two different ensemble approaches which work very similarly to stationary ensemble algorithms; the first being a seed based ensemble, ACTSER-Seeds, and the second a random forest styled approach, termed ACTSER-RForest.

ACTSER-Seeds is a simply constructed ensemble of multiple ACTSER trees, each started with a different random seed. Each tree sees every example once and distinct trees are formed as a result of different random cut points being draw when splits are formed. The predictions from the individual learners are combined into the ensemble prediction using a majority vote.

On the other hand, ACTSER-RForest extends the randomness within the ensemble. Every tree is again started with a different seed and sees all the data, but, for each split, only a subset of features are considered. For a data stream with F features available, we follow Geurts et al. [2006] who find that randomly choosing $\sqrt{F}$ possible splitting features is most effective. Furthermore, in our experiments we use ensembles of 25 trees for all ensembles considered. As the ensemble predictions should be more robust to an incorrect split that a single learner, we also choose $n_{min} = 25$ for the base learners in the ensembles, enabling the ensembles to train faster.

5.4 Evaluation

In our experiments we use a collection of 9 different datasets for evaluation, a summary of which can be found in Table 5.2. The datasets are intended to be diverse, varying in: shape, size, task and feature composition.

The ACTIVITY [Stisen et al., 2015] and WISDM [Lockhart et al., 2011] datasets both focus on the same task: predicting a user's activity from collections of sensor data read from wireless smartphones and smartwatches. The ACTIVITY dataset was collected in a scripted environment while the WISDM data was collected during real world activity. While the task in the PAMAP2 dataset [Reiss and Stricker, 2012] is similarly to classify a user's activity, the dataset is formed from the readings of 3 specialised inertial measurement unit sensors and a heartrate monitor.

Sensor data is also a component in the GAS dataset [Huerta et al., 2016], which contains readings from eight gas sensors and a temperature and humidity sensor. Each reading is taken whilst the sensors are exposed to either wine, a banana or neither object. The task is therefore to predict which object, if any, is present.

Adapted from the dataset compiled by Ikonomovska et al. [2011b], the FLIGHTDELAY dataset contains information on US domestic flights in 2008. The original regression task was to predict the length of delay for each flight. The task was adapted to a binary classification problem by creating two classes: whether flights are delayed, or not.

The FOREST_COVER dataset [Blackard and Dean, 1999] contains a very different problem. The goal here is to classify which type of forest cover is present from a collection of cartographic variables which describe each 30×30 metre cell. The actual cover type information in the dataset was obtained through US Forest Service Region 2 Resource Information System.

Provided for the Give Me Some Credit competition hosted on kaggle, the goal in the GMSC dataset [Kaggle Competition, 2012] is a common and often challenging task for banks: to predict whether an individual will experience financial distress in the next two years. The prediction is based on a number of current financial indicators.

Another frequent challenge in a different area is the prediction of network intrusions: whether a connection is a normal connection or an attack. The KDDCUP99 dataset [OpenML, 1999] contains records of computer network traffic in which the task is to predict such intrusions.

The final dataset, POKERHAND [OpenML, 2007], contains information on the suits and cards contained in different poker hands. The task is to predict the ranking of the poker hand based on the card information.

Table 5.3. An overview of ACTSER hyperparameters

Description	Symbol	Value
Num examples after which splits are randomly selected	m_{min}	10
Num examples after which the best split is chosen	n_{min}	200
Desired significance for a split	δ	10^{-7}
Split tie threshold for the Hoeffding bound	τ	0.05
Page-Hinckley test threshold	λ	50
Page-Hinckley minimum detection threshold	α	0.005
Num examples to estimate Page-Hinckley σ_γ	γ	250
Alt tree comparison frequency	T_{min}	150

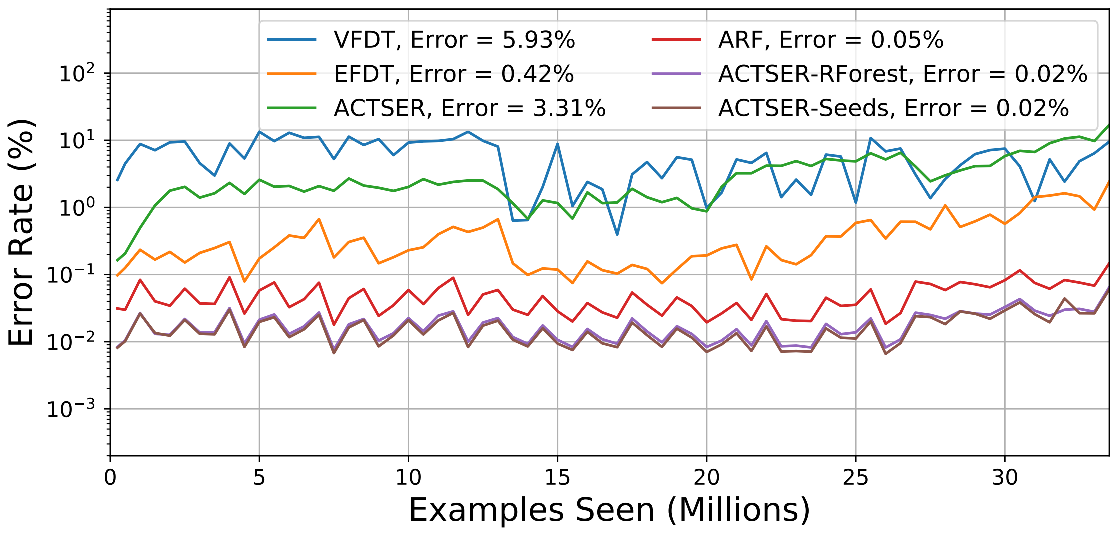


Figure 5.1. Performance on the ACTIVITY dataset. Results are averaged over 1k windows.

Although we have already provided details of the hyperparameters and their values throughout the main test, a summary is available in Table 5.3.

5.5 Results

We report results from a comparison of the ACTSER methods with a selection of classification algorithms. While looking at individual learners, we use the VFDT [Domingos and Hulten, 2000] and EFDT [Manapragada et al., 2018] for comparison, whereas we compare the ensemble methods with the recent ARF ensemble [Gomes et al., 2017a]. To ensure a fair comparison, the ACTSER algorithm and ensembles were implemented in the MOA framework [Bifet et al., 2010a], which already contains implementations of the reference algorithms. All results were obtained on a laptop with an Intel Core i5-6200U dual core 2.3GHz CPU, L2 cache: 513kb, L3 cache: 3Mb, 16Gb RAM @ 2133MHz.

Table 5.4. Classification Error (%) of each model across datasets. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the best performance. [1]

	Single Learners			Ensembles		
	VFDT	EFDT	ACTSER	ARF	ACTSER-Seeds	ACTSER-RForest
Activity	5.93 (0.0)	0.42 (0.0)	3.31 (3.378)	0.05 (0.001)	**0.02 (0.000)**	**0.02 (0.001)**
Flightdelay	34.57 (0.0)	35.05 (0.0)	34.98 (0.119)	34.80 (0.179)	33.06 (0.079)	**31.53 (0.035)**
Forest_cover	21.74 (0.0)	10.31 (0.0)	13.54 (0.059)	5.84 (0.058)	**5.13 (0.008)**	5.79 (0.016)
Gas	1.24 (0.0)	0.26 (0.0)	0.22 (0.032)	0.02 (0.001)	**0.01 (0.000)**	**0.01 (0.000)**
Gmsc	5.49 (0.0)	5.98 (0.0)	5.53 (0.035)	**5.38 (0.007)**	5.54 (0.010)	5.50 (0.011)
Kddcup99	0.11 (0.0)	0.04 (0.0)	0.06 (0.001)	0.01 (0.000)	0.01 (0.000)	**0.01 (0.000)**
Pamap2	5.38 (0.0)	0.23 (0.0)	0.60 (0.028)	0.03 (0.000)	0.02 (0.001)	**0.01 (0.000)**
Pokerhand	22.01 (0.0)	13.5 (0.0)	22.5 (0.151)	17.75 (0.492)	**5.42 (0.024)**	10.81 (0.038)
Wisdm	20.91 (0.0)	8.21 (0.0)	11.66 (0.195)	**5.52 (0.017)**	6.54 (0.023)	6.05 (0.013)
Average Rank	5.11	4.44	4.89	2.78	2	1.78

[1] While the ensemble results on the KDDCUP99 data appear to be tied due to rounding, they are in fact not. To 4 significant figures, the ensemble errors on Kddcup99 are: ARF: 0.0139(0.00036), ACTSER-Seeds: 0.0138(0.00015) and ACTSER-RForest: 0.0098(0.00023)

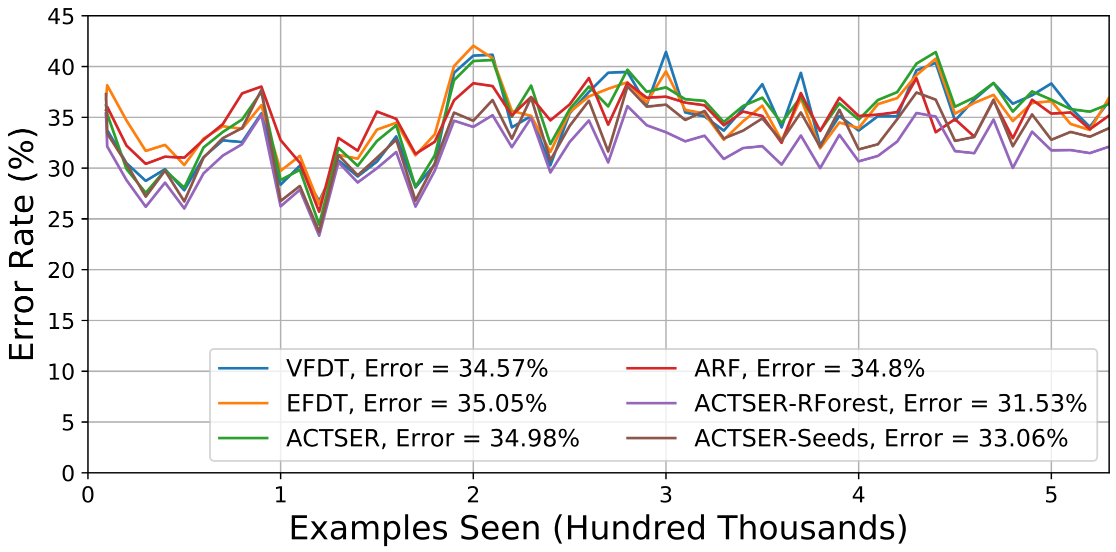


Figure 5.2. Performance on the FLIGHTDELAY dataset. Results are averaged over 1k windows.

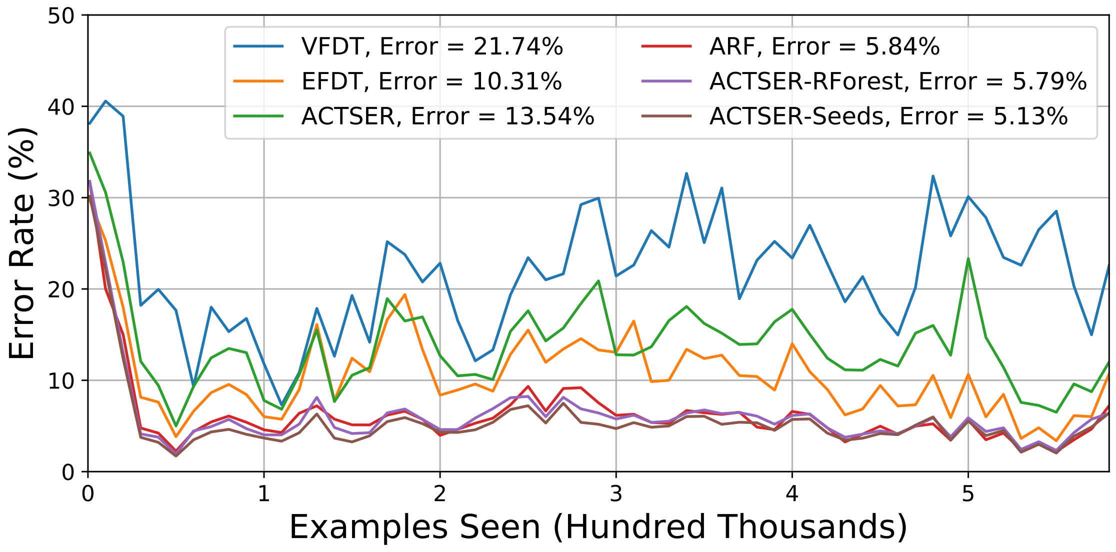


Figure 5.3. Performance on the FOREST_COVER dataset. Results are averaged over 1k windows.

5.5.1 Performance

The performance of each method is summarised in Table 5.4, and individual results for each dataset are shown in Figures 5.1-5.9. The errors were obtained via prequential evaluation [Dawid, 1984], in which predictions are made and the error obtained as each example arrives, after which the example is subsequently used to update the model. To ensure legibility, the error rates shown in the Figures were averaged over windows of results. In general, windows of 10,000 predictions were used, although for the smallest GMSC dataset the window is 1,000 predictions, while for the larger KDDCUP99 and PAMAP2 datasets it is 50,000 predictions and for the largest ACTIVITY dataset it is 500,000 predictions. As there are drastic differences between the best and worst performers, the ACTIVITY, GAS, KDDCUP99 and PAMAP2 results are displayed on a log scale to avoid focusing on the

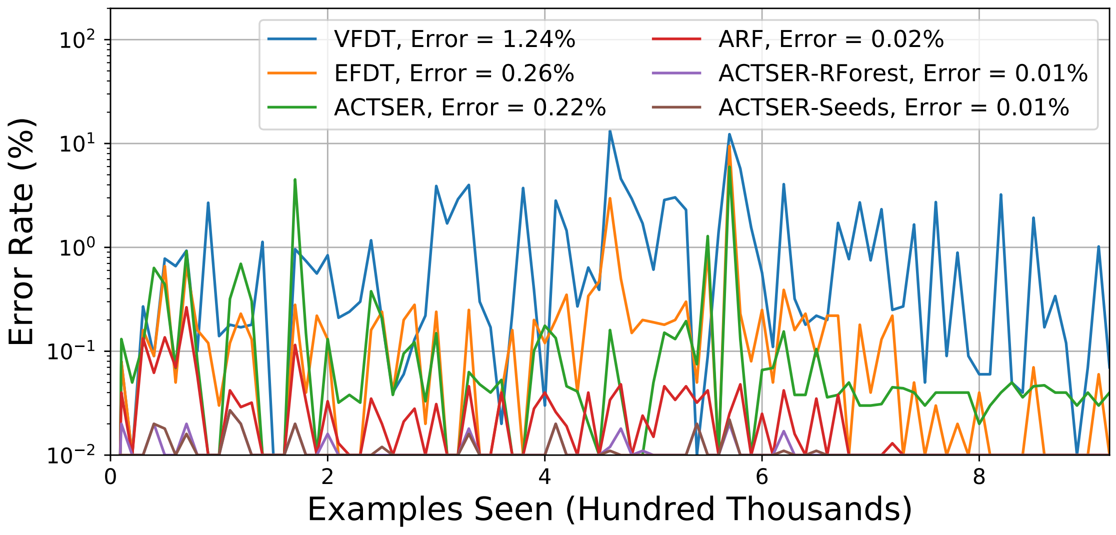


Figure 5.4. Performance on the GAS dataset. Results are averaged over 1k windows.

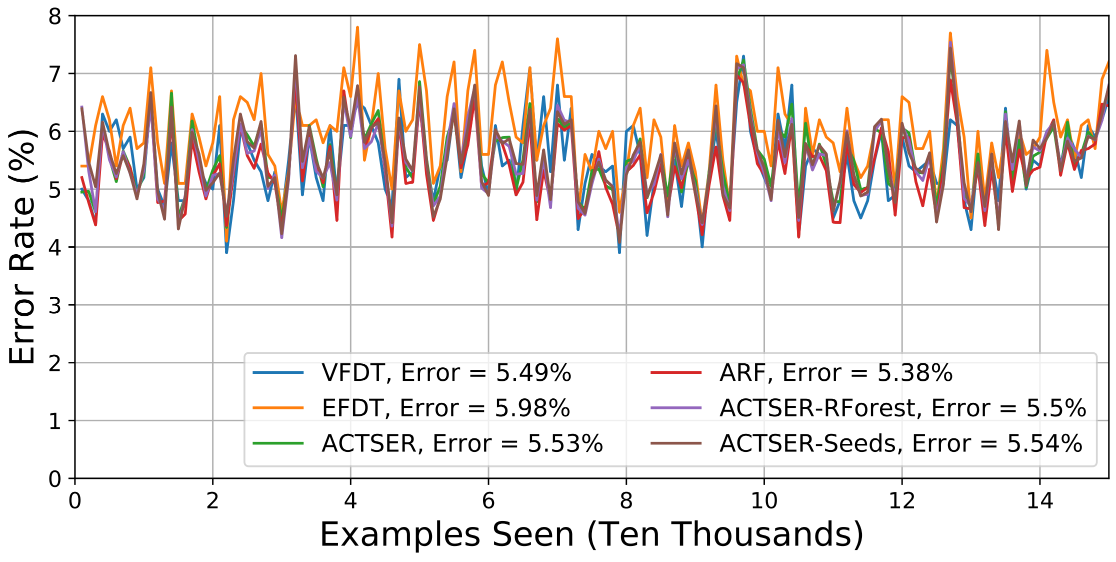


Figure 5.5. Performance on the GMSC dataset. Results are averaged over 1k windows.

worst performing methods. In these cases the error is truncated at the reciprocal of the window size, such that the lower limit shown represents exactly one incorrect prediction per window.

When considering the single learners, although the best is different across every dataset, the EFDT has the best performance on six of the nine. This makes sense since the VFDT is designed for stationary scenarios so has no adaption mechanism, while the randomness of the ACTSER tree makes it less practical outside of an ensemble. Due to its inability to handle change, the VFDT has more variable performance compared to the other learners, as can be seen from the spikes in error in Figures 5.1, 5.4, 5.6 and 5.7.

However, for every dataset it is an ensemble method has the best performance. In fact, all of the ensemble methods usually outperform all of the single learners, although there is

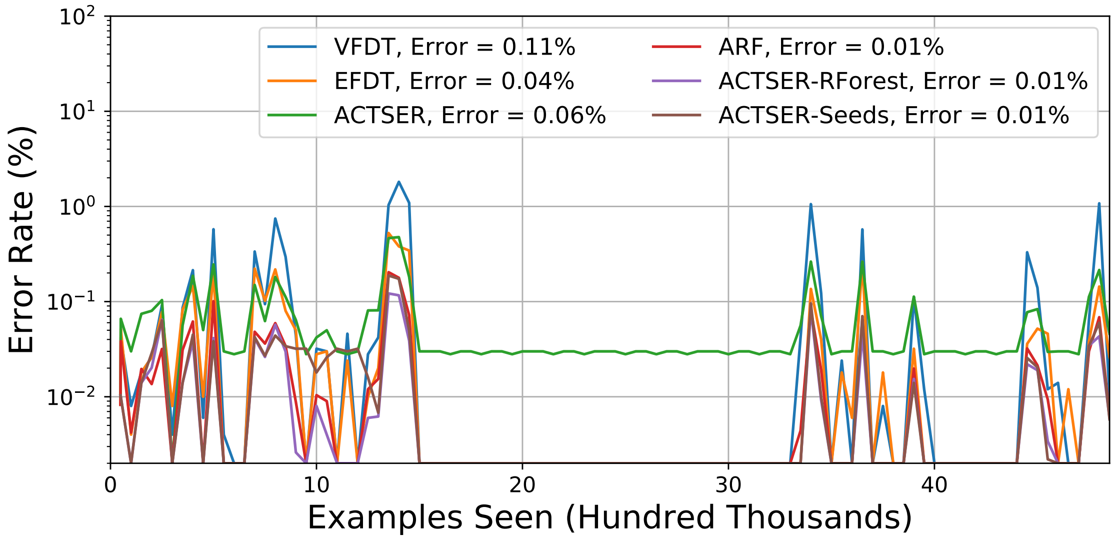


Figure 5.6. Performance on the KDDCUP99 dataset. Results are averaged over 1k windows.

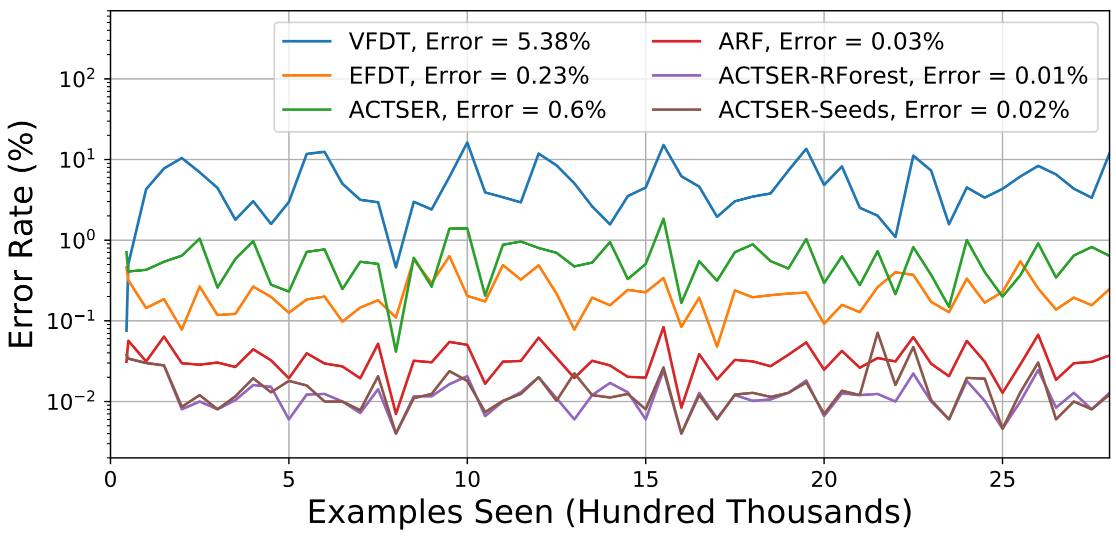


Figure 5.7. Performance on the PAMAP2 dataset. Results are averaged over 1k windows.

an exception on the GMSC dataset for which the VFDT outperforms the ACTSER based ensembles. The fairly constant error rate in Figure 5.5 suggests very little, if any, concept drift is present in this dataset. This would in turn explain the stronger performance of the VFDT and ARF algorithms, as the primary learners in both are designed for stationary data.

On the other hand, the ACTSER based ensembles have the best performance on most of the other datasets, with both ACTSER-Seeds and ACTSER-RForest outperforming all other models on the ACTIVITY, FLIGHTDELAY, FOREST_COVER, GAS, PAMAP2 and POKERHAND datasets. Furthermore, whilst ACTSER-Seeds ties for second best with ARF on the KDDCUP99 dataset, ACTSER-RForest still has the best performance. Other than on GMSC, the only other situation in which they are outperformed is by the ARF ensemble on the WISDM dataset.

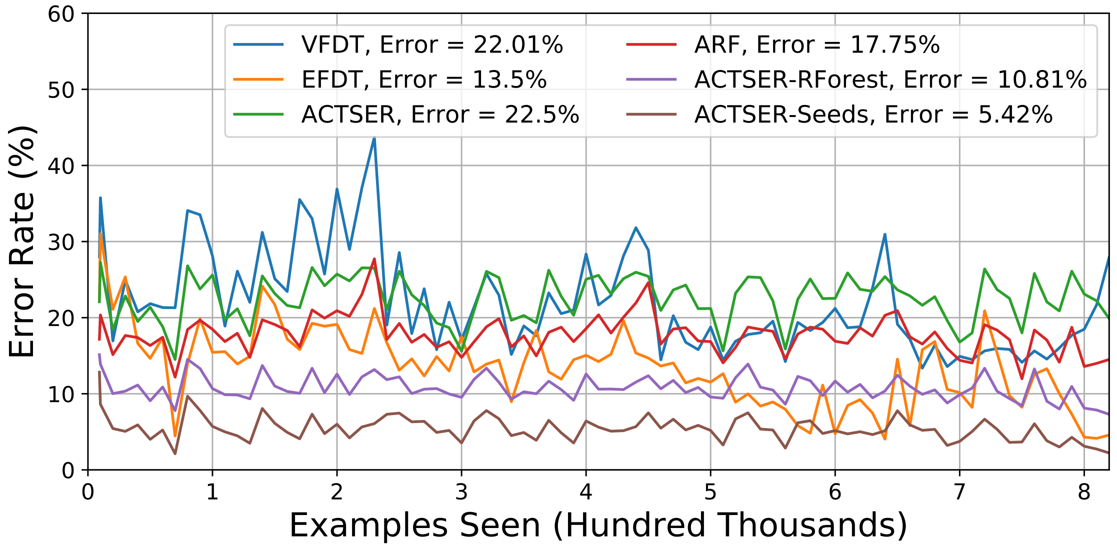


Figure 5.8. Performance on the POKERHAND dataset. Results are averaged over 1k windows.

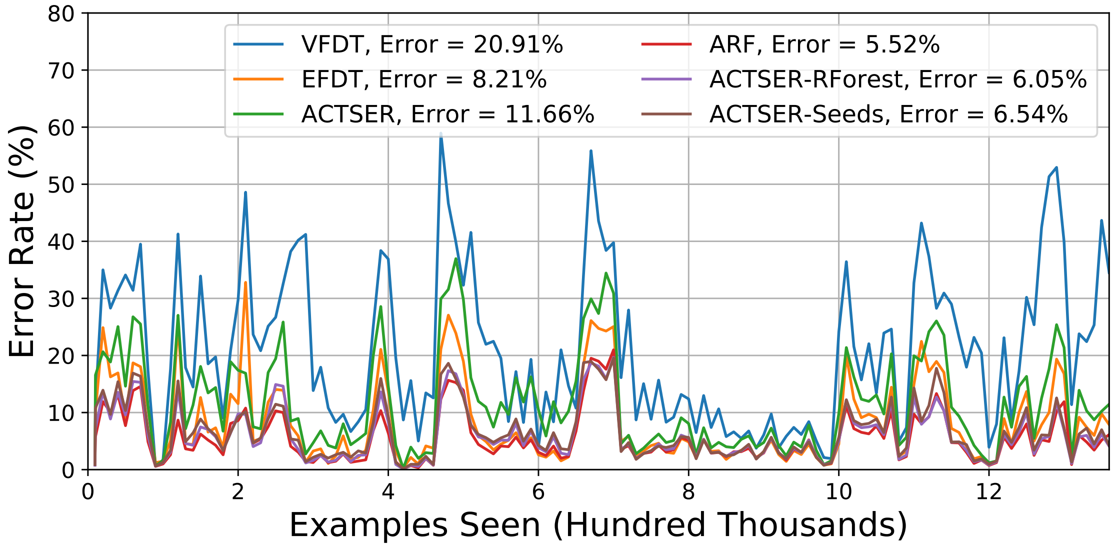


Figure 5.9. Performance on the WISDM dataset. Results are averaged over 1k windows.

It is interesting to note that in cases where an ACTSER based ensemble performs best, it appears to do so from the beginning and continues to do so for the majority of the dataset. This is most prominently visible in Figures 5.2, 5.3, 5.7 and 5.8. This effect may indicate that early performance can be used as an indicator of whether the ACTSER ensembles are appropriate for specific datasets.

5.5.2 Run Times

As might be expected, since none of the ensembles use parallelization, Table 5.5 shows that the single learners all have faster run times than the ensemble approaches. However, as training the ensemble members is an embarrassingly parallel problem, the use of parallelization should be able to reduce the times for the ACTSER based ensembles to be similar to those of a single ACTSER tree. On the other hand, while the ARF algorithm will also improve after parallelization, it will likely remain slower in general, since the online bagging process involved [Oza and Russell, 2001] means each component learner in ARF trains an average of 6 times on each example seen [Gomes et al., 2017a].

A somewhat more surprising observation is that the amongst the single learners, although the EFDT is slower in general, the VFDT and ACTSER algorithms have very similar run times, with either the VFDT or ACTSER algorithm running the fastest in each scenario. This is unexpected since, although not empirically observable, the simpler splitting mechanism may lead us to expect reduced training times for the ACTSER trees as less time is spent on maintaining the split statistics in the decision process [Krawczyk et al., 2017]. However, this may be due to the bulk of the time taken in the algorithm being spent updating the Naïve Bayes predictors in the leaves, which Holmes et al. [2005] find to be significant in comparison to splitting times.

5.5.3 Effect of Leaf Models

Similar to Section 4.5.4, we again examine the effect of removing the leaf models, with the aim of discerning whether the performance of the different algorithms is derived from the trees, or the models in the leaves. Tables 5.7 and 5.6 show the performance and runtimes of the algorithms with majority class classifiers in the leaves instead of Naïve Bayes classifiers. We note that as the EFDT is designed only for use with Naïve Bayes classifiers in the leaves, the results shown for the EFDT still use a Naïve Bayes classifier in the leaves.

As expected, we see that with majority class classifiers in the leaves, the performance of the ACTSER and VFDT single learners worsens, as expected. This change in performance also filters through to the ARF ensemble, which also exhibits poorer performance on every dataset. Interestingly, however, the ACTSER-Seeds and ACTSER-RForest algorithms show similar, and in some cases slightly better, performance with a majority class classifier (such as on the FOREST_COVER dataset). While surprising, this result is similar to that of Section 4.5.4, in which removing the linear models from the ATSER ensemble algorithms may sometimes be beneficial. Although these results suggests that the ACTSER ensemble algorithms are relatively robust to the different choice of leaf model tested, it still remains unclear as to why this is the case.

On the other hand, in terms of run time, the use of majority class classifiers in the leaves leads to all of the algorithms having reduced run times. However, relative to each other, the VFDT, ACTSER and ensemble algorithms all perform similarly to the case with Naïve Bayes classifiers, with ACTSER remaining the fastest on average.

5.6 Summary

In this chapter we have introduced ACTSER, an adaption of the ATSER algorithm presented in Chapter 4 for categorization. This method incrementally trains classification trees using extreme randomization in the splitting process, which uses a novel adaption approach based on McNemar's test. Taking a number of real world datasets, we have demonstrated the comparable performance of the algorithm relative to other popular individual learners, namely the VFDT and EFDT.

We have also provided two ensemble algorithms which use ACTSER trees as base learners: ACTSER-Seeds and ACTSER-RForest. Using the same real world datasets, we have shown that they both have strong predictive performance relative to the contemporary ARF ensemble, as well being significantly faster in most cases. Although we have currently only applied methods to handle change detection to the base learners, possible extensions to the ensemble algorithm may also be lead to improve predictions, such as through weighting schemes similar to those used in Brzeziński and Stefanowski [2011] or through stacked generalization, as in Wolpert [1992].

Table 5.5. CPU time taken by each model across datasets. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the fastest times.

	Single Learners			Ensembles		
	VFDT	EFDT	ACTSER	ARF	ACTSER-Seeds	ACTSER-RForest
Activity	**584 (21.2)**	630 (10.7)	**542 (36.3)**	3158 (36.8)	1853 (33.4)	2444 (160.7)
Flightdelay	**11 (0.3)**	16 (0.2)	13 (0.2)	911 (14.2)	228 (3.3)	278 (1.4)
Forest_cover	**21 (0.2)**	40 (0.6)	24 (0.2)	563 (7.6)	214 (2.7)	208 (2.4)
Gas	17 (0.2)	18 (0.2)	**15 (0.2)**	107 (1.8)	61 (0.5)	60 (0.5)
Gmsc	**3 (0.1)**	4 (0.1)	**3 (0.1)**	61 (1.2)	19 (0.4)	19 (0.2)
Kddcup99	116 (1.3)	143 (1.0)	**113 (0.4)**	683 (10.8)	730 (15.3)	712 (3.9)
Pamap2	193 (0.5)	178 (0.8)	**136 (0.7)**	518 (1.8)	963 (3.5)	969 (6.6)
Pokerhand	**14 (0.1)**	19 (0.2)	**14 (0.1)**	497 (4.7)	103 (0.2)	111 (0.4)
Wisdm	**58 (0.3)**	80 (0.7)	68 (2.0)	585 (4.1)	767 (2.3)	525 (3.8)
Average Rank	1.67	2.89	1.44	5.44	4.78	4.78

Table 5.6. CPU time taken by each model across datasets using majority class predictions rather than Naïve Bayes models in the leaves. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the fastest times.

	Single Learners			Ensembles		
	VFDT	EFDT	ACTSER	ARF	ACTSER-Seeds	ACTSER-RForest
Activity	**486 (5.1)**	611 (5.7)	**489 (15.8)**	2359 (33.7)	1087 (36.4)	1320 (40.8)
Flightdelay	9 (0.6)	15 (0.6)	**7 (0.7)**	957 (12.6)	81 (2.3)	64 (2.5)
Forest_cover	**15 (0.1)**	40 (0.9)	**15 (0.5)**	507 (6.8)	124 (3.1)	140 (3.2)
Gas	15 (0.2)	18 (0.3)	**13 (0.2)**	70 (1.3)	35 (0.2)	35 (0.5)
Gmsc	3 (0.0)	4 (0.1)	**2 (0.1)**	53 (0.4)	13 (0.3)	13 (0.2)
Kdcup99	**97 (0.9)**	138 (1.2)	104 (1.1)	462 (6.9)	606 (1.7)	604 (3.6)
Pamap2	114 (0.7)	173 (0.7)	**112 (0.5)**	288 (3.4)	516 (2.5)	520 (5.9)
Pokerhand	**11 (0.1)**	19 (0.1)	**11 (0.1)**	443 (5.4)	68 (0.5)	74 (0.5)
Wisdm	**38 (0.2)**	78 (0.5)	39 (0.6)	475 (2.9)	237 (3.3)	277 (1.6)
Average Rank	1.56	3	1.44	5.56	4.56	4.89

Table 5.7. Classification Error (%) of each model across datasets using majority class predictions rather than Naïve Bayes models in the leaves. Results are averaged over 10 runs using different random seeds and the standard error is given in brackets. Bold indicates result(s) with the best performance. [2]

	Single Learners			Ensembles		
	VFDT	EFDT	ACTSER	ARF	ACTSER-Seeds	ACTSER-RForest
Activity	31.77 (0.0)	0.42 (0.0)	18.41 (5.581)	0.18 (0.002)	**0.02 (0.000)**	0.02 (0.000)
Flightdelay	38.47 (0.0)	35.05 (0.0)	35.86 (0.247)	35.34 (0.210)	34.21 (0.044)	**32.55 (0.040)**
Forest_cover	36.02 (0.0)	10.31 (0.0)	24.31 (0.338)	7.43 (0.074)	**4.92 (0.013)**	5.39 (0.027)
Gas	19.24 (0.0)	0.26 (0.0)	2.48 (0.171)	0.04 (0.001)	**0.01 (0.000)**	0.01 (0.000)
Gmsc	5.61 (0.0)	5.98 (0.0)	5.53 (0.036)	**5.38 (0.008)**	5.56 (0.009)	5.53 (0.006)
Kdcup99	1.04 (0.0)	0.04 (0.0)	0.8 (0.018)	0.05 (0.003)	0.01 (0.000)	**0.01 (0.000)**
Pamap2	47.35 (0.0)	0.23 (0.0)	6.01 (3.463)	0.12 (0.002)	**0.01 (0.000)**	**0.01 (0.000)**
Pokerhand	29.41 (0.0)	13.5 (0.0)	33.73 (0.093)	20.63 (0.421)	**5.17 (0.109)**	11.79 (0.148)
Wisdm	33.23 (0.0)	8.21 (0.0)	17.17 (0.257)	6.96 (0.030)	6.42 (0.032)	**6.02 (0.010)**
Average Rank	5.78	3.89	4.89	3.11	1.78	1.56

[2] While the ensemble results on the activity, gas and kddcup99 data appear to be tied due to rounding, they are in fact not

Chapter 6

An Approximately Bayesian Online Randomized Forest for Regression

While traditional predictive modelling techniques only focus on the predictive performance of a model, when working with streaming data there is also a need to consider the run time. As discussed in Chapter 3, Domingos and Hulten [2000] introduced the Very Fast Decision Tree (VFDT) which incrementally trains a classification decision tree on a streaming dataset, using the Hoeffding Bound [Hoeffding, 1963] to guarantee the splits of the tree are sufficiently similar to those of a tree trained on a stationary version of the dataset. However, as data streams are often nonstationary, Hulten et al. [2001] introduced the Concept Adapting Very Fast Decision Tree (CVFDT), which adapts to changes in the data stream by replacing questionable parts of the tree.

Although techniques designed for streaming data were developed in a categorisation setting, we have also seen from Chapter 3 that adaptions to regression scenarios have been introduced, with the Fast Incremental Model Trees with Drift Detection (FIMT-DD) [Ikonomovska et al., 2011b] often considered a benchmark. Similar to the CVFDT, the FIMT-DD algorithm depends on the Hoeffding Bound to incrementally train a decision tree with splits guaranteed relative to the performance on a stationary dataset. It also adapts to concept drift through a mechanism to test for changes in the data stream and update affected subtrees accordingly.

Many works for regression streaming data have been based around the FIMT-DD algorithm, with it acting a base learner in many ensemble algorithms. Osojnik et al. [2017] use FIMT-DD trees as base learners for a multi-target regression ensemble, while Gomes et al. [2017a] use them as base learners and leverage the use of randomness in the Adaptive Random Forest (ARF) algorithm to improve performance.

All of the streaming data techniques mentioned so far rely on tree based methods for handling concept drift. However, these methods are often slowed by the constant training process involved and their performance depends on the choices for many different parameters. In this chapter, we focus on exploiting randomness to overcome these issues. We begin by presenting a fully Bayesian concept adapting random forest algorithm for regression, in which we pre-create the structures for trees in the ensemble, before any data is observed. We then dynamically weight the contribution of ensemble members using evidence based weights, demonstrating that this allows the ensemble to automatically adapt to concept drift.

The pre-creation of tree saves time due to the removal of the traditional Hoeffding-based splitting process, whilst also allowing for random projections as opposed to axis-aligned splits. Additionally, the idea of data independent tree structures also means that there is no need to update sub-trees or replace trees themselves in the ensemble, both of which are common but complex and time-consuming features of data streaming ensemble algorithms. This in turn leads to simpler algorithms and data structures and thus a reduced number of model parameters, allowing for greater ease of use. Furthermore, the Bayesian approach gives a principled evidence based method for weighting members of the ensemble.

However, as is common across many Bayesian models, this approach is relatively slow compared to other contemporary algorithms. Consequently, we then develop an approximation of the Bayesian model which we show retains comparable predictive performance, while having reduced run times in comparison to other contemporary approaches. Overall, our contributions in this work can be summarised as follows:

- A Bayesian random forest approach to stream regression, capable of efficiently updating in the presence of concept drift.
- A fully random, incrementally updating, pre-grown ensemble of trees for learning from streaming data.
- The ability to adapt to changes in the data stream with no need for explicit change detection and adaption.
- An approximation of our Bayesian approach with fast training times, due to our data independent prior allowing for construction of an ensemble before data is observed.
- Comparatively few hyperparameters, allowing for effective use with little user input.

The rest of this Chapter proceeds as follows: we first address relevant literature in Section 6.1, before proceeding to detail the construction of the ensemble and the update method in Section 6.2. We then discuss the number of parameters involved and ease of use in Section 6.3. Section 6.4 then introduces the data and methodology used for evaluation, before we present our findings in Section 6.5. Finally, we summarise in Section 6.6.

6.1 Related Work

Similar to many other works for regression streaming data, the Approximately Bayesian Online Randomized Forest (ABORF) algorithm uses an ensemble of regression trees. However, unlike traditional approaches it does not incrementally grow and update the trees, instead opting to pregrow a set of regression trees before any data is observed, resulting in each tree effectively providing a fixed partition of the data. Learning continually occurs in linear models in the leaves, and the weight of each tree is then updated as data is observed, allowing the ensemble to adapt to changes in the data stream.

The use of a pre-grown ensemble of trees was suggested by Quadrianto and Ghahramani [2015], who use a Bayesian framework in which a data independent prior is used to generate a large ensemble of decision trees for classification. After observations have been made, the weight of each tree is updated. While this approach is naturally able to handle data arriving in an online fashion, the large ensemble size (1000 trees) and slow Bayesian update procedure makes it unsuitable for most streaming data. Furthermore, the original approach is unable to adapt the individual trees in the presence of concept drift, leaving them susceptible to change. However, the data independent prior allows for a rapid and lightweight construction on trees, with very few parameters necessary, and we adopt this approach in the ABORF algorithm.

Instead of a Bayesian update mechanism, fast updating ensemble approaches have been previously suggested for use with classification problems on streaming data. As described in Chapter 3, one of the first from Wang et al. [2003] introduces the Accuracy Weighted Ensemble (AWE) ensemble, which weights member predictions based on their past accuracy. It divides the incoming data stream into chunks, training a new classifier on each incoming chunk. Classifiers are then evaluated based on their performance on the current chunk, with the normalised performance being used as their weighting for the next chunk. The ensemble members are then updated based on the weight, with only the best performing classifiers being used as ensemble members for predictions each chunk. More recently, error based ensemble weighting approaches have been suggested for regression. Soares and Araújo [2015] introduce the On-line Weighted Ensemble (OWE), which weights ensemble members based on the logarithm of the inverse of the smoothed error of each member. When evaluating this approach, the authors also apply the Learn^{++}NSE algorithm [Elwell and Polikar, 2011] described in Chapter 3 to a regression scenario. In the ABORF algorithm, we use an evidence-based method for weighting ensemble members in the regression space, with the functional form being akin to weighting using inverse errors raised to an appropriate power.

One option in the ABORF ensemble is the use of oblique splits. Dasgupta and Freund [2008] create Random Projection (RP) trees which, unlike traditional trees, form oblique splits along a random projection in the feature space, as opposed to axis-aligned splits. Random projections are typically used to obtain a low dimensional embedding of the data, and have been previously been used in this manner in conjunction with Hoeffding Trees for online classification [Pham et al., 2017]. However, our approach uses random projections to ensure there is no alignment dependence of the ensemble members, allowing us to capture oblique feature dependence with shallower trees.

Overall, the ABORF algorithm has a number of advantages over traditional approaches. By using pre-grown trees we are able to easily reduce the data dependence of the ensemble members, meaning there is: less parameter dependence, faster training times, and no need to update members of the ensemble. Furthermore, creating random projections to split on, rather than axis-aligned splits, allows us to capture some of the benefits of multivariate decision trees [Brodley and Utgoff, 1995], but without the traditional overhead of taking time to choose optimal splits. Finally, the Bayesian approach allows us to obtain evidence based weights for ensemble members, as opposed to taking a more arbitrary approach.

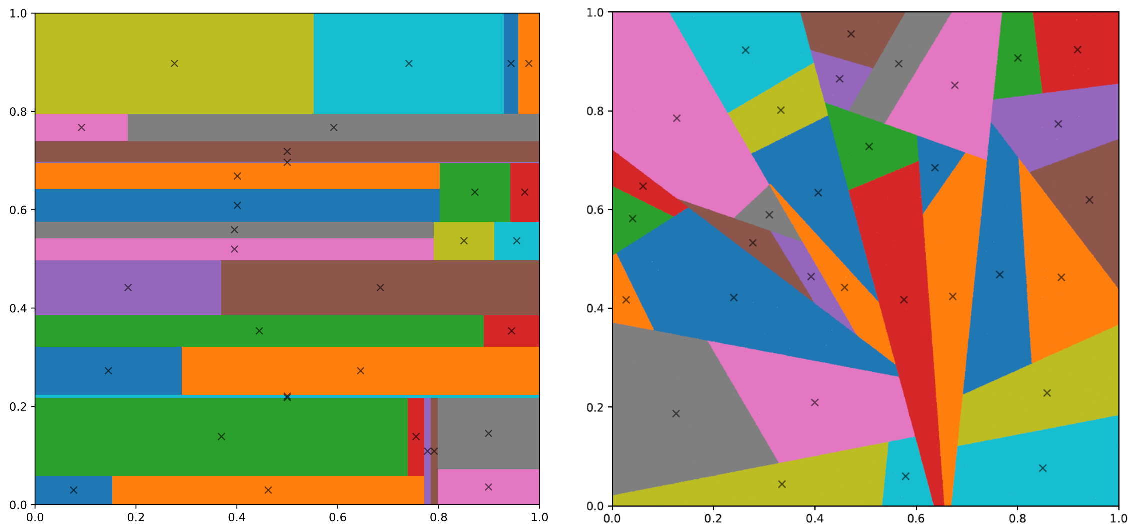
6.2 A Concept-Adapting Approximately Bayesian Regression Forest

Here we describe the model for a Concept-Adapting Approximately Bayesian Regression Forest. We first outline a fully Bayesian approach for the model, before approximating it to our final ABORF approach. While aspects of the fully Bayesian model are similar to the approach described by Quadrianto and Ghahramani [2015], we outline the model in full for clarity.

6.2.1 The Bayesian Model

We begin by assuming a stream of input features $(\mathbf{x}_1, ..., \mathbf{x}_T)$ and matching targets $(y_1, ..., y_T)$, where each pair $(\mathbf{x}_t, y_t)$ arrives sequentially. We assume the input features to be D dimensional and all data to be real, such that $\mathbf{x}_n = (x_{t,1}, ..., x_{t,D}) \in \mathbb{R}^D$ and $y_t \in \mathbb{R}$. The model trains an ensemble of decision trees with the goal of learning a mapping $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Y} = \mathbb{R}$. Each tree contains linear models in the leaves and is generated using randomised split points and split features. As a result, the individual trees in the ensemble can be viewed as a method of partitioning the feature space into randomised blocks, each of which contains a local linear model.

The construction of the model consists of two main stages. Firstly, an ensemble of regression trees is grown before any of the examples are observed, and untrained linear models are created in each of the leaves. The model then updates as each new example is



(a) UNIVARIATE SPLITS (b) MULTIVARIATE SPLITS

Figure 6.1. Example partition of a 2d unit hypercube from a) an axis-aligned tree and b) a random projection tree. Both trees are of depth 5. Leaf node centres are marked with an “×”.

observed, incrementally updating both the tree weights and the linear models.

The Tree Prior

We now describe how to generate the ensemble members using a data independent prior for each tree. The trees are balanced and of depth Q, with randomised splitting hyperplanes at each node. An example of the multivariate tree partition we employ can be seen in comparison to the partition obtained by an axis-aligned (also known as univariate), tree in Figure 6.1.

Initially, before any data is observed, K balanced binary trees of depth Q are created. At each split node, the splitting condition is provided by a $D-1$ dimensional hyperplane. The partitioning hyperplanes for each tree are then generated randomly. At each split node ν in the tree, the centre of the parent partition element covered by ν, $\mathbf{c}_\nu$, is found, and a vector of cut points $\boldsymbol{\tau}_\nu$ is chosen for each feature d:

$$\mathbf{c}_\nu = (c_{\nu,1}, ..., c_{\nu,D}) \tag{6.1}$$

$$\boldsymbol{\tau}_\nu = (\tau_{\nu,1}, ..., \tau_{\nu,D}), \qquad \tau_{\nu,d} \sim U[-1,1]. \tag{6.2}$$

The cut points are then normalised such that

$$\bar{\boldsymbol{\tau}}_\nu = \frac{\boldsymbol{\tau}_\nu}{||\boldsymbol{\tau}_\nu||}. \tag{6.3}$$

The cut points and the centre of the parent node are then used to assign incoming examples to either the left child node if

$$\mathbf{x}_n \cdot \bar{\boldsymbol{\tau}}_\nu \leq \mathbf{c}_\nu \cdot \bar{\boldsymbol{\tau}}_\nu \tag{6.4}$$

or the right child node if

$$\mathbf{x}_n \cdot \bar{\boldsymbol{\tau}}_\nu > \mathbf{c}_\nu \cdot \bar{\boldsymbol{\tau}}_\nu. \tag{6.5}$$

When creating the partitions, we want to be able to obtain the centre of each partition element before seeing any data. To achieve this, we bound the space by assuming all observations lie in the range $[0, 1]$, enabling us to estimate the node centres, $\mathbf{c}_\nu$. This is done by creating a number of points, each drawn randomly from a uniform distribution $U[0, 1]^D$. The points are then passed down the tree and used to estimate $\mathbf{c}_\nu$ to be the centre of mass of the points falling in node ν, effectively performing a Monte Carlo estimation of the integral that gives the centre of mass of the node.

Using $\mathbf{c}_\nu$, we can then generate the splitting hyperplane and pass the points to their respective child nodes, before repeating this process until the trees are generated. While there are a multiple ways to estimate the centre of the points (for example, the mean or median of the points falling within a node), in practice, we find $3 \times 2^{D-1}$ points to be sufficient for using mean centres.

To ensure the assumption that all values lie in the range $[0, 1]$ is valid, when we finally observe the data, we apply min-max feature scaling to values of $\mathbf{x}_d$. By tracking $x_{max,d}$ and $x_{min,d}$, the maximum and minimum observations for feature d up to time t, we normalise each feature observation according to

$$\frac{x_{t,d} - x_{min,d}}{x_{max,d} - x_{min,d}}. \tag{6.6}$$

It is worth noting that both $x_{min,d}$ and $x_{max,d}$ are dependant on the currently observed data and will, consequently, change over time. This may become problematic in the case of concept drift since if the underlying distribution for a feature drifts away from its original range, $x_{max,d}$ and $x_{min,d}$ will span a larger range relative to the current spread of observed feature values, thus causing incoming examples to become clustered into subrange of the original $[0, 1]$. While we do not empirically find this to be prohibitive, this failing suggests that alternative normalization approaches which are more robust to concept drift may be necessary.

Finally, we note that in the Bayesian model we have made the assumption that all of the features are real. However, many datasets also include nominal features, and we include the potential to assign a binary splits based on a nominal feature in the ABORF approximation by randomly assigning categories to each side of the split when they are first observed.

Although this ensemble generation process is similar to that of Quadrianto and Ghahramani [2015], we have made a couple of key changes. Unlike the univariate splits chosen by Quadrianto and Ghahramani, we have opted for multivariate splits to easily capture effects based on feature interactions. Additionally, we have elected to use balanced trees of depth Q in the ensemble, unlike the probability based tree structure of Quadrianto and Ghahramani, in which imbalanced trees are grown with each node having a fixed chance of being a split node. We choose balanced trees since up to the maximum depth, Q, they are able to capture any possible splits that would occur in imbalanced trees. Furthermore, this approach leads to more uniform distribution of the areas of each leaf, meaning each leaf will see similar amounts of data. This is advantageous since, as we will see in Section 6.2.1, the evidence of the tree is a multiplicative contribution of the evidence of each linear model, meaning that a if a few linear models see very little data, they have a high dependence on the prior which can have a magnified impact of the weight of the tree in the ensemble.

Tree Likelihood

Unlike the categorization approach taken in Quadrianto and Ghahramani [2015], we work in a regression situation, and thus arrive at a different form for the tree likelihood. In our case, to obtain the tree likelihood we begin by considering a single arbitrary leaf node in tree k, and $\mathbf{X}_{\nu,n}$ and $\mathbf{y}_{\nu,n}$ the set of examples reaching leaf node ν after n examples have been observed in the leaf. These examples are used to train a Bayesian linear regression model at each leaf, with the mean of the conditional distribution for $y_{\nu,n}$ modelled as

$$y_{\nu,n} = \mathbf{x}_{\nu,n}^T \hat{\boldsymbol{\beta}}_{\nu,n} + \epsilon_{\nu,n}, \tag{6.7}$$

where $\hat{\boldsymbol{\beta}}_{\nu,n}$ is a D dimensional weight vector, and $\epsilon_{\nu,n} \sim N(0, \hat{\sigma}^2_{\nu,n})$ is an $i.i.d.$ random variable. For simplicity, in the following we drop the dependence on ν and n in the notation. Then, the examples arriving at the leaf have the min-max feature scaling removed and are instead standardised according to

$$\frac{x_d - \mu_{\mathbf{x}_d}}{\sigma_{\mathbf{x}_d}}, \tag{6.8}$$

where $\mu_{\mathbf{x}_d}$ and $\sigma_{\mathbf{x}_d}$ are the mean and standard deviation of $\mathbf{x}_d$ respectively.

We now note that for a variable $\sigma^2 \sim \mathcal{IG}(a, b)$, the form of an inverse-gamma distribution is

$$\mathcal{IG}(a, b) \propto (\sigma^2)^{-a-1} \exp\{\tfrac{-b}{\sigma^2}\}, \tag{6.9}$$

with the mean $\mu_{\mathcal{IG}}$ and variance $V_{\mathcal{IG}}$ given by

$$\mu_{\mathcal{IG}} = \frac{b}{a-1} \quad \text{for} \quad a_n > 1 \tag{6.10}$$

$$V_{\mathcal{IG}} = \frac{b^2}{(a-1)^2(a-2)} \quad \text{for} \quad a > 2. \tag{6.11}$$

By choosing a normal-inverse-gamma conjugate prior, we end up with an analytical posterior for the coefficients (see, for example, [Bernardo and Smith, 2000, Appendix A]):

$$\begin{aligned} p(\boldsymbol{\beta}, \sigma^2 \,|\, \mathbf{y}, \mathbf{X}) &\propto p(\boldsymbol{\beta} \,|\, \sigma^2, \mathbf{y}, \mathbf{X}) p(\sigma^2 \,|\, \mathbf{y}, \mathbf{X}) && (6.12)\\ &= \mathcal{N}(\hat{\boldsymbol{\beta}}, \hat{\sigma}^2 \boldsymbol{\Lambda}^{-1}) \mathcal{IG}(a, b) && (6.13) \end{aligned}$$

where we have defined (with a subscript 0 denoting priors):

$$\begin{aligned} \boldsymbol{\Lambda} &= \mathbf{X}^T\mathbf{X} + \boldsymbol{\Lambda}_0 && (6.14)\\ \hat{\boldsymbol{\beta}} &= \boldsymbol{\Lambda}^{-1}(\boldsymbol{\Lambda}_0 \hat{\boldsymbol{\beta}}_0 + \mathbf{X}^T\mathbf{y}) && (6.15)\\ a &= a_0 + \frac{n}{2} && (6.16)\\ b &= b_0 + \frac{1}{2}(\mathbf{y}^T\mathbf{y} + \hat{\boldsymbol{\beta}}_0^T \boldsymbol{\Lambda}_0 \hat{\boldsymbol{\beta}}_0 - \hat{\boldsymbol{\beta}}^T \boldsymbol{\Lambda} \hat{\boldsymbol{\beta}}). && (6.17) \end{aligned}$$

The marginal likelihood, or evidence, for the leaf model, is now given by

$$p(\mathbf{y} \,|\, \mathbf{X}) = \int p(\mathbf{y} \,|\, \mathbf{X}, \boldsymbol{\beta}, \sigma) p(\boldsymbol{\beta}, \sigma) \, d\boldsymbol{\beta} \, d\sigma. \tag{6.18}$$

This integral can be performed analytically (for example [Bishop, 2006, Exercise 3.23]), and leads to

$$p(\mathbf{y} \,|\, \mathbf{X}) = \frac{1}{(2\pi)^{n/2}} \sqrt{\frac{|\boldsymbol{\Lambda}_0|}{|\boldsymbol{\Lambda}|}} \cdot \frac{b_0^{a_0}}{b^a} \cdot \frac{\Gamma(a)}{\Gamma(a_0)}. \tag{6.19}$$

Now we have the evidence for the model in each leaf, we can find the evidence for the entire tree, $\mathcal{T}$. Given the partitioning model for the tree, we assume that the targets $\mathbf{y}$ are independent across leaf nodes, and $i.i.d.$ within each leaf. Reintroducing the subscript ν, we are now able to obtain the tree likelihood:

$$\begin{aligned} p(\mathbf{y}_t \,|\, \mathbf{X}_t, \mathcal{T}) &= \int \prod_{\nu \in \Omega_{\mathcal{T}}} p(\mathbf{y}_\nu \,|\, \mathbf{X}_\nu, \boldsymbol{\beta}_\nu, \sigma_\nu) p(\boldsymbol{\beta}_\nu, \sigma_\nu) \, d\boldsymbol{\beta}_\nu, d\sigma_\nu && (6.20)\\ &= \prod_{\nu \in \Omega_{\mathcal{T}}} \frac{1}{(2\pi)^{n/2}} \sqrt{\frac{|\boldsymbol{\Lambda}_{\nu,0}|}{|\boldsymbol{\Lambda}_\nu|}} \cdot \frac{b_{\nu,0}^{a_{\nu,0}}}{b_\nu^{a_\nu}} \cdot \frac{\Gamma(a_\nu)}{\Gamma(a_{\nu,0})} && (6.21) \end{aligned}$$

where $\mathbf{X}_t$ and $\mathbf{y}_t$ denote the set of examples seen by the tree and $\Omega_{\mathcal{T}}$ represents the set of all leaf nodes in the tree. As we typically don't expect prior information to be available in a streaming scenario, a simple and uninformative choice of prior is $\mathbf{\Lambda}_0 = c\mathbf{I}$, $\hat{\boldsymbol{\beta}}_0 = \mathbf{0}$, and $a_0 = b_0 = 0.001$. This form of the prior effectively leads to a Bayesian ridge regression, with a ridge penalty of magnitude c on the weights.

Tree Posterior

Using Bayes rule, we can find the tree posterior to be

$$p(\mathcal{T}|\mathbf{X},\mathbf{y}) \propto p(\mathbf{y}|\mathbf{X},\mathcal{T}) \times p(\mathcal{T}). \tag{6.22}$$

While we have previously obtained $p(\mathbf{y}|\mathbf{X},\mathcal{T})$ and $p(\mathcal{T})$, it is clear that, like other Bayesian tree models, the posterior calculation is not possible analytically. Instead, estimating the posterior requires the use of other approaches, which have traditionally focussed on Markov Chain Monte Carlo based methods. While Metropolis-Hastings methods are a popular choice [Chipman et al., 1998, 2006; Schetinin et al., 2007], other approaches are also possible, such as that of Lakshminarayanan et al. [2013], who follow a Sequential Monte Carlo approach. We follow Quadrianto and Ghahramani [2015] in using importance sampling, a fast and simple method of estimating the posterior.

The predictive distribution

As suggested by Quadrianto and Ghahramani [2015], we use importance weights to obtain the predictive distribution of the ensemble. For a previously unseen test point $\mathbf{x}'$, the predictive distribution of the ensemble is given by:

$$\begin{aligned}
p(y' \mid \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) &= \int p(y' \mid \mathcal{T}, \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) p(\mathcal{T} \mid \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) d\mathcal{T} && (6.23)\\
&= \int p(y' \mid \mathcal{T}, \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) \frac{p(\mathcal{T} \mid \mathbf{X}_t, \mathbf{y}_t)}{p(\mathcal{T})} p(\mathcal{T}) d\mathcal{T} && (6.24)\\
&\approx \sum_k p(y' \mid \mathcal{T}_k, \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) \frac{p(\mathcal{T}_k \mid \mathbf{X}_t, \mathbf{y}_t)}{p(\mathcal{T}_k)} && (6.25)\\
&\approx \sum_k p(y' \mid \mathcal{T}_k, \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) p(\mathbf{y}_t \mid \mathbf{X}_t, \mathcal{T}_k) && (6.26)
\end{aligned}$$

where the importance weight $w_k = p(\mathbf{y}_t \mid \mathbf{X}_t, \mathcal{T}_k)$, and $p(y' \mid \mathcal{T}_k, \mathbf{x}', \mathbf{X}_t, \mathbf{y}_t) =$
$p(y' \mid \mathcal{T}_k, \mathbf{x}', \mathbf{X}_n, \mathbf{y}_n) = \mathcal{N}(\mathbf{X}_n^T \hat{\boldsymbol{\beta}}_n, \hat{\sigma}^2)$. We note that $\mathbf{X}_n$ and $\hat{\boldsymbol{\beta}}_n$ correspond to the particular node in each tree, $\mathcal{T}_k$, into which $\mathbf{x}'$ falls.

While, as noted by Quadrianto and Ghahramani [2015], a linear combination of these weights will lead to Bayesian model averaging, we follow their choice of using a power likelihood to prevent all the mass from being placed in a single decision tree. However, unlike Quadrianto and Ghahramani [2015] who study a range of power likelihoods, we instead opt to use $1/|\Omega_{\mathcal{T}}|$, effectively treating the likelihood of each tree as the geometric mean of the likelihood of the leaves. We are then able to obtain a point estimate for the ensemble prediction $\hat{y}'$ according to

$$\hat{y}' = \frac{1}{Z} \sum_{k=1}^{K} (w_k)^{\frac{1}{|\Omega_{\mathcal{T}}|}} (\mathbf{x}')^T \hat{\boldsymbol{\beta}}_n \ , \tag{6.27}$$

where $Z = \sum_{k=1}^{K} (w_k)^{\frac{1}{|\Omega_{\mathcal{T}}|}}$.

Overall, we now have a fully Bayesian model which can be incrementally trained and used to make predictions in a stationary regression context. As each example is seen, it falls into exactly one leaf in each tree, which then makes a prediction. The evidence for the predicting leaf is updated, and used to update the tree weight. Finally, the predictions from each tree are combined into the ensemble prediction using the corresponding tree weights. This methodology results in every prediction by the model being a weighted ensemble of local Bayesian linear model predictions, where each linear model is trained on slightly different data depending on the layout of the partitions, and where the weight of each prediction is dependant on the evidence for the tree as a whole.

6.2.2 Adapting to Concept Drift

We have now established an approach to train a fully Bayesian regression forest designed for use in a stationary scenario. However, in a streaming scenario we cannot assume that the mapping we are learning is stationary, and instead expect it to be dependant on time. This results in a time-dependant mapping of the form $\mathcal{F}(t) : \mathcal{X} \rightarrow \mathcal{Y}$. Practically this means that to account for drifts in the data stream, we want to decrease our prior belief in older data as new examples arrive. We now adapt our model to incorporate this belief.

As examples in the data stream arrive, they reach each leaf as row vectors, $\mathbf{x}_n$. Under the stationary approach we have previously described, this results in updates to the parameters of the Bayesian linear regression models in the leaves given by:

$$\boldsymbol{\Lambda}_n = \mathbf{x}_n^T \mathbf{x}_n + \boldsymbol{\Lambda}_{n-1} \tag{6.28}$$

$$\hat{\boldsymbol{\beta}}_n = \boldsymbol{\Lambda}_n^{-1} (\boldsymbol{\Lambda}_{n-1} \hat{\boldsymbol{\beta}}_{n-1} + \mathbf{x}_n^T y_n) \tag{6.29}$$

$$a_n = a_{n-1} + \frac{1}{2} \tag{6.30}$$

$$b_n = b_{n-1} + \frac{1}{2} (\mathbf{y}_n^T \mathbf{y}_n + \hat{\boldsymbol{\beta}}_{n-1}^T \boldsymbol{\Lambda}_{n-1} \hat{\boldsymbol{\beta}}_{n-1} - \hat{\boldsymbol{\beta}}_n^T \boldsymbol{\Lambda}_n \hat{\boldsymbol{\beta}}_n) \tag{6.31}$$

where we have again left out the ν subscripts for legibility.

To account for concept drift, we now introduce a decay term, $\gamma < 1$ which we use to downweight the contribution of older data, resulting in precision and weight updates of the form:

$$\mathbf{\Lambda}_n = \mathbf{x}_n^T \mathbf{x}_n + \gamma \mathbf{\Lambda}_{n-1} + (1-\gamma)\mathbf{\Lambda}_0 \tag{6.32}$$

$$\hat{\boldsymbol{\beta}}_n = \mathbf{\Lambda}_n^{-1}(\gamma \mathbf{\Lambda}_{n-1} \hat{\boldsymbol{\beta}}_{n-1} + \mathbf{x}_n^T y_n) \tag{6.33}$$

and corrections to the inverse gamma parameters in the form of

$$a'_n = 2 + \gamma(a_n - 2) \tag{6.34}$$

$$b'_n = \frac{(a'_n - 1)}{(a_n - 1)} b_n. \tag{6.35}$$

The precision and weight updates effectively mean each previously seen point i has a time-dependant contribution. For the precision matrix, points have a contribution of $\gamma^i \mathbf{x}_{n-i}^T \mathbf{x}_{n-i}$, while the magnitude of the original prior $\mathbf{\Lambda}_0$ is constant. Similarly, each point also has a contribution to $\mathbf{\Lambda}_n \hat{\boldsymbol{\beta}}_n$ of $\gamma^i \mathbf{x}_{n-i}^T y_{n-i}$.

On the other hand, the corrections for a_0 and b_0 are designed to inflate the variance of the inverse-gamma distribution whilst preserving the same mean. Therefore, to inflate the variance by a factor $\frac{1}{\gamma}$ while preserving the mean, we can see from Equations 6.10 and 6.11 that we require

$$\frac{b'_n}{a'_n - 1} = \frac{b_n}{a_n - 1} \tag{6.36}$$

and

$$\frac{b'^2_n}{(a'_n - 1)^2 (a'_n - 2)} = \frac{b_n^2}{\gamma (a_n - 1)^2 (a_n - 2)}. \tag{6.37}$$

Solving these for a' and substituting back in for b' leads to the corrections given in Equations 6.34 and 6.35.

This approach allows us to adapt to concept drift within the linear model at each leaf. However, this approach weights every leaf equally, and it is unclear whether it is entirely correct, or whether leaves which have seen data more recently should have a higher weight. Consequently, we introduce a second, tree-level decay, $\gamma_{\mathcal{T}}$ which decays the evidence of the entire tree as each new example arrives according to

$$p(\mathbf{y}_n \,|\, \mathbf{X}_n, \mathcal{T}) = \frac{\gamma_{\mathcal{T}}}{1+\gamma_{\mathcal{T}}} \, p(\mathbf{y}_{n-1} \,|\, \mathbf{X}_{n-1}, \mathcal{T}) + \frac{1}{1+\gamma_{\mathcal{T}}} p(\mathbf{y}_n \,|\, \mathbf{X}_n, \mathcal{T}). \tag{6.38}$$

6.2.3 Approximating the Model

The model we have derived so far is theoretically robust and suited to adapt in the presence of concept drift. However, in practice aspects such as $\hat{\boldsymbol{\beta}}_n$ are very expensive to compute,

Algorithm 6.1 The `ABORF` algorithm

```
1: input: $D, \mathbf{f}, Q, K$    ▷ Number of features, feature types, tree depth, ensemble size
2: $\mathbf{w} := (1/K, 1/K, ..., 1/K)$    ▷ Ensemble members have equal weight initially
3: $\mathcal{E} :=$ createEnsemble$(\mathbf{f}, Q, K)$    ▷ Create the ensemble
4: for $\mathbf{x}_t, y_t$ in stream do
5:     $\mathbf{x}_t, \mathbf{x}_{min}, \mathbf{x}_{max} :=$ scale$(\mathbf{x}_t, \mathbf{x}_{min}, \mathbf{x}_{max})$    ▷ Scale the features to [0,1]
6:     $\hat{\mathbf{y}}_t :=$ predict$(\mathcal{E}, \mathbf{x}_t)$    ▷ Make a prediction for each tree
7:     $\hat{Y}_t = \mathbf{w} \cdot \hat{\mathbf{y}}_t / ||\mathbf{w}||$    ▷ Get the ensemble prediction
8:     yield: $\hat{Y}_t$    ▷ Return the prediction
9:     for $\mathcal{T}$ in $\mathcal{E}$ do
10:        $\mathcal{T} :=$ updateLeaf$(\mathcal{T}, \hat{y}_t, \mathbf{x}_t, y_t)$    ▷ Update linear model in leaf
11:        $w_{\mathcal{T}}, \mathcal{T} :=$ updateWeight$(D, w_{\mathcal{T}}, \mathcal{T}, \hat{y}_t, y_t)$    ▷ Update member weight
12:    end for
13: end for
```

since the $\mathbf{\Lambda}_n^{-1}$ term in Equation 6.29 requires a full matrix inversion after each additional observation, which is at best considered to be an $O(D^{2.376})$ [Coppersmith and Winograd, 1990]. Furthermore, the determinant $|\mathbf{\Lambda}_n|$ is also necessary for calculating the evidence in Equation 6.21, which has the same $O(D^{2.376})$ complexity at best. While it is possible to update a matrix inverse and determinant through the use of $O(D^2)$ rank-one updates [Krause and Igel, 2015] in some circumstances, we note that in this case it also effectively results in a different choice of prior, $\mathbf{\Lambda}_0$, after each update, making it unsuitable for our model.

As a result, updating each tree in the fully Bayesian model after the observation of an additional example has at best a time complexity of $O(D^{2.376})$. Since this is considered too slow for use in many streaming circumstances, we now approximate this to a model with an $O(D)$ update time complexity for each tree, with the resulting ABORF algorithm shown in Algorithm 6.1.

With the primary goal of providing accurate predictions $\hat{y}'$, we thus need to approximate the current terms in $\hat{y}' = \frac{1}{Z}\sum_k (w_k)^{\frac{1}{|\Omega_{\mathcal{T}}|}} (\mathbf{x}')^T \hat{\boldsymbol{\beta}}_n$. We begin be finding an approximation for

$$\hat{\boldsymbol{\beta}}_n = \mathbf{\Lambda}_n^{-1}(\mathbf{\Lambda}_{n-1}\hat{\boldsymbol{\beta}}_{n-1} + \mathbf{x}_n^T y_n), \tag{6.39}$$

the slowest step of the algorithm. Furthermore, we will then see that this necessitates a new expression for the evidence used in the weight term,

$$w_k = p(\mathbf{y}_n \,|\, \mathbf{X}_n, \mathcal{T}_k), \tag{6.40}$$

in which we will find a faster approximation for $|\mathbf{\Lambda}_n|$.

Approximating $\hat{\boldsymbol{\beta}}_n$

We begin by approximating $\hat{\boldsymbol{\beta}}_n$. By recursively applying Equations 6.32 and 6.33 we can write the terms for $\boldsymbol{\Lambda}_n$ and $\boldsymbol{\Lambda}_n\hat{\boldsymbol{\beta}}_n$ as summations:

$$\boldsymbol{\Lambda}_n = \boldsymbol{\Lambda}_0 + \sum_{i=1}^{n} \gamma^{n-i}\mathbf{x}_n^T\mathbf{x}_n = \boldsymbol{\Lambda}_0 + \mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{X}_n \tag{6.41}$$

$$\boldsymbol{\Lambda}_n\hat{\boldsymbol{\beta}}_n = \boldsymbol{\Lambda}_0\hat{\boldsymbol{\beta}}_0 + \sum_{i=1}^{n} \gamma^{n-i}\mathbf{x}_n^T y_n = \boldsymbol{\Lambda}_0\hat{\boldsymbol{\beta}}_0 + \mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{y}_n \tag{6.42}$$

where $\boldsymbol{\Gamma}$ is a diagonal matrix with elements $\boldsymbol{\Gamma}_{i,i} = \gamma^{n-i}$. Under a prior of the form $\boldsymbol{\Lambda}_0 = c\mathbf{I}$ and $\hat{\boldsymbol{\beta}}_0 = \mathbf{0}$, we can then write:

$$\boldsymbol{\Lambda}_n = c\mathbf{I} + \mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{X}_n \tag{6.43}$$

$$\hat{\boldsymbol{\beta}}_n = \boldsymbol{\Lambda}_n^{-1}(\mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{y}_n). \tag{6.44}$$

Here, $\boldsymbol{\Gamma}$ is akin to a weight matrix, and we end up with the form of a weighted ridge regression estimate for $\hat{\boldsymbol{\beta}}_n$:

$$\hat{\boldsymbol{\beta}}_n = (\mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{X}_n + c\mathbf{I})^{-1}(\mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{y}_n). \tag{6.45}$$

We now approximately solve for $\hat{\boldsymbol{\beta}}_n$ using a variation of stochastic gradient descent (SGD) known as RMSProp SGD [Hinton et al., 2012]. As detailed in Section 4.2.1, RMSProp SGD has a similar learning rule to regular SGD, but adds a gradient based regularisation term to the learning rate, allowing the learning rate to adapt efficiently to the data. As each example arrives, we incrementally update the linear model in the leaf, as can be seen in Algorithm 6.2. We are able to account for the decay term by simply scaling the initial weight by γ before each update, effectively rescaling every previously taken step by γ. Furthermore, in the case that information about nominal features is available, they are then one-hot encoded at the leaf, with each binary category then treated as an additional numerical feature when learning the linear model.

Approximating w_k

Approximating the tree likelihood is more challenging. As we have chosen to use a non-Bayesian linear model to obtain $\hat{\boldsymbol{\beta}}_n$, we no longer incorporate prior information into our model, meaning the expression we have previously obtained for $p(\mathbf{y}_n \mid \mathbf{X}_n, \mathcal{T}_k)$ in Equation 6.21 is no longer applicable. Consequently, we now obtain the weight in Equation 6.40 using a different form of the evidence expression; the general case in which no form for the prior is specified.

The empirical Bayes, or evidence approximation (see [Bishop, 2006, Section 3.5]), assumes

Algorithm 6.2 The `updateLeaf` function

```
1: input: $\mathcal{T}, \hat{y}, \mathbf{x}, y$    ▷ tree, tree prediction, features, target
2: $\epsilon = 10^{-8}$    ▷ Ensure no divide by 0 issues
3: $\eta, \gamma :=$ getParameters($\mathcal{T}$)    ▷ User specified learning rate and decay coefficient
4: $\hat{\boldsymbol{\beta}}, \mathbf{g}, \mathbb{E}[\mathbf{g}^2] :=$ getModelState($\mathcal{T}$)    ▷ Get stored model stats
5: for $x_j$ in $\mathbf{x}$ do    ▷ Loop over each feature
6:     $g_j := (\hat{y} - y)x_j$
7:     $\mathbb{E}[g_j^2] := \gamma\mathbb{E}[g_j^2] + (1-\gamma)g_j^2$
8:     $\hat{\beta}_j := \gamma\hat{\beta}_j - \frac{\eta}{\sqrt{\mathbb{E}[g_j^2]+\epsilon}}g_j$    ▷ Update the weights
9: end for
10: $\mathcal{T} :=$ storeModelState($\mathcal{T}, \hat{\boldsymbol{\beta}}, \mathbf{g}, \mathbb{E}[\mathbf{g}^2]$)    ▷ Store the model stats
11: yield: $\mathcal{T}$    ▷ Return the tree with the updated model
```

that the evidence is centred around the most probable values of the weights and that the evidence is approximately normally distributed. Under these assumptions, we can marginalise over the weights to approximate the evidence of a Bayesian linear regression model in node ν (where we have again dropped the subscript ν for legibility) with an error term of the form

$$\begin{aligned} E(\hat{\boldsymbol{\beta}}_n) &= \rho_n E_{residual}(\hat{\boldsymbol{\beta}}_n) + \alpha_n E_{ridge}(\hat{\boldsymbol{\beta}}_n) && (6.46)\\ &= \frac{\rho_n}{2}(\mathbf{y}_n - \mathbf{X}_n\hat{\boldsymbol{\beta}}_n)^T\boldsymbol{\Gamma}_n(\mathbf{y}_n - \mathbf{X}_n\hat{\boldsymbol{\beta}}_n) + \frac{\alpha_n}{2}\hat{\boldsymbol{\beta}}_n^T\hat{\boldsymbol{\beta}}_n && (6.47)\end{aligned}$$

to be

$$p(\mathbf{y}_n \,|\, \mathbf{X}_n, \hat{\boldsymbol{\beta}}_n, \rho_n, \alpha_n) = \left(\frac{\rho_n}{2\pi}\right)^{n/2} (\alpha_n)^{D/2} \left|\rho_n\boldsymbol{\Lambda}_n\right|^{-1/2} \exp\{-E(\hat{\boldsymbol{\beta}}_n)\} \qquad (6.48)$$

where ρ and α are unknown parameters to be optimised, which relate to the precision of the target noise and the magnitude of the ridge penalty respectively, and $\boldsymbol{\Lambda}_n = \frac{\alpha_n}{\rho_n}\mathbf{I} + \mathbf{X}_n^T\boldsymbol{\Gamma}_n\mathbf{X}_n$.

Using this, we reintroduce the leaf node ν notation and, again making the assumption that the leaves are independent, multiply across the leaves for the tree likelihood:

$$p(\mathbf{y}_n \,|\, \mathbf{X}_n, \mathcal{T}_k) = \prod_{\nu\in\Omega_{\mathcal{T}}} \left(\frac{\rho_{n,\nu}}{2\pi}\right)^{n_\nu/2} (\alpha_{n,\nu})^{D/2} \left|\rho_n\boldsymbol{\Lambda}_{n,\nu}\right|^{-1/2} \exp\{-E(\hat{\boldsymbol{\beta}}_{n,\nu})\}. \qquad (6.49)$$

Recognising that $|\rho_n\boldsymbol{\Lambda}_{n,\nu}| = \rho_n^D|\boldsymbol{\Lambda}_{n,\nu}|$, and that we may cancel out terms which are constant across the tree weights, we can now write:

$$\begin{aligned} w_k &= \prod_{\nu\in\Omega_{\mathcal{T}}} (\rho_{n,\nu})^{(n_\nu-D)/2} (\alpha_{n,\nu})^{D/2} \left|\boldsymbol{\Lambda}_{n,\nu}\right|^{-1/2} \exp\{-E(\hat{\boldsymbol{\beta}}_{n,\nu})\} && (6.50)\\ &= \exp\{-\sum_{\nu\in\Omega_{\mathcal{T}}} E(\hat{\boldsymbol{\beta}}_{n,\nu})\} \prod_{\nu\in\Omega_{\mathcal{T}}} (\rho_{n,\nu})^{(n_\nu-D)/2} (\alpha_{n,\nu})^{D/2} \left|\boldsymbol{\Lambda}_{n,\nu}\right|^{-1/2}. && (6.51)\end{aligned}$$

We can further simplify this expression by noting that since we want to encourage a diverse ensemble, we are more interested in penalising the residual error, rather than the ridge error in the term $E(\hat{\boldsymbol{\beta}}_n) = \rho_n E_{residual}(\hat{\boldsymbol{\beta}}_n) + \alpha_n E_{ridge}(\hat{\boldsymbol{\beta}}_n)$. We can achieve this by fixing α_n to the same small constant value for each leaf, reducing the tree weights to:

$$w_k = \exp\{-\sum_{\nu\in\Omega_\mathcal{T}} \rho_n E_{residual}(\hat{\boldsymbol{\beta}}_{n,\nu})\} \prod_{\nu\in\Omega_\mathcal{T}} (\rho_{n,\nu})^{(n_\nu - D)/2} \, |\boldsymbol{\Lambda}_{n,\nu}|^{-1/2}. \tag{6.52}$$

The challenge in evaluating this expression for the weights in a streaming setting now lies in calculating $|\boldsymbol{\Lambda}_{n,\nu}|$, an $O(D^3)$ operation. We now note that since we have approximated α_n to the same small constant for each term, we can write $\boldsymbol{\Lambda}_{n,\nu} \approx \mathbf{X}_{n,\nu}^T \boldsymbol{\Gamma}_{n,\nu} \mathbf{X}_{n,\nu}$, a weighted covariance matrix. By assuming the input features are independent, $\boldsymbol{\Lambda}_{n,\nu}$ is a diagonal matrix with diagonal entries $n_\nu = \sum_{i=1}^n \gamma^i \approx \frac{1}{1-\gamma}$, assuming n is sufficiently large. This means $|\boldsymbol{\Lambda}_{n,\nu}| \approx \left(\frac{1}{1-\gamma}\right)^D$, which is independent of the leaf data and thus constant across trees, meaning the term may be dropped from the tree weights, leading to:

$$w_k = \exp\{-\sum_{\nu\in\Omega_\mathcal{T}} \rho_n E_{residual}(\hat{\boldsymbol{\beta}}_{n,\nu})\} \prod_{\nu\in\Omega_\mathcal{T}} (\rho_{n,\nu})^{\frac{1-D(1-\gamma)}{2(1-\gamma)}} . \tag{6.53}$$

Finally, we need to determine $\rho_{n,\nu}$. This parameter determines the relative strength of the residual error term in the error function, and is typically chosen to maximise the evidence of the model, which assuming we have sufficient examples is given by (see [Bishop, 2006, Section 3.5]):

$$\rho_{n,\nu} = \frac{n_\nu}{2E_{residual}(\hat{\boldsymbol{\beta}}_n)}. \tag{6.54}$$

However, as the linear models in this scenario are combined into a tree, it is unclear whether optimizing this for the linear model at each leaf is sufficient. Consequently, we propose three levels for setting $\rho_{n,\nu}$:

> Leaf - By optimizing using the residuals at each leaf, we maximize the evidence of each individual linear model. However, this will result in a non-uniform penalty function both within and across trees.
>
> Tree - Optimizing using the tree residuals results in a value for each tree $\rho_{n,\nu} = \rho_{n,k}$, effectively also enforcing a uniform penalty function with each tree. However, this will still result in a non-uniform penalty function across trees.
>
> Ensemble - This approach maximizes the evidence over the entire ensemble, resulting in a single value: $\rho_{n,\nu} = \rho_{n,k} = \rho_{n,ensemble}$, with a uniform penalty function applying both within each tree and across trees.

The full weight update process can be seen in Algorithm 6.3. The weight is updated using

Algorithm 6.3 The `updateWeight` function, based on a leaf level update for $\rho_{n,\nu}$

```
1: input: $w_{\mathcal{T}}, \mathcal{T}, \hat{y}_t, y_t$    ▷ tree weight, tree, tree prediction, target
2: $e_{t-1,\mathcal{T}}, b_{t-1,\mathcal{T}}, \sum e_{t-1}^2, n_{t-1} :=$ getStoredStatistics($\mathcal{T}$)
3:    ▷ Get stored statistics for tree and leaf
4: $e_t := y_t - \hat{y}_t$    ▷ Get Prediction Error
5: $n_t := \gamma n_{t-1} + 1$    ▷ Decay and update instances seen
6: $\sum e_t^2 := \gamma \sum e_{t-1}^2 + e_t^2$    ▷ Decay and update sum of squared errors
7: $\rho_t = \frac{n}{\sum e_t^2}$    ▷ Get $\rho_t$ from a leaf-level update
8: $e_{t,\mathcal{T}} := \gamma_{\mathcal{T}}\ (e_{t-1,\mathcal{T}}) + \frac{1}{2}\rho_t e_t^2$    ▷ Decay and update tree error
9: $b_{t,\mathcal{T}} := b_{t-1,\mathcal{T}} \left(\rho_t^{\frac{n_t - D}{2}}\right)\left(\rho_{t-1}^{\frac{D-n_{t-1}}{2}}\right)$    ▷ Update the scaling term
10: $w_{\mathcal{T}} = (\exp\{-e_{t,\mathcal{T}}\} b_{t,\mathcal{T}})^{1/|\Omega_{\mathcal{T}}|}$    ▷ Update the tree weight
11: $\mathcal{T} :=$ storeStatistics($e_{t,\mathcal{T}}, b_{t,\mathcal{T}}, \sum e_t^2, n_t$)    ▷ Store relevant statistics
12: yield: $w_{\mathcal{T}}, \mathcal{T}$    ▷ Return updated weight and tree with updated statistics
```

a leaf-level update, which can be seen since the contributions to ρ_t come from the error observed in the leaf (Algorithm 6.3 lines 6-7). The update occurs according to the approximation result in Equation 6.53, which we break up into two terms in Algorithm 6.3: the error term $e_{t,\mathcal{T}} = \sum_{\nu \in \Omega_{\mathcal{T}}} \rho_n E_{residual}(\hat{\boldsymbol{\beta}}_{n,\nu})$ and the scaling term, $b_{t,\mathcal{T}} = \prod_{\nu \in \Omega_{\mathcal{T}}} (\rho_{n,\nu})^{\frac{1-D(1-\gamma)}{2(1-\gamma)}}$.

In this section we have introduced a fully Bayesian model for making predictions with regression data streams, in which an ensemble of trees is pre-grown before data is observed. We have designed adaptions to this model to account for the effects of concept drift, before finally introducing ABORF: a lightweight approximation of the model with $O(D)$ update times for each ensemble member, which is better suited to the high frequency environment of many data streams. We now proceed to discuss the advantages of this model design in more detail, before providing an empirical evaluation in comparison to other streaming algorithms.

6.3 Hyperparameters

Due to the complex nature of reacting to unknown drifts, many modern data streaming techniques require a large number of user-specified parameters. While many of them may not have a large impact on performance, the uniqueness of each data stream means that even in cases when sensitivity to parametrization has been examined in the literature, it is hard to generalise to other data streams. Consequently, there is a high demand on the user, especially given there can be little time to examine parameter dependence when working with streaming data and that the impact of changing a parameter is often unintuitive.

The ABORF algorithm we have introduced uses fewer parameters than other contemporary methods, meaning it can be appropriately used in an off-the-shelf manner in many situations. A comparison of the number of parameters used is shown in Table 6.1. That relatively

	No. Hyperparameters
FIMT-DD	9
ARF-Reg	13
Bayesian	6
ABORF	5

Table 6.1. The number of hyperparameters used in different data streaming algorithms. "Bayesian" refers to the fully Bayesian model we have just described in Sections 6.2.1 and 6.2.2.

fewer parameters are used in the ABORF algorithm results from the choice to treat the tree structures of the ensemble members as a method of partitioning the data, rather than as a method of leaning the data structure. This in turn simplifies the algorithm in a number of ways relative to other contemporary approaches:

No learning splits - Typically, tree based online learning requires parameters to be set for the accuracy of splits, and the speed at which they are learnt. On the other hand, the pre-grown trees in ABORF only require a choice of the desired depth, Q.

No change detection - In ABORF, the constant update process is naturally able to adapt to changes in the data, and there is thus no need to explicitly detect changes, reducing the number of parameters relating to change detection.

No explicit change adaption - Many online tree-based learners update the structure of the trees in the presence of drift, to ensure they are able to constantly provide meaningful partitions of the data into homogeneous regions. However, in ABORF, the partitions do not need to be meaningful, and thus no adaption to the tree structure is necessary in the presence of change.

6.4 Evaluation and Parameter Selection

We evaluate the performance of the ABORF algorithm on a variety of different datasets, all of which we have previously described in Chapter 4. Along with the synthetic FRIEDMAN dataset with abrupt changes, and the synthetic HYPERPLANE dataset with incremental changes, we also use the real world datasets AIRLINE08, HOUSING, YEARMSD and ZURICH. As previously indicated in Section 4.4.1, the real world datasets can be found on the OpenML and UCI machine learning repository databases, while the FRIEDMAN dataset with abrupt changes suggested by Ikonomovska et al. [2011b] is created from 10 independently and uniformly distributed features in the range (0,1). 5 of the features are solely included as noise, while the other 5 are combined in a nonlinear manner and added to a noise term to create the target. As before, there are 1 million points in the dataset, with abrupt changes created at 3 different points: 25%, 50% and 75% of the way through. The changes are introduced by permuting the informative features in the formula used

Table 6.2. An overview of ABORF hyperparameters

Description	Symbol	Value
Tree depth	Q	10
Leaf learning rate	η	0.25
Leaf prior	c	≈ 0
Linear model smoothing factor	γ	0.96
Tree evidence smoothing factor	$\gamma_{\mathcal{T}}$	0.9996

to generate the target. The HYPERPLANE dataset is also created in the same manner as in Chapter 4, with the target representing the squared distance of a point denoted by 5 informative features from a rotating hyperplane. There are again 1 million points in the dataset, with incremental changes that take place over 10% of the examples, occurring between 25% and 35%, 50% and 60%, and 75% and 85% of the way through the examples.

Predictive performance is evaluated using the prequentially computed Mean Absolute Error (MAE) and CPU timings are also provided. Unless otherwise stated, we have use balanced trees of depth $Q = 10$, a learning rate and prior of $\eta = 0.25$ and $c \approx 0$ respectively for the linear models, and smoothing factors of $\gamma = 0.96$ and $\gamma_{\mathcal{T}} = 0.9996$ for smoothing at the leaf and tree levels respectively. A summary of these ABORF hyperparameters can be found in Table 6.2. While we use an ensemble of $K = 10$ trees for comparison with other approaches, we also examine the impact of larger ensemble sizes. All results given were obtained using an implementation in the MOA framework, on a laptop with Intel Core i5-6200U dual core 2.3GHz CPU, L2 cache: 513kb, L3 cache: 3Mb, 16Gb RAM @ 2133MHz.

The choice of $\gamma = 0.96$ naturally leads to rapid forgetting in the leaves of the tree, enabling them to react quickly to drifts while maintaining a sufficiently large effective sample size to make meaningful predictions. Choosing an overly large value for γ would lead to the leaves being unable to react to drifts, while an overly small value would lead to insufficient data to make effective predictions. This suggests that while the optimal value for γ will depend on the timescale of any changes in the data, a small value which maintains a sufficiently large effective sample size, as we have chosen, should be appropriate for any situation. Furthermore, this choice of $\gamma = 0.96$ should be suitable for all forms of drift, since it effectively leads to a smaller and more recent sample of data being used for the leaf models. However, the smaller effective sample size may also lead to poorer performance in stationary scenarios. In comparison to other approaches, the advantage of this choice of γ being appropriate for all forms of drift is most likely to show on gradual and incremental drifts rather than abrupt drifts. This is because the nature of the change detection in many contemporary algorithms means they are usually worse suited to gradual and incremental drifts.

Another aspect linked to the choice of γ is that the learning rate, $\eta = 0.25$, is very high compared to typical algorithms. This learning rate was determined on the basis of preliminary experiments and is to speed up learning and compensate for the short memory in leaves, since $\gamma = 0.96$ corresponds to an effective memory of $(1-\gamma)^{-1} = (1-0.96)^{-1} = 25$ examples.

Furthermore, while we have chosen ensembles of $K = 10$ trees for comparability with other algorithms, in general it is desirable to use more trees in ABORF. One important factor to consider when determining whether the number of trees is sufficient is the dimensionality of the data, especially if there exists a low dimensional manifold which contains most of the data. While individual trees in the ensemble do not learn from the data, and are thus robust to any differences in the data, the performance of the ensemble as a whole will be affected by its ability to capture any lower dimensional embedding of the data, as is true of all algorithms. Consequently, if it is expected that the data lie in a low dimensional manifold rather than being uniformly distributed in the feature space, one approach may be to choose a value for K that is high enough that the ensemble is likely to generate trees with discriminative splits along the manifold. Alternatively, if nothing is known about the data beforehand, an option to account for the existence of a low dimensional embedding may be to include a dimensionality reduction step for preprocessing the data before passing it to the trees.

6.5 Results and Discussion

In this section we first examine the different approaches for determining $\rho_{n,\nu}$ and the effectiveness of the approximation. We then compare the performance and run times of the ABORF algorithm with other contemporary approaches. Finally, we discuss the use of multivariate splits and the effect of ensemble size.

6.5.1 Selecting $\rho_{n,\nu}$

It is challenging to empirically evaluate the approximation we have made from the fully Bayesian model described in Sections 6.2.1 and 6.2.2, to the ABORF algorithm. This is especially so because we have switched from a conjugate prior model in the linear regression to an empirical Bayes approach in the approximation. However, two ways to do so are by examining the weights of the model, and the shape of the mean absolute error curves. Consequently, we compare the Bayesian result with the result of the leaf, tree and ensemble level approaches (see page 112) for determining $\rho_{n,\nu}$ on the synthetic FRIEDMAN and HYPERPLANE datasets.

In Table 6.3 we provide a summary of the performance of both models across all datasets

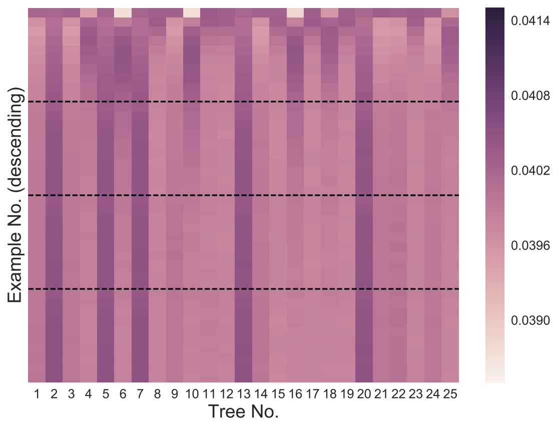
Table 6.3. Mean Absolute Error the Bayesian model described in Sections 6.2.1 and 6.2.2, and of the ensemble level ABORF approximation with no nominal features. Results have been averaged over 10 runs, with the standard error given in parentheses. The best performing algorithm(s) on each dataset is shown in bold.

	Bayesian	ABORF
Friedman	**1.43 (0.001)**	**1.43 (0.001)**
Hyperplane	0.281 (0.0001)	**0.265 (0.0001)**
Airline08	18.93 (0.02)	**17.68 (0.008)**
Zurich	67.3 (0.85)	**56.85 (0.049)**
YearMSD	8.14 (0.023)	**5.69 (0.007)**
Housing	**92122 (49.6)**	**92122 (49.6)**

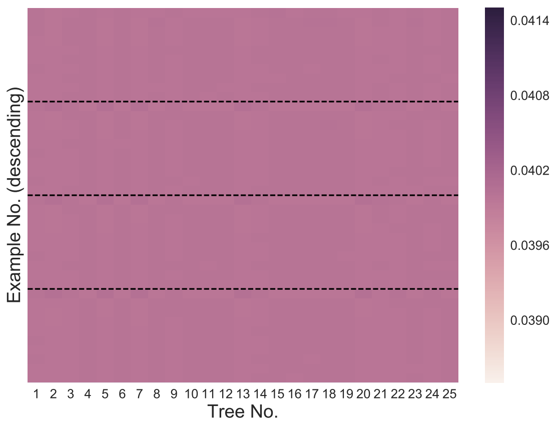
(excluding the use of nominal features) under a fixed prior of $\mathbf{\Lambda}_0 = 0.1\mathbf{I}$ for the Bayesian model and no regularization term in the ABORF approximation. It is worth noting that while the full Bayesian model is slower, for the purpose of our experiments it was not prohibitively so and we were able to perform 10 runs on each of the datasets. As can be seen in Table 4.1, this may be because other than the YEARMSD dataset, which contains relatively few examples, the other timeseries datasets have few numerical features, which would be the main slowing factors in our $O(D^3)$ update complexity implementation. We also note that, somewhat surprisingly, the ABORF approximation performs as well as, or outperforms, the Bayesian model in all cases. As we will see in the rest of this section, this may be due to the better adaption of the RMSProp linear models in the ABORF algorithm, compared to the Bayesian model.

However, experiments showed that under a uniform parametrization across trees of $\mathbf{\Lambda}_0 = 0.1\mathbf{I}$ for the ridge prior term in the fully Bayesian approach, the degree of change in the synthetic datasets is insufficient to illustrate the effectiveness of the ensemble weighting. Consequently, for the remainder of this section, when generating the Bayesian ensembles we have randomly drawn an integer i $\sim U[0, 5]$ for each tree. We have then set the strength of the prior used in the linear models of that tree to be $\mathbf{\Lambda}_0 = 10^{\mathrm{i}}\mathbf{I}$, to demonstrate the relevance of the ensemble weighting. This methodology means that the prior is very good for some of the trees, and very poor for others, which illustrates the effectiveness of the evidence weighting in the ensemble by giving it the opportunity to down-weight trees with poor priors. To ensure comparability, we have introduced fixed regularization terms for each tree in the ABORF models, set to the same value as the priors in the Bayesian trees, such that $c\mathbf{I} = 10^{\mathrm{i}}\mathbf{I}$ in Equation 6.45. This means that, at least initially, the same ridge penalty is applied to each tree in both the Bayesian model and ABORF.

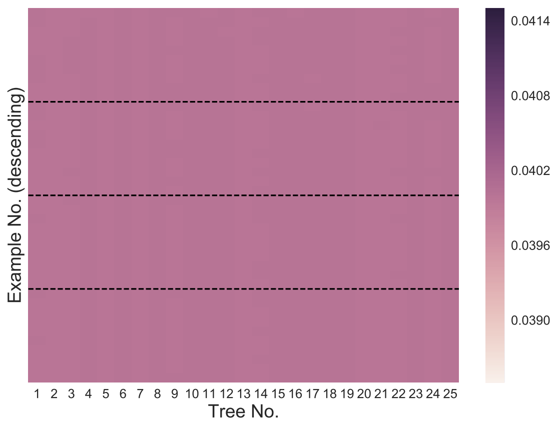
Figures 6.2 and 6.3 show the weights learned by an ensemble of 25 trees on the synthetic FRIEDMAN and HYPERPLANE datasets respectively, while the MAEs of each method and the are given in Figures 6.4 and 6.5. Perhaps the biggest surprise is that in the FRIEDMAN dataset, there is very little variation in the magnitude of the weights, with the scale of the



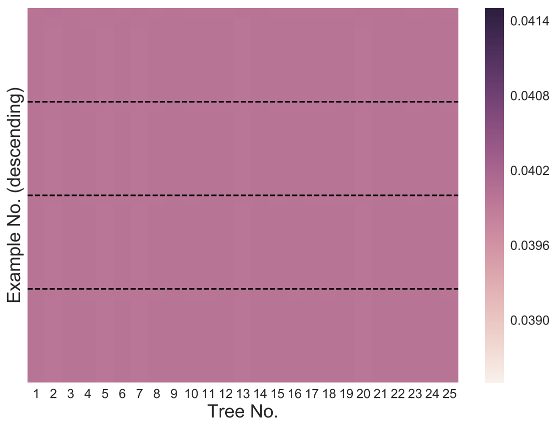
(a) BAYESIAN



(b) LEAF LEVEL APPROXIMATION



(c) TREE LEVEL APPROXIMATION

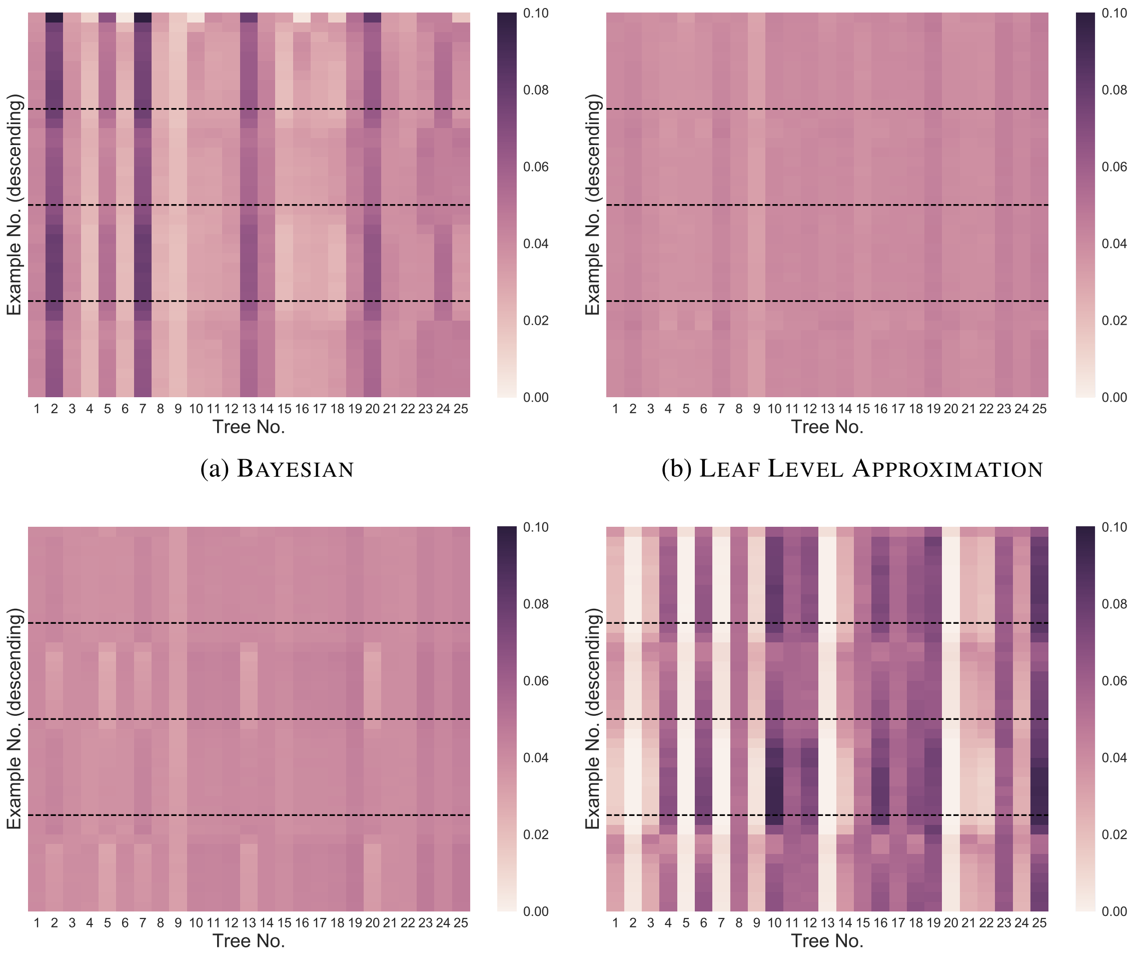


(d) ENSEMBLE LEVEL APPROXIMATION

Figure 6.2. Example ensemble weights on the FRIEDMAN dataset. Weights have been averaged over a window of 25000 examples. The colour scale runs from 0.0385 to 0.0415 on every graph and the dashed horizontal lines illustrate the times at which changes occur in the dataset.

weights in Figure 6.2 only ranging from 0.0375 to 0.0475, at most a 20% deviation from the average weight of $1/25$. Furthermore, the only noticeable variation is in the weights of the Bayesian model, in which some of the trees are persistently slightly higher-weighted. It is also worth noting from Figure 6.2a that trees 10, 16, 18, 25 are initially slightly higher-weighted, but become less important after the first change in the data, suggesting that the changes have an impact on the weights.

Although there are minor differences in the weights, Figure 6.4 indicates that all of the approximation approaches are equally effective, returning the same results as the benchmark scenario of equal weights. Furthermore, the performance of the Bayesian approach is also equivalent to the benchmark scenario. Despite this, the Bayesian and ABORF approaches show very distinct learning curves. As these difference show in the case where the weights are all effectively equal, this suggests that the approximation for $\hat{\beta}_n$ in Section 6.2.3 is responsible for the differences. The faster initial learning rate and adaption to changes in the ABORF approach is likely due to the very large step size of $\eta = 0.25$. However, this may also lead to overtraining and contribute to the ABORF



(a) BAYESIAN (b) LEAF LEVEL APPROXIMATION

(c) TREE LEVEL APPROXIMATION (d) ENSEMBLE LEVEL APPROXIMATION

Figure 6.3. Example ensemble weights on the HYPERPLANE dataset. Weights have been averaged over a window of 25000 examples. The colour scale runs from 0.0 to 0.1 on every graph and the dashed horizontal lines illustrate the times at which changes occur in the dataset.

approaches stopping learning at an MAE of 1.5. Alternatively, this reduced learning may also be due to the regularization term in ABORF being fixed, unlike in the Bayesian models in which the effect of the prior is reduced as more data is seen.

On the other hand, the results on the HYPERPLANE dataset are much more telling. Although Figures 6.3b and 6.3c again show limited variation in the weights of the leaf and tree level approximations, both Figures 6.3a and 6.3d exhibit a marked variation in the weights, showing that both the Bayesian and ensemble level approximations have a consistent preference for certain trees. Furthermore, the generally higher-weighted trees react to changes in the dataset, with their weights increasing and decreasing after the changes occur. We note that these changes in the weights do not happen immediately, likely due to the incremental drift occurring over 100000 points. Another point of interest is that the higher-weighted trees in the Bayesian model are generally lower-weighted in the ensemble approximation, and vice versa. As these differences are consistent over the entire dataset, this suggests that the optimal values for the priors in the Bayesian approach and for the

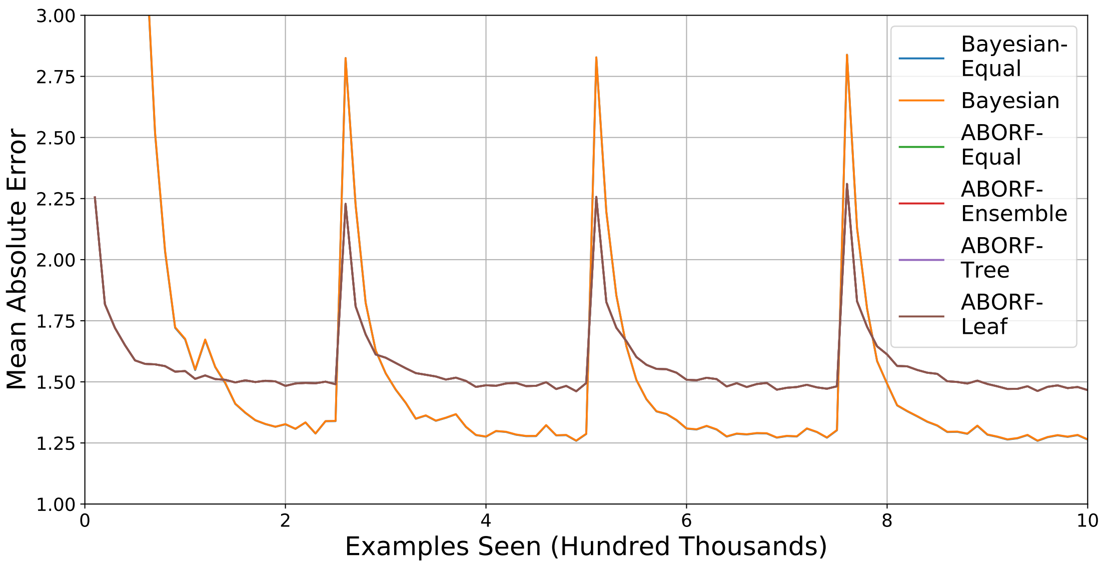


Figure 6.4. Mean Absolute Error of the Bayesian method and ABORF approximations on the FRIEDMAN dataset, compared to the benchmark of equal weights. The performance of each "ABORF" run is almost identical, showing as the red line, while the identical "Bayesian" approaches show as the orange line. Results have been averaged over 10 runs and smoothed over a window of 10000 points.

regularization in the approximation are likely different.

From Figure 6.5, we can see that the varied weights lead to improvements in performance in both the Bayesian and ABORF methods, relative to the benchmark scenarios of equal weights. While all of the ABORF approximations show improvement over the benchmark, the Ensemble level approximation performs especially well, probably due to the greater variation in the weights. Furthermore, the ABORF-ensemble method somewhat surprisingly also outperforms the Bayesian approach. While somewhat counter-intuitive, this may in part be due to the benefits of the extreme learning rate allowing the trees in the ABORF approach to rapidly adapt to changes. We also note that the shape of the benchmark MAEs are again different in both cases, suggesting that the differences in shape between the Bayesian and ABORF learning curves are again due to the approximation for $\hat{\boldsymbol{\beta}}_n$ in Section 6.2.3.

Overall, setting $\rho_{n,\nu}$ at the ensemble level appears to provide the best approximation to the original Bayesian approach. The weights vary on similar magnitudes in both cases, and although the optimal values for the priors in the Bayesian approach and for the regularization in the approximation appear different, both algorithms exhibit similar characteristics on the HYPERPLANE data, suggesting that the weight approximation described in Section 6.2.3 is effective. Furthermore, the ensemble level approximation has the best performance relative to the other approximations. Consequently, for the remainder of this chapter we proceed using the ensemble level approximation.

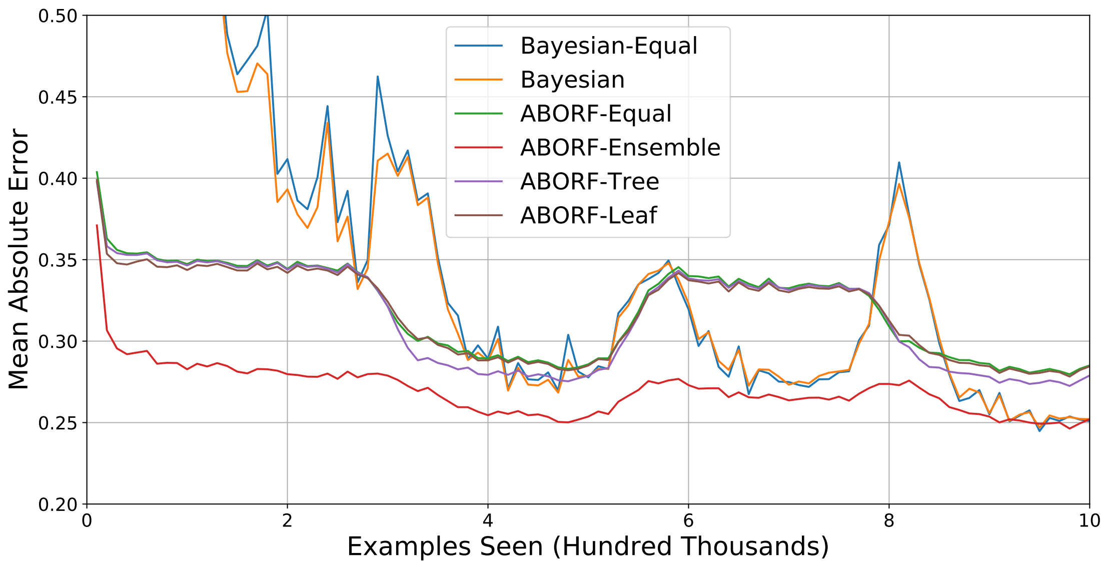


Figure 6.5. Mean Absolute Error of the Bayesian and ABORF approximations on the HYPERPLANE dataset, compared to the benchmark of equal weights. Results have been averaged over 10 runs and smoothed over a window of 10000 points.

Table 6.4. Mean Absolute Error of different methods. Results have been averaged over 10 runs, with the standard error given in parentheses. The best performing algorithm on each dataset is shown in bold.

	FIMT-DD	ORTO-A	ARF-Reg	ABORF
Friedman	1.79 (0.001)	2.14 (0.00)	**1.20** (0.001)	1.43 (0.001)
Hyperplane	0.418 (0.0003)	0.722 (0.0005)	0.268 (0.0001)	**0.265** (0.0001)
Airline08	19.47 (0.008)	20.21 (0.00)	20.95 (0.013)	**16.22** (0.007)
Zurich	64.27 (0.025)	63.49 (0.00)	65.74 (0.031)	**48.99** (0.048)
YearMSD	10.1 (0.08)	17.12 (0.00)	11.59 (0.011)	**5.69** (0.007)
Housing	97724 (48.4)	92,121 (0)	102627 (48.4)	**57293** (45.3)
Average Rank	2.67	3.17	3	1.17

6.5.2 Comparative Performance

Table 6.4 shows the performance of variations of the ABORF algorithm, relative to the benchmark FIMT-DD and contemporary ORTO-A and ARF-Reg algorithms, with Figures 6.6 - 6.11 showing a more detailed comparison of the performance on each dataset. On each of the real-world datasets, the ABORF algorithm performs well comparatively, having the best performance in each case. However, it is outperformed by the ARF-Reg algorithm on the synthetic FRIEDMAN dataset. This distinct difference is possibly due to the long stationary periods with little noise in the FRIEDMAN dataset, something not typically found in real-world datasets. This can be further seen in the HYPERPLANE dataset, where in the stationary periods, ARF-Reg has better performance, but is not able to update as easily in the presence of drift.

Table 6.5. Runtime of different methods in seconds. Results have been averaged over 10 runs, with the standard error given in parentheses. The best performing algorithm(s) on each dataset is shown in bold.

	FIMT-DD	ORTO-A	ARF-Reg	ABORF
Friedman	33 (0.4)	**32 (1)**	157 (2.6)	63 (0.3)
Hyperplane	**28 (0.2)**	**28 (0)**	146 (5.6)	46 (0.2)
Airline08	106 (1.2)	**94 (1)**	541 (1.5)	271 (2.0)
Zurich	106 (6.2)	**75 (1)**	886 (227.5)	307 (3.2)
YearMSD	97 (0.7)	**78 (0)**	554 (10.5)	170 (0.7)
Housing	**233 (4.7)**	278 (2)	2078 (7.3)	1066 (7.0)
Average Rank	1.67	1.17	4	3

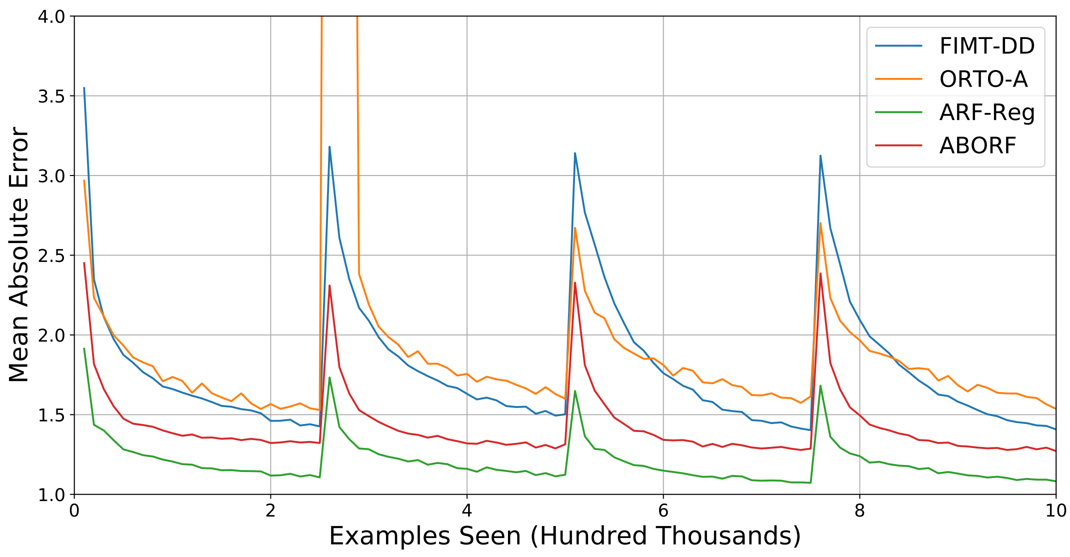


Figure 6.6. Mean Absolute Error on the FRIEDMAN dataset. Results have been averaged over 10 runs and smoothed over a window of 10000 points.

It is again worth noting the performance of the algorithms on the FRIEDMAN, HYPERPLANE and YEARMSD datasets, which do not contain nominal features, in order to determine whether the difference in performance is due to the inclusion of nominal features in ABORF. However, unlike in Chapter 4 , it is inappropriate to remove the linear models from the leaves for comparison, since they are an intrinsic part of the ABORF approach. In terms of performance, we see that ABORF considerably outperforms the other algorithms on the YEARMSD data, and also has the best performance on the synthetic HYPERPLANE dataset. However, as ARF-Reg performs best on the FRIEDMAN dataset, it is difficult to determine how much of the better performance in ABORF is due to the inclusion of nominal features.

On the other hand, in terms of timing the ABORF algorithm is faster than the comparative ARF-Reg ensemble algorithm in every case, as can be seen in Table 6.5. While this result is expected due to the simplicity of updating the ABORF algorithm, these run times could be further reduced using parallelization. In contrast, ABORF is slower than the FIMT-DD and

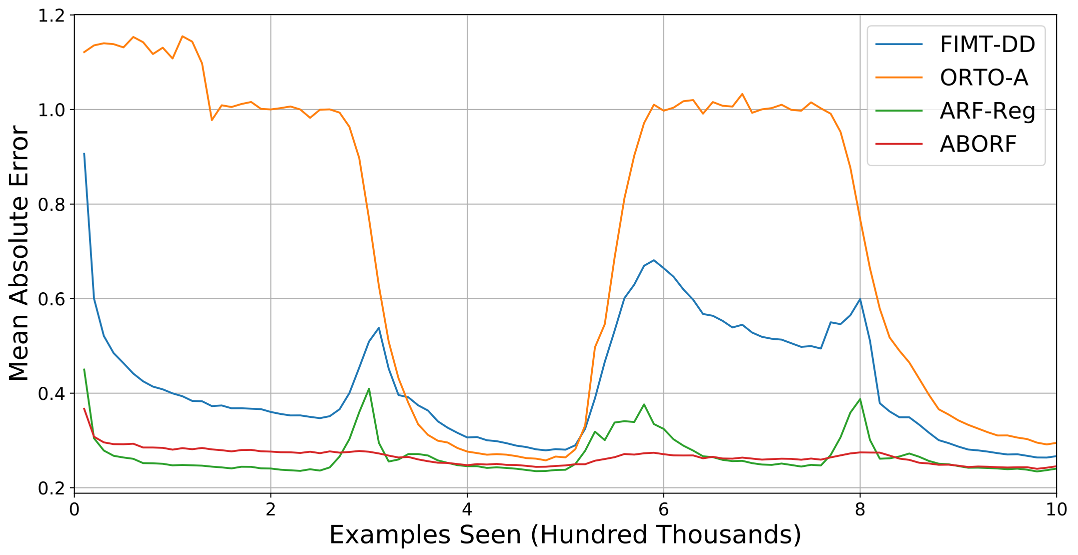


Figure 6.7. Mean Absolute Error on the HYPERPLANE dataset. Results have been averaged over 10 runs and smoothed over a window of 10000 points.

ORTO-A approaches in every case. This is unsurprising since the FIMT-DD tree is a single learner and the training approach in the ORTO-A algorithm is, by design, a faster adaption of the FIMT-DD algorithm. This in turn means that the update time of both FIMT-DD and ORTO-A is equivalent to that of a single tree, rather than an ensemble of trees. However, this also means they are consequently unable to benefit from parallelization, and that a parallel implementation of the ABORF ensemble may reduce the disparity in wall clock times. Additional results detailing the predictive performance vs. run time tradeoff can be found in Appendix A.2

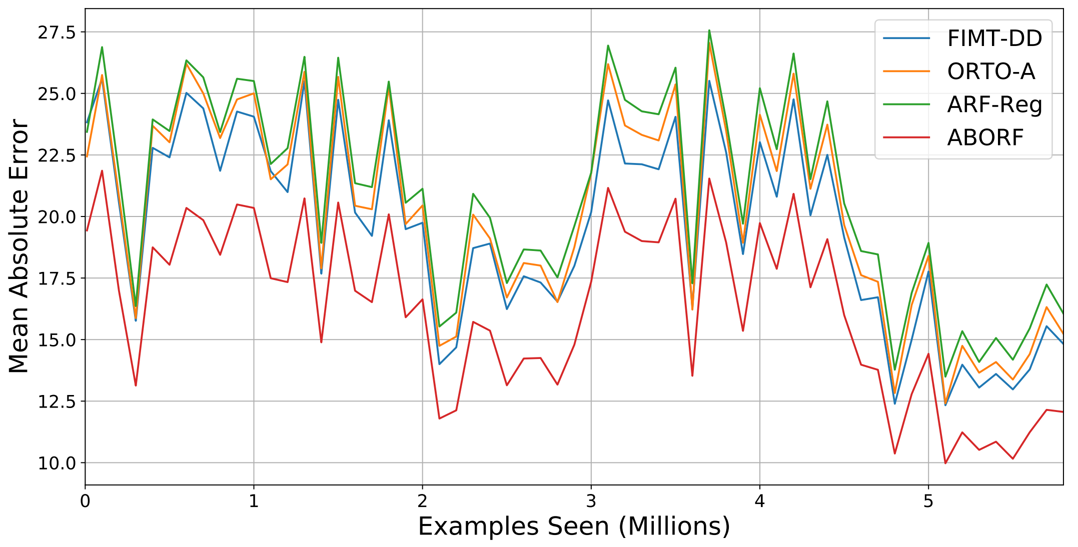


Figure 6.8. Mean Absolute Error on the AIRLINE08 dataset. Results have been averaged over 10 runs and smoothed over a window of 100000 points.

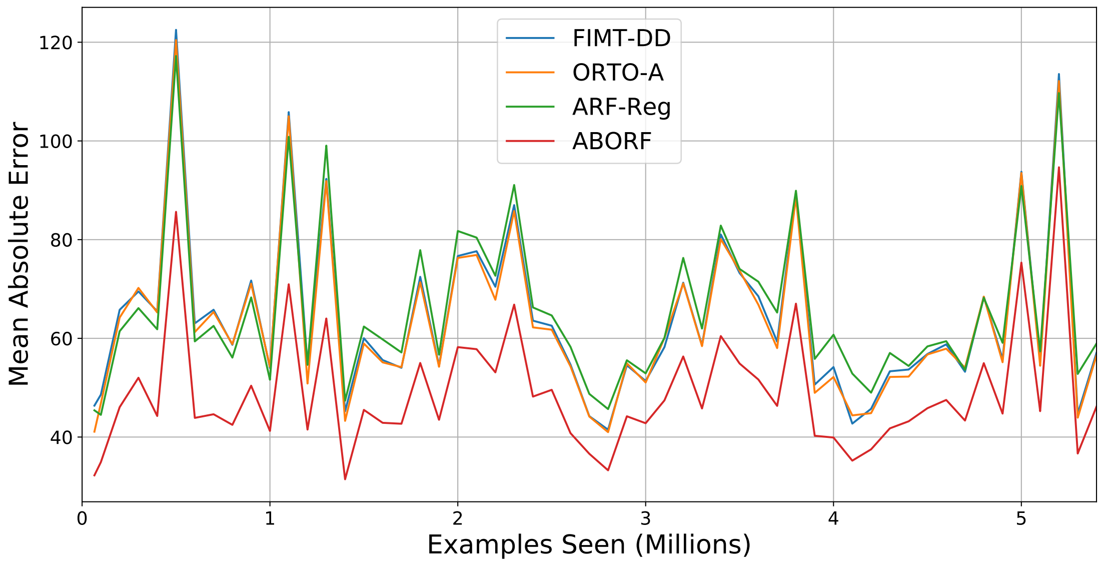


Figure 6.9. Mean Absolute Error on the ZURICH dataset. Results have been averaged over 10 runs and smoothed over a window of 10000 points.

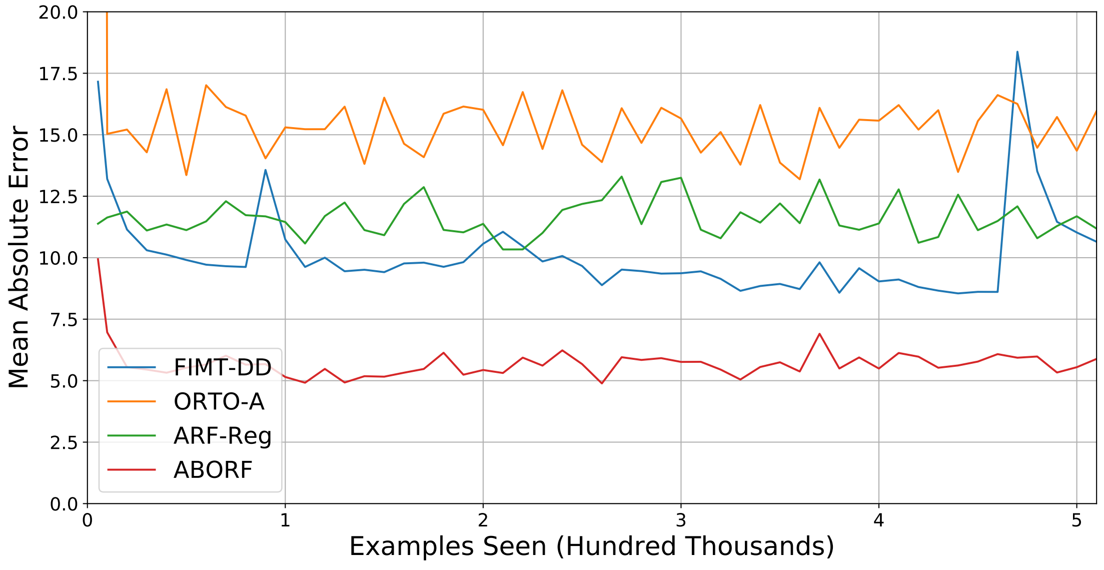


Figure 6.10. Mean Absolute Error on the YEARMSD dataset. Results have been averaged over 10 runs and smoothed over a window of 10000 points.

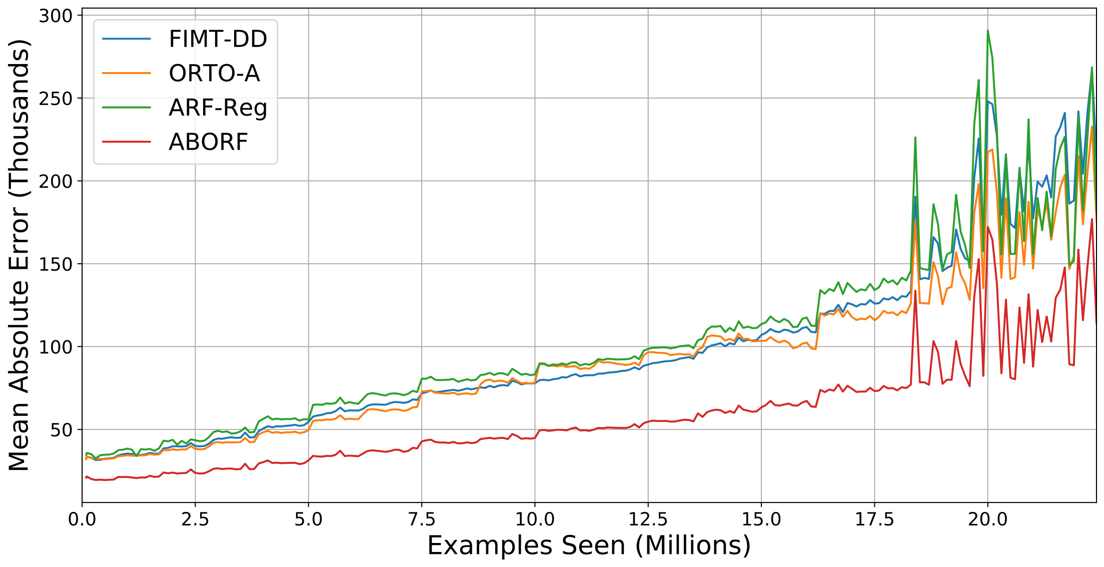


Figure 6.11. Mean Absolute Error on the HOUSING dataset. Results have been averaged over 10 runs and smoothed over a window of 100000 points.

Table 6.6. Mean Absolute Error of the ABORF in both univariate and multivariate cases. Results have been averaged over 10 runs, with the standard error given in parentheses. The best performing algorithm on each dataset is shown in bold.

	ABORF - Univariate	ABORF - Multivariate
Friedman	1.5 (0.001)	**1.43 (0.001)**
Hyperplane	0.272 (0.0001)	**0.265 (0.0001)**
Airline08	**16.19 (0.007)**	16.22 (0.007)
Zurich	50.42 (0.048)	**48.99 (0.048)**
YearMSD	5.78 (0.006)	**5.69 (0.007)**
Housing	57372 (45.3)	**57293 (45.3)**

6.5.3 Multivariate Splits

We have opted for the use of multivariate splits over traditional univariate (or, as they are more commonly know, axis-aligned) splits in the ABORF algorithm. This is to enable the depth-capped trees to capture oblique feature dependence while being relatively shallow. To verify the benefits of choosing multivariate splits, we compare the performance of both splitting approaches in Table 6.6, while a graphical comparison can be seen in Figure 6.12.

While there is an improvement on most datasets, there appears to be little difference on a few of them. In the case of the HOUSING dataset, the lack of difference is expected, since multivariate splits only occur on continuous features. However, it is more surprising that there is little difference on the AIRLINE08 dataset. This could be due to features being largely independent, so there is little benefit to be had from performing multivariate splits, or it could be due to large amounts of noise in the data obscuring any benefits.

Examining the performance on the synthetic FRIEDMAN and HYPERPLANE datasets in

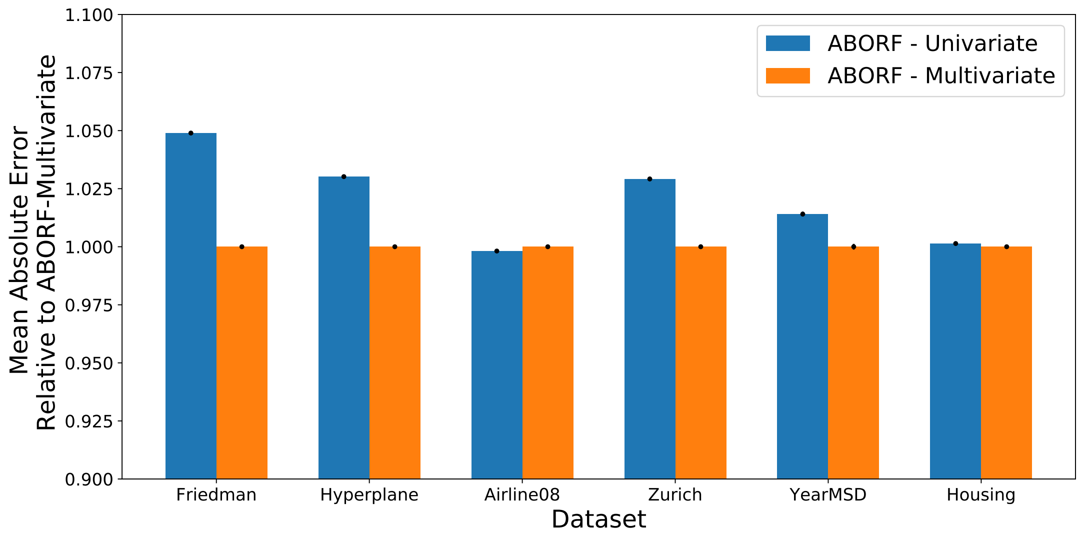


Figure 6.12. A comparison between multivariate and univariate splitting on each dataset. The results for each dataset are relative to the result of the multivariate case on that dataset, such that the multivariate result is always 1. The error bars for the AIRLINE08 and HOUSING datasets overlap.

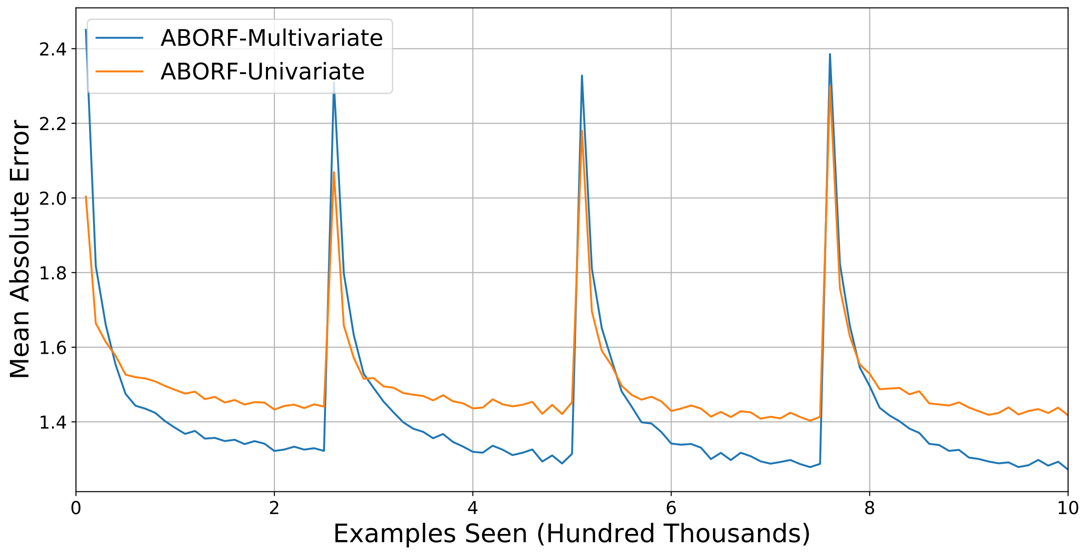


Figure 6.13. A comparison between multivariate and univariate splitting on the FRIEDMAN dataset.

more detail in Figures 6.13 and 6.14, we see that the performance improvement is largely consistent across the whole dataset which suggests that, in a stationary scenario, the multivariate splits would be expected to outperform the univariate splits. However, at both the beginning of and after each change in the FRIEDMAN dataset, the univariate approach is temporarily better. While this does not persist for a long period of time, it may indicate that in a situation with rapid abrupt changes, a univariate approach may be beneficial. Full results of the comparison on each of the other datasets can be seen in Appendix A.3

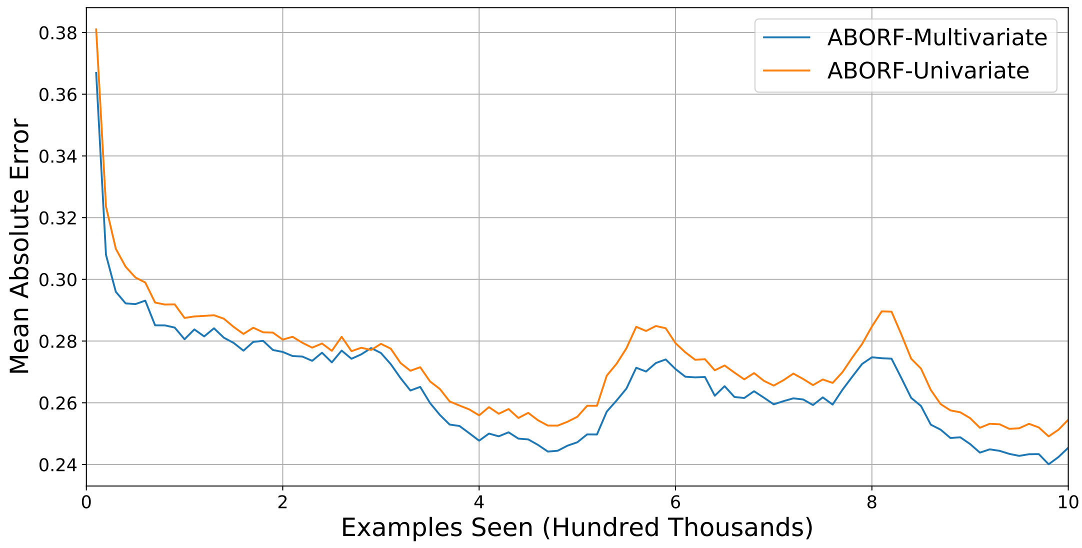


Figure 6.14. A comparison between multivariate and univariate splitting on the HYPERPLANE dataset.

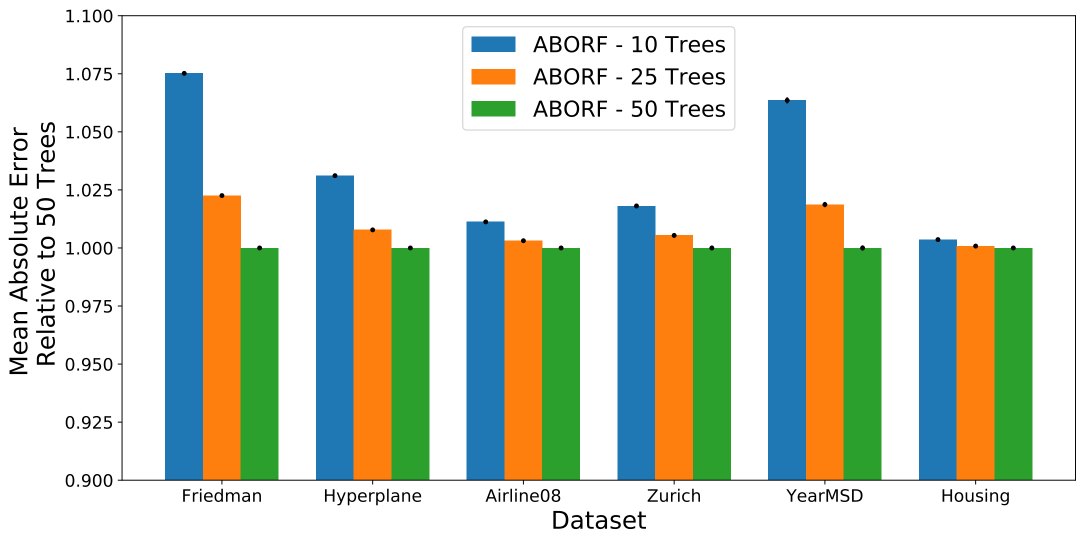


Figure 6.15. The effect of ensemble size on each dataset.

6.5.4 Ensemble Size

While we have used an ensemble of 10 trees for comparison with other algorithms, we might expect that to take advantage of the randomisation in the ABORF algorithm, a larger ensemble would be beneficial. Figure 6.15 and Table 6.7 show the results of using larger ensemble sizes. A detailed comparison on each dataset can be found in Appendix A.4.

Increasing the ensemble size leads to an improvement in almost all cases. The only unclear case is between 25 and 50 trees on the HOUSING dataset, on which the standard errors overlap. This improvement suggests that a larger ensemble than 10 trees is optimal, which we might expect, although without parallelization this comes with the trade-off of being slower.

Table 6.7. Mean Absolute Error of the ABORF with 10, 25, and 50 trees. Results have been averaged over 10 runs, with the standard error given in parentheses. The best performing algorithm on each dataset is shown in bold.

	ABORF - 10 Trees	ABORF - 25 Trees	ABORF - 50 Trees
Friedman	1.43 (0.001)	1.36 (0.001)	**1.33 (0.001)**
Hyperplane	0.265 (0.0001)	0.259 (0.0001)	**0.257 (0.0001)**
Airline08	16.22 (0.007)	16.09 (0.007)	**16.04 (0.007)**
Zurich	48.99 (0.048)	48.38 (0.048)	**48.12 (0.048)**
YearMSD	5.69 (0.007)	5.46 (0.006)	**5.35 (0.006)**
Housing	57293 (45.3)	57133 (45.3)	**57087 (45.3)**

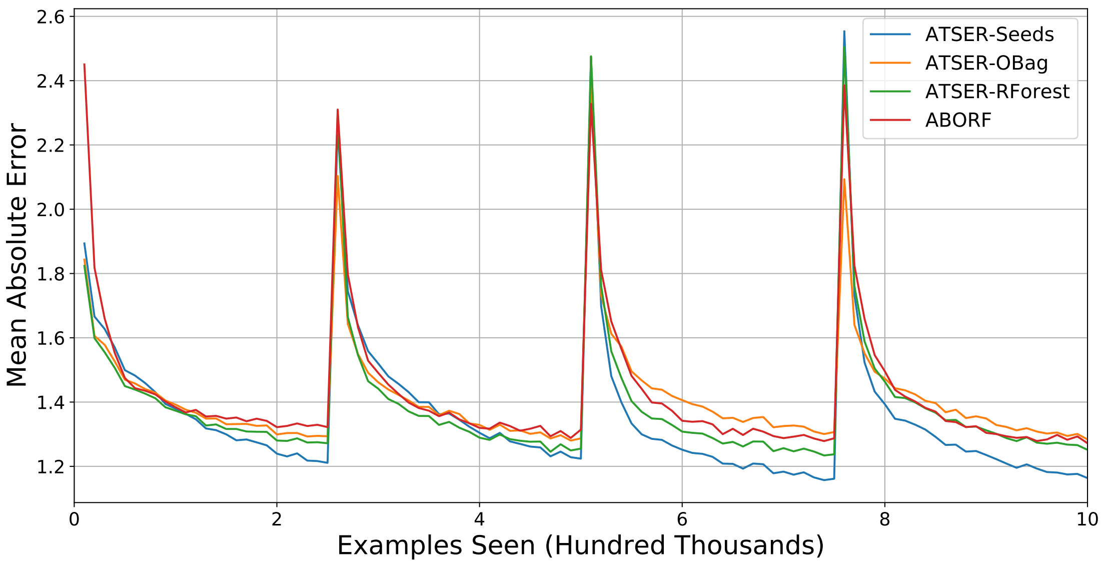


Figure 6.16. A comparison between 10 tree ensembles of ATSER and ABORF on the FRIEDMAN dataset.

The largest relative improvements occur on the FRIEDMAN and YEARMSD datasets between 10 and 25 trees. This is not a surprising result, since we expect that these datasets are largely stationary, which means an improvement is able to show over a larger portion of the dataset. On the other hand, increasing the ensemble size seems to do little to improve performance on the HOUSING dataset. This is somewhat more surprising, and may in part be due to a large portion of the error coming from the incremental drift across the dataset due to house prices increasing over time.

6.5.5 Comparison with ATSER

Having introduced and compared ABORF with a number of algorithms, we now also compare the performance with the various ATSER ensembles introduced in Chapter 4. We use the previously suggested parameters for the ATSER ensembles and report results based on ensembles of both 10 and 50 trees, as shown in Table 6.8. Figures 6.16 to 6.21 show a breakdown of the 10 tree results on each of the datasets, while a breakdown for ensembles

Table 6.8. Mean Absolute Error of ABORF and ATSER with both 10 and 50 trees. Results have been averaged over 10 runs, with the standard error given in parentheses. The best performing algorithm on each dataset is shown in bold.

		ATSER-Seeds	ATSER-RForest	ATSER-OBag	ABORF
10 Trees	Friedman	**1.36 (0.01)**	1.39 (0.01)	1.40 (0.01)	1.43 (0.001)
	Hyperplane	0.28 (0.000)	0.289 (0.000)	0.293 (0.000)	**0.265 (0.000)**
	Airline08	**14.97 (0.02)**	15.07 (0.02)	16.39 (0.01)	16.22 (0.007)
	Zurich	39.90 (0.11)	45.20 (0.10)	**38.78 (0.05)**	48.99 (0.048)
	YearMSD	7.52 (0.01)	7.71 (0.01)	8.44 (0.01)	**5.69 (0.007)**
	Housing	**55857 (25)**	55933 (21)	59268 (52)	57293 (45.3)
	Average Rank	1.5	2.5	3.33	2.67
50 Trees	Friedman	1.31 (0.001)	1.34 (0.001)	**1.3 (0.001)**	1.33 (0.001)
	Hyperplane	0.27 (0.000)	0.279 (0.001)	0.271 (0.000)	**0.257 (0.000)**
	Airline	**14.69 (0.008)**	14.77 (0.008)	15.91 (0.007)	16.04 (0.007)
	Zurich	38.25 (0.047)	43.51 (0.048)	**36.78 (0.047)**	48.12 (0.048)
	YearMSD	6.39 (0.007)	6.52 (0.007)	6.72 (0.006)	**5.35 (0.006)**
	Housing	**55669 (46.3)**	55707 (46.4)	58525 (46.6)	57087 (45.3)
	Average Rank	1.67	3.0	2.67	2.67

of 50 trees in given in Appendix A.5.

As can be seen from Table 6.8, the ASTER-Seeds ensemble appears to have the strongest performance overall, regardless of whether the ensembles are of 10 or 50 trees. However, we note that ABORF has stronger performance on the HYPERPLANE and YEARMSD datasets, suggesting that the optimal algorithm likely depends on the dataset.

In particular, we observe from Figure 6.16 that the MAE curve for ABORF flattens out toward the end of each stationary section, unlike the ATSER curves, suggesting that ABORF may be less effective for stationary scenarios, as noted previously in Section 6.5.2. Furthermore, both ATSER and ABORF appear to react similarly to the abrupt changes. On the other hand, Figure 6.17 suggests that ABORF is much more robust to incremental drift, since although it exhibits mediocre performance over the initial stationary section, after the incremental changes occur ABORF is much less affected.

6.6 Summary

In this chapter we have introduced a fully Bayesian approach for regression data stream predictions and shown it is able to perform well in the presence of concept drift. We have then approximated this model to develop the ABORF algorithm, which we have empirically shown to be faster, while retaining strong performance in comparison to both the original Bayesian model, and also in comparison to other contemporary algorithms.

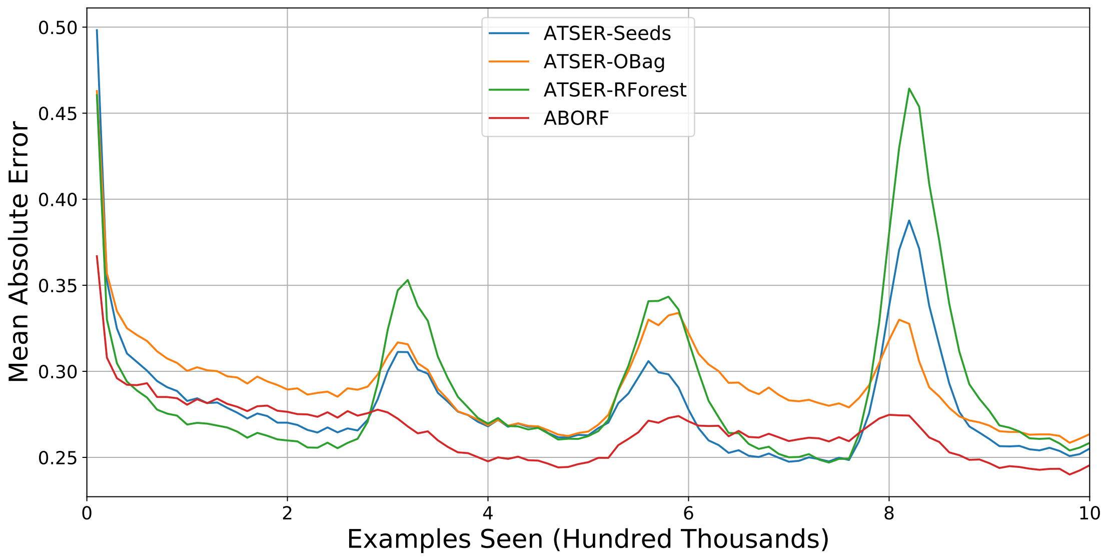


Figure 6.17. A comparison between 10 tree ensembles of ATSER and ABORF on the HYPERPLANE dataset.

We have also investigated a number of aspects of ABORF, providing three different approaches for the approximation, and further examining the impact of our decision to use multivariate splits as opposed to tradition axis-aligned splits. Furthermore, we have also tested the impact of ensemble size on the performance of the ABORF algorithm. Finally, we have presented a comparison with the ATSER ensembles previously introduced in Chapter 4.

Overall, we have shown that the ABORF algorithm is simple and easy to implement, requiring fewer hyperparameters that its contemporary counterparts. We have demonstrated that the use of multivariate splits leads to improved performance over the traditional axis-aligned choice and we have shown that while larger ensembles lead to improved performance, even smaller ensembles of 10 trees are able to perform comparably to other streaming algorithms.

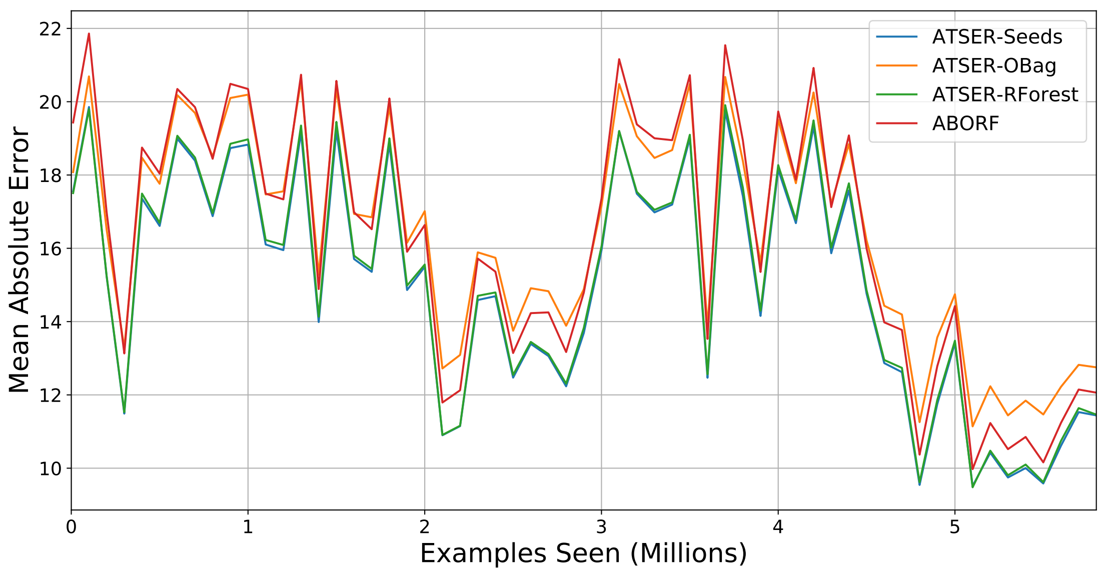


Figure 6.18. A comparison between 10 tree ensembles of ATSER and ABORF on the AIRLINE08 dataset.

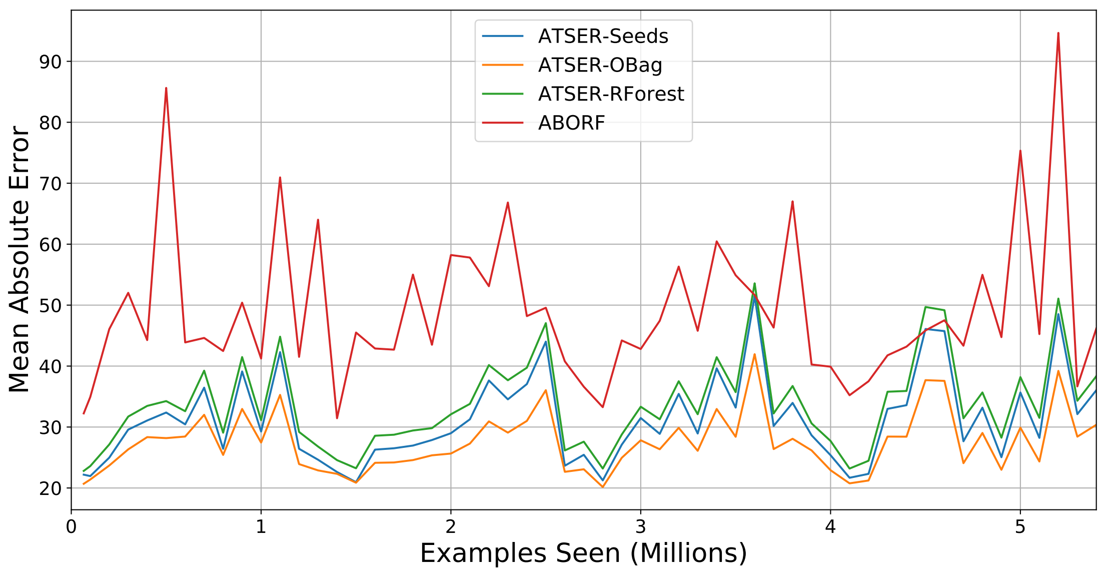


Figure 6.19. A comparison between 10 tree ensembles of ATSER and ABORF on the ZURICH dataset.

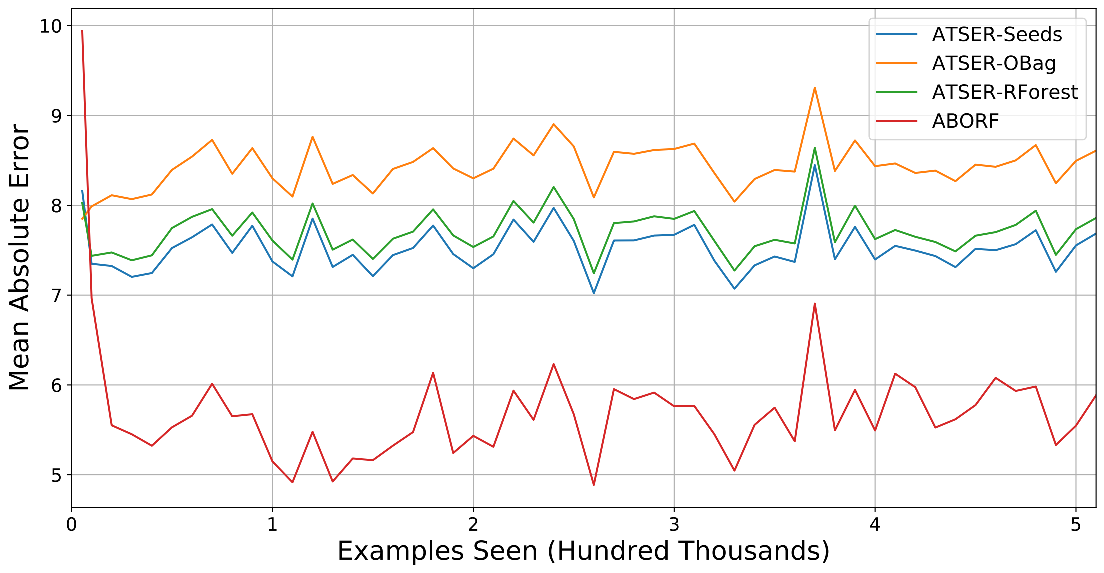


Figure 6.20. A comparison between 10 tree ensembles of ATSER and ABORF on the YEARMSD dataset.

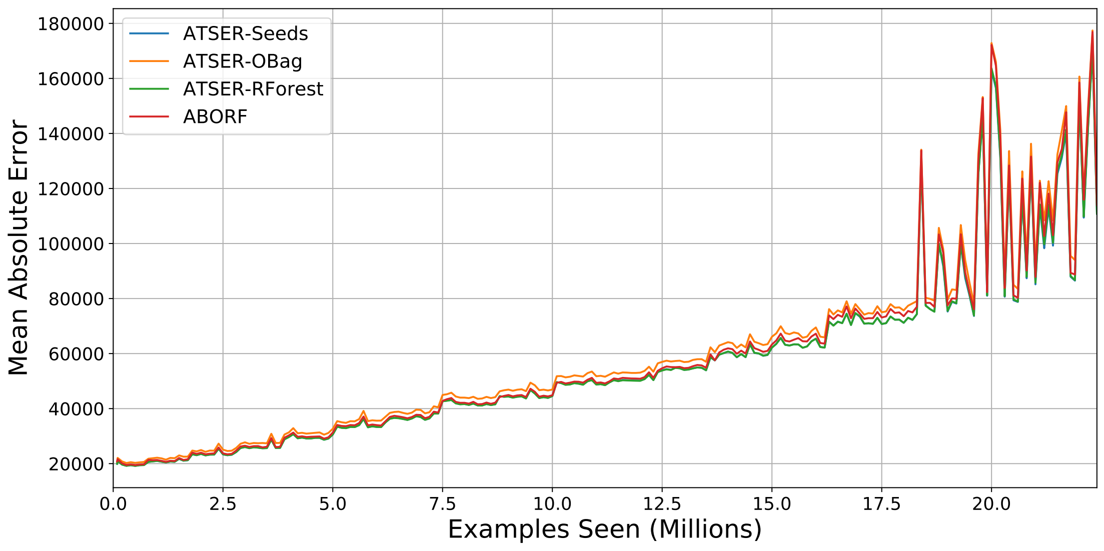


Figure 6.21. A comparison between 10 tree ensembles of ATSER and ABORF on the HOUSING dataset.

Chapter 7

Conclusion

In this thesis we have suggested algorithms for online learning, primarily focussing on regression, but also including adaptions for classification. We have provided empirical evaluations of each, investigating performance, timings, and sensitivity. The design of the algorithms we have introduced is to incrementally train decision tree based models, with linear models in the leaves. These trees are then combined into ensembles and used to make predictions on a data stream. Each of the algorithms we have provided focus on the use of randomness to improve both predictive performance and run times, compared to traditional online algorithms.

In Chapter 4 we introduced the ATSER algorithm, designed for online predictive modelling in regression streaming problems. It builds upon the previous FIMT-DD algorithm, offering improvements in areas such as the splitting process and modelling in the leaves. Motivated by the Extra Trees algorithm, it introduces the concept of extreme randomisation to an online learning setting with a novel two stage splitting process, which at each potential split point rapidly selects one random split per feature, before secondly choosing the best of the randomly selected splits.

We have shown that the use of ATSER leads to improvements in predictive performance compared to the benchmark FIMT-DD algorithm, and that a large part of this can be attributed to the randomized splitting in the online tree generation process. Simultaneously, we have also shown that the use of ATSER increases the training speed of the tree, although this is partially offset by the inclusion of nominal features, which are not used in the comparative algorithms.

Taking ATSER trees as a base learners, we have proposed several ensemble algorithms for regression streams, allowing further improvements in performance when compared to other contemporary ensemble algorithms on a number of real world datasets. This is especially so when parallel hardware is available, as is often the case on modern machines.

Adapting ATSER to a classification scenario in Chapter 5, we have introduced the ACTSER algorithm for predictive modelling which incrementally trains classification trees using extreme randomization in the splitting process. In a classification scenario, the ease of including nominal splits in this process is particularly valuable, since the traditional incorporation of nominal splits for classification trees is particularly cumbersome. Taking a number of real world datasets, we have again demonstrated the comparable performance of the algorithm relative to other popular individual learners, namely the VFDT and EFDT.

As with ATSER, we have again provided ensemble algorithms which use ACTSER trees as base learners: ACTSER-Seeds and ACTSER-RForest. Using real world datasets, we have shown that they both have strong predictive performance relative to the contemporary ARF ensemble, as well being significantly faster in most cases.

Finally, in Chapter 6 we have developed the ABORF ensemble algorithm for use in a regression streaming scenario, which creates a fully randomized ensemble of trees before any data has been seen. The trees are formed with multivariate splits, enabling shallow trees to capture non-linear relationships in the features. As the trees are not dependant on the data, we are able to avoid the need to explicitly detect change and adapt in the trees. Instead, learning occurs in the evidence-based ensemble weights, and in linear models in the leaves of the trees, both of which are able to react to concept drift through the inclusion of a decay factor for previously learnt information.

We have demonstrated that despite having completely randomized tree structures, ABORF is well suited to data streaming scenarios. We have shown empirically that, although slower than the single learners FIMT-DD and ORTO-A, relative to other regression streaming ensemble algorithms it is faster while retaining strong performance. Furthermore, without the need for explicit change detection and adaption, as is common in other contemporary streaming algorithms, ABORF has fewer hyperparameters to choose and therefore requires less prior knowledge about the data stream it is to be used on, making it suitable for “out of the box” use.

7.1 Future Work

We conclude with possible directions for future research. Throughout this thesis, we have suggested a number of alternative approaches for predictive modelling with time-changing data streams. We have developed both the ATSER and ACTSER approaches from existing algorithms and, consequently, they follow with common challenges across many data streaming algorithms. In particular, as seen in Section 3.2.1, change detection is a challenging topic. As a result, change detectors are a potential source of inefficiency in the algorithm, with every incorrect detection leading to the unnecessary construction of an alternative subtree, while every missed detection of an actual change can lead to poorer performance than might be expected if the change were detected. Furthermore, a single change can be detected in a series of nodes in the same path down the tree, leading to an unnecessary number of alternative subtrees being grown. To this end, there is a need to examine the efficiency of the detectors, and whether a sparser approach is possible.

Another area linked to change detection in which both ATSER and ACTSER may benefit is that we have currently only included mechanisms to adapt to change in the base learners. It may be possible to extend the change adaption to the entire ensemble as a whole, which can also improve predictions. We have done this to a certain extent in ABORF, in which we have modified the contribution of individual learners to the ensemble prediction based on their evidence. While this method may not be suitable for ATSER or ACTSER, other approaches may be more appropriate, such as weighting schemes [Brzeziński and Stefanowski, 2011] or stacked generalization [Wolpert, 1992].

Research is also needed on extending both ATSER and ACTSER to multi-target problems. While this may be possible through similar approaches to those taken by Ikonomovska et al. [2011a] and Osojnik et al. [2018], these methods work by choosing the best split based on a weighted combination of the split evaluations for each target, which is not directly compatible with the idea of randomly choosing splits. As a result, further investigation is needed into whether combining the ATSER and ACTSER algorithms with such an approach is feasible.

On the other hand, the ABORF algorithm behaves very differently to traditional streaming data algorithms. Without the need for explicit change detection, the change adaption is tied to the learning rate in the leaves and the decay of older data. However, it is not immediately clear how to apply the decay rates and further work is necessary to determine this. In ABORF we decay parameters in the leaves each time a new example is seen in that leaf, effectively handing all leaves of the tree an equal weight. While this ensures that each leaf is appropriate for current data, some leaves may be used more recently than others, and alternative approaches may be more effective, such as decaying parameters at *every* leaf in the tree each time a new example is seen.

Another consideration for ABORF is that we weight the trees in the ensemble based on the average performance of the tree across all nodes. While this is a theoretically motivated approach, further investigation is needed into whether other methods may be more effective. One possible method may be to use the evidence for the predicting leaf as a proxy for the tree weight at each time, thereby obtaining more local weights and allowing trees which perform well in some regions and badly in others to be weighted accordingly, rather than being moderately weighted at all times, as they would be in the current ABORF algorithm.

The feature scaling in ABORF could also be improved. Ideally, the transformation would map incoming features to uniformly span the $[0, 1]$ range, meaning the number of examples seen by each leaf in the tree would be directly proportional to its hypervolume. However, the min-max feature scaling we are currently using is susceptible to concept drift and outliers, since if the bounds expand relative to the spread of the data then the examples will become clustered after being transformed. Furthermore, since the features may not initially be uniformly distributed, a transform which maps features to be uniformly distributed is also desirable. Therefore, while min-max feature scaling works well empirically, other methods of normalisation may be more robust to drift and improve the distribution of data to the leaves.

A final direction for future work is in the characterization and suitability of different datasets for evaluating online algorithms. One common feature across both Chapters 4 and 6 is that, relative to comparative methods, both ATSER and ABORF seem much better suited to the real-world datasets rather than the synthetic FRIEDMAN and HYPERPLANE datasets. This is likely a result of both ATSER and ABORF involving a comparatively high degree of randomization, which is well known to lead to stronger performance in the noisy environments which characterize the real-world datasets. This view is further supported by the distinctive shapes of the learning curves on the synthetic datasets which, especially on the FRIEDMAN data, are predictable and contain little noise in comparison to the noisy shape of the learning curves on the real-world data. Consequently, it may be worth using the categorizations of drift in Chapter 2 (and possibly extending them to include "noisiness") to attempt to categorize real-world datasets or generate a larger pool of synthetic datasets for online regression scenarios, which would allow methods to be better evaluated in terms of their suitability to different types of data and drift.

Appendices

Appendix A

Additional Experimental Results

A.1 Additional Results for Section 4.5.4: MAE of Algorithms without Linear Models in the Leaves

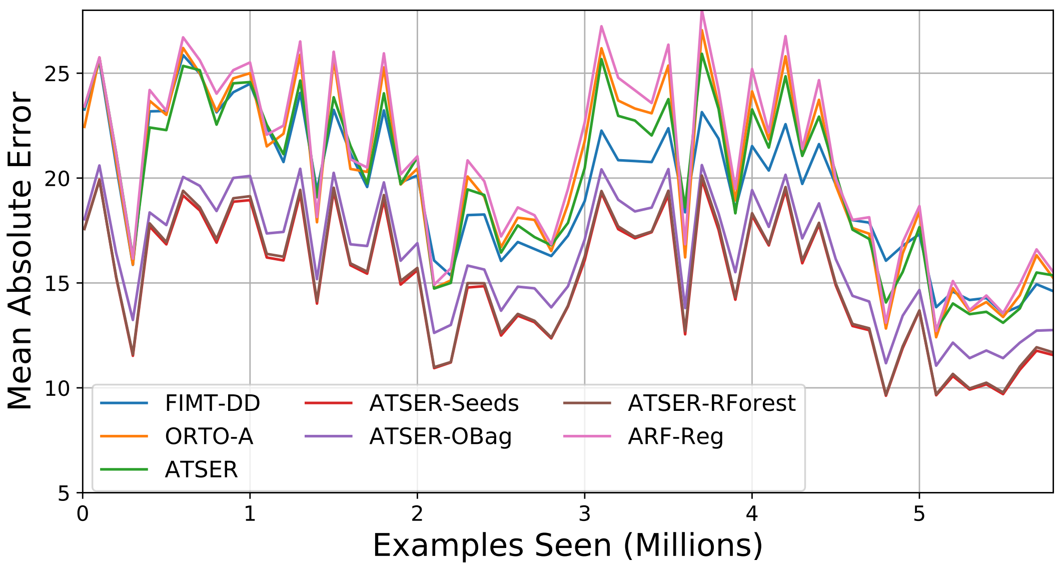


Figure A.1. Performance with no linear models on AIRLINE 08 over 100k windows.

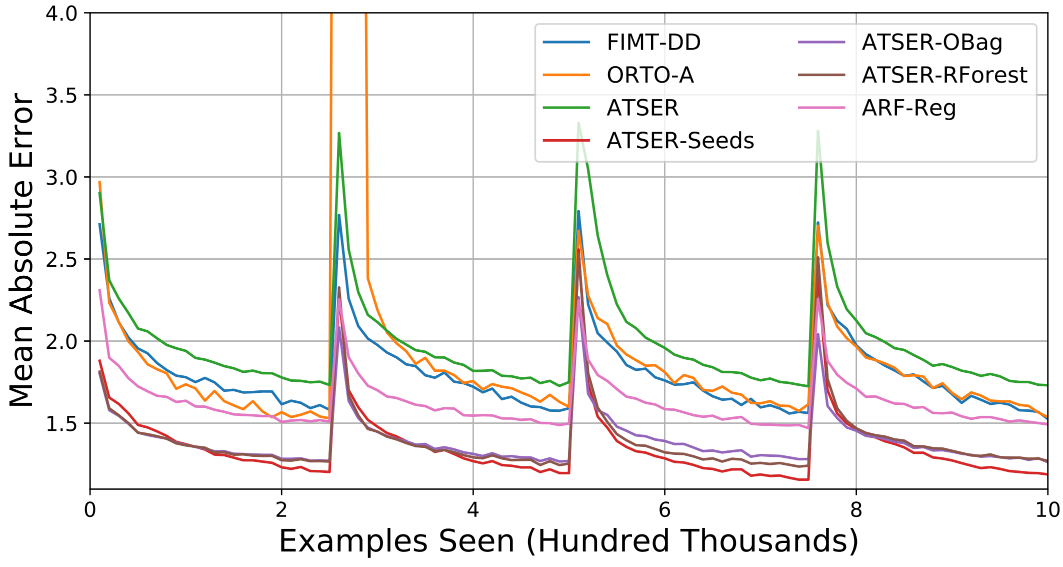


Figure A.2. Performance with no linear models on FRIEDMAN over 10k windows. A section of the ORTO-A results which peaks at 14.5 has been omitted for visibility.

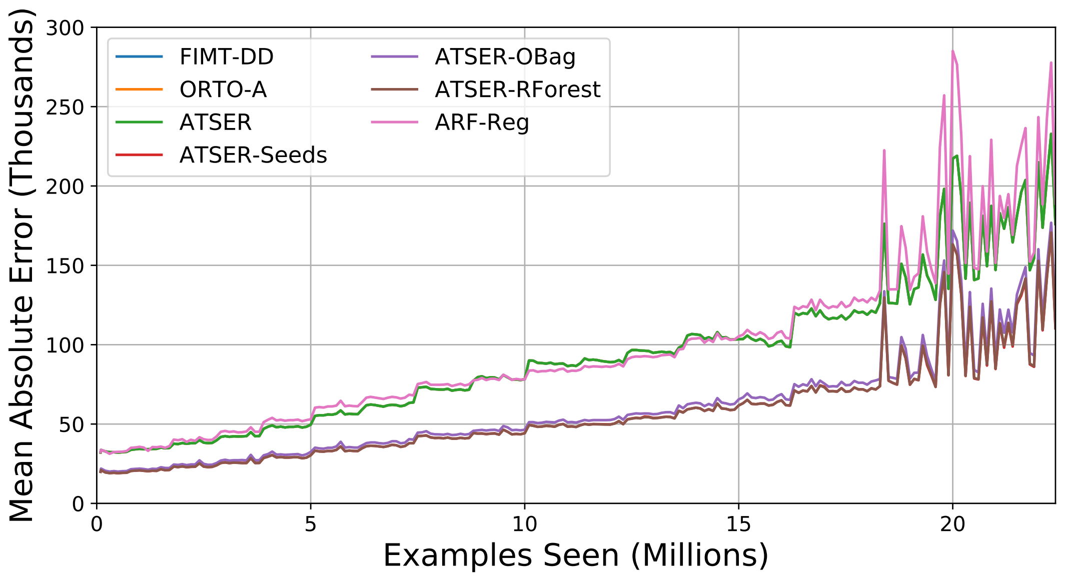


Figure A.3. Performance with no linear models on HOUSING over 100k windows. The FIMT-DD, ORTO-A and ATSER results are identical.

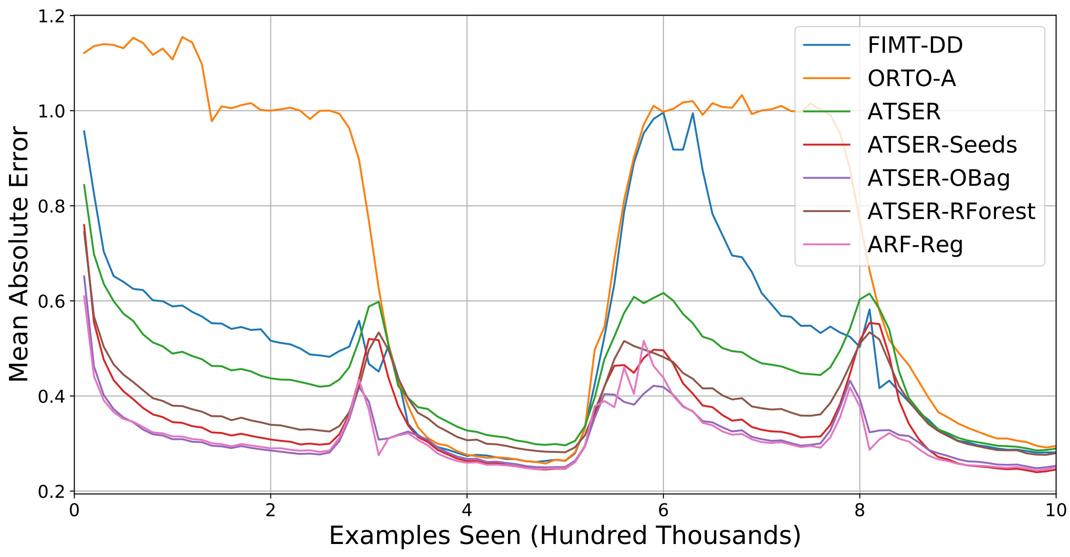


Figure A.4. Performance with no linear models on HYPERPLANE over 10k windows.

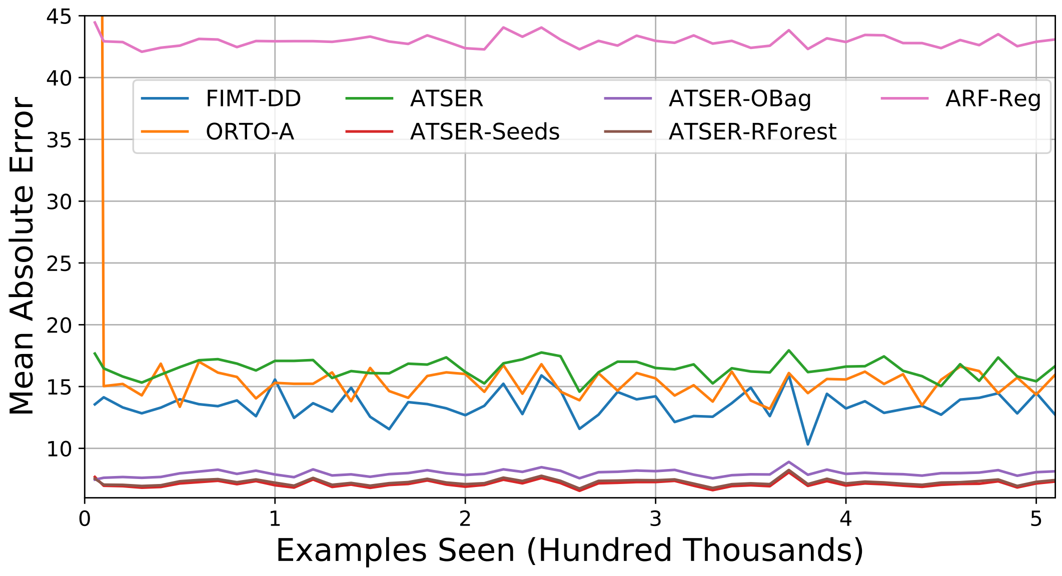


Figure A.5. Performance with no linear models on YEARMSD over 10k windows.

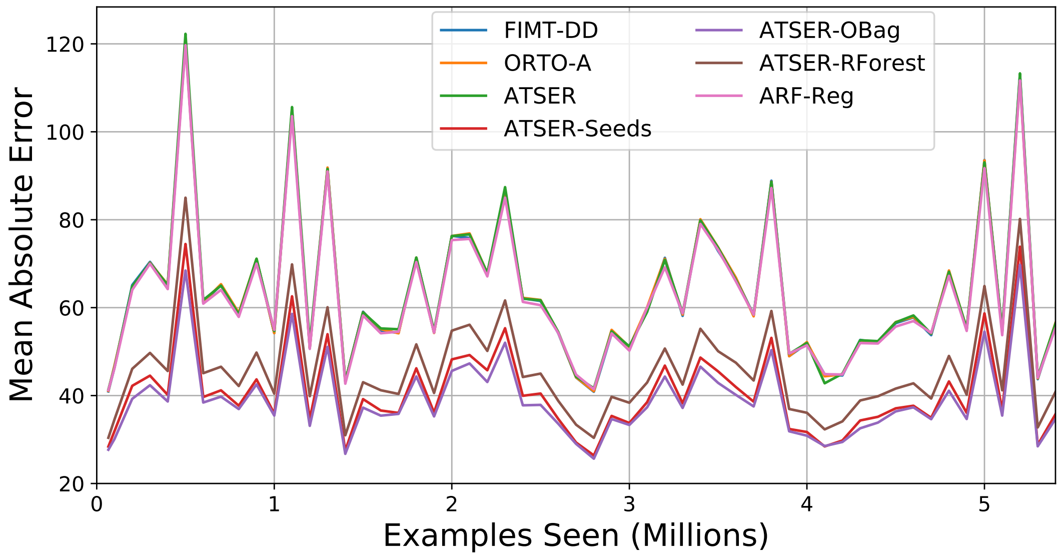


Figure A.6. Performance with no linear models on ZURICH over 100k windows.

A.2 Additional Results for Section 6.5.2: Comparison of MAEs Between Models, with the Pareto Front Shown

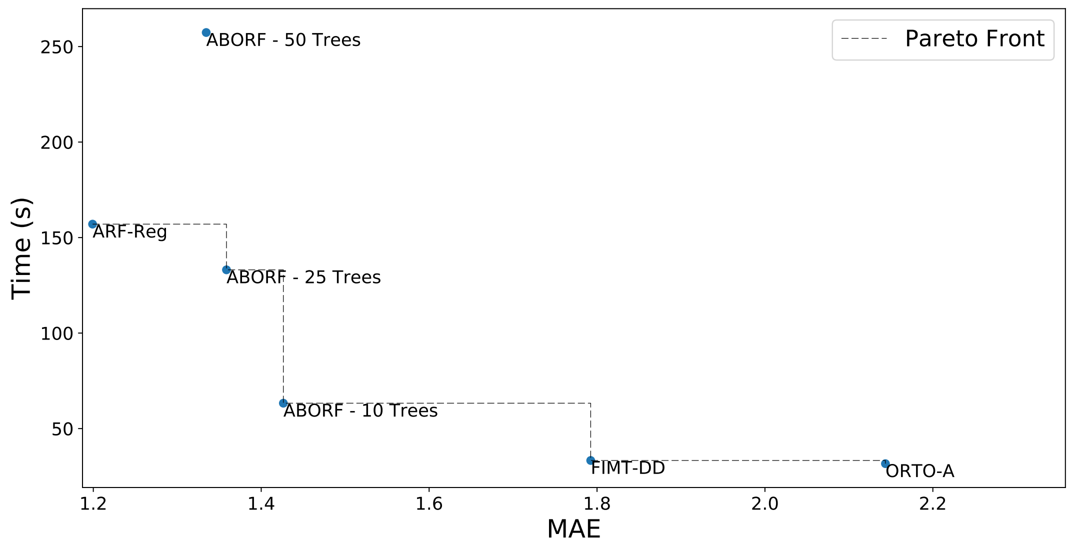


Figure A.7. The effectiveness of different models and the pareto front on the FRIEDMAN dataset.

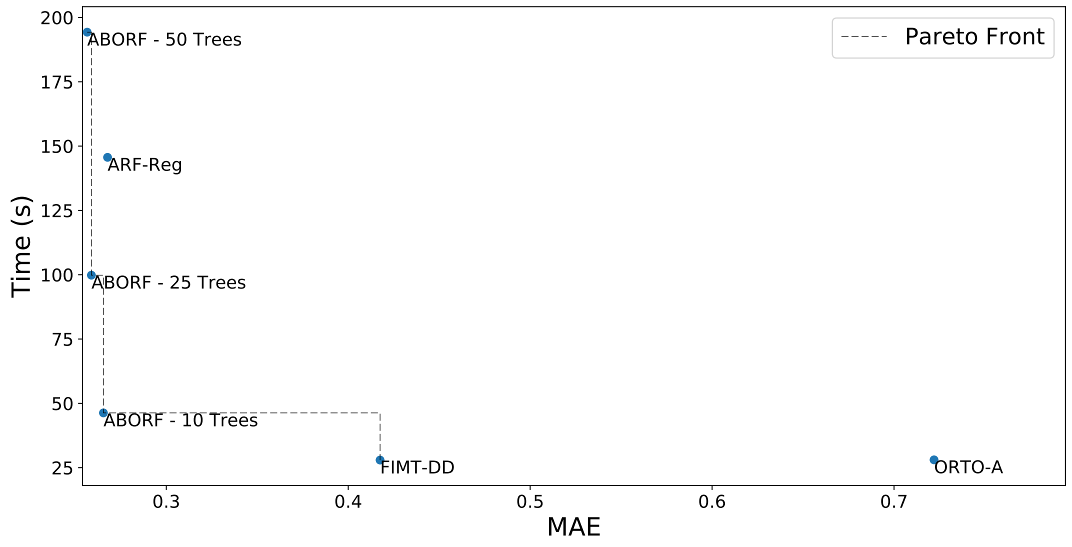


Figure A.8. The effectiveness of different models and the pareto front on the HYPERPLANE dataset.

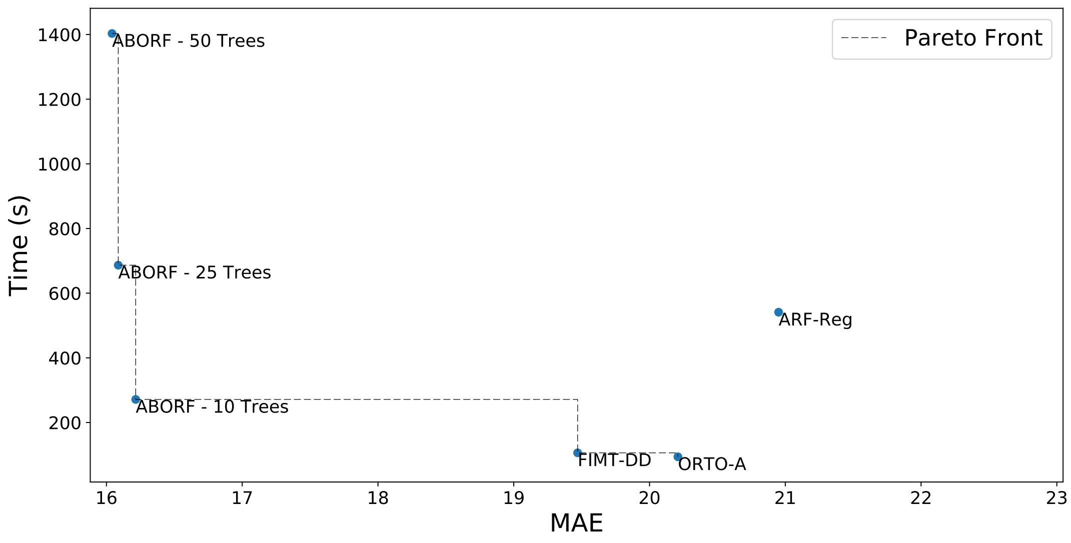


Figure A.9. The effectiveness of different models and the pareto front on the AIRLINE08 dataset.

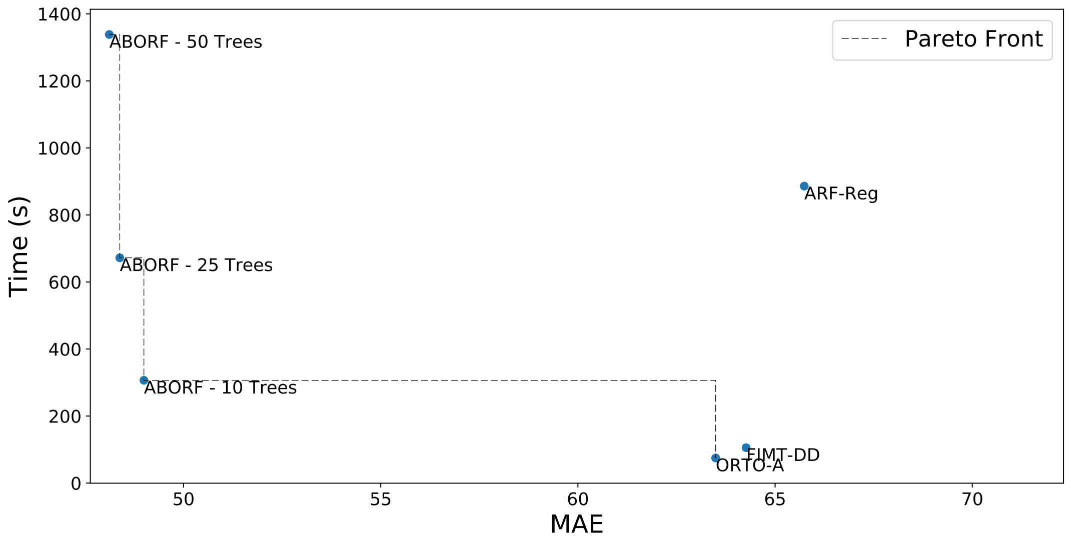


Figure A.10. The effectiveness of different models and the pareto front on the ZURICH dataset.

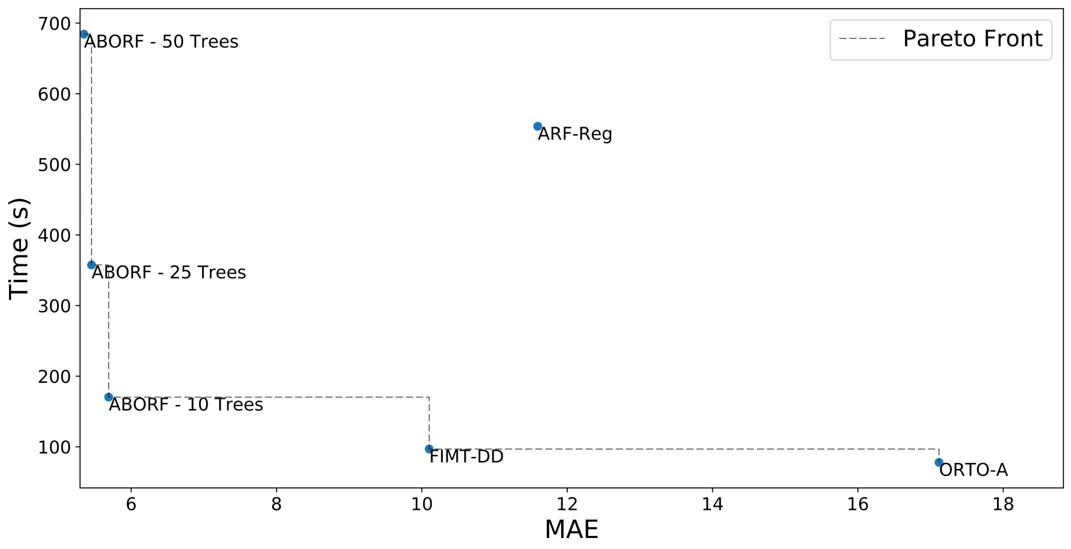


Figure A.11. The effectiveness of different models and the pareto front on the YEARMSD dataset.

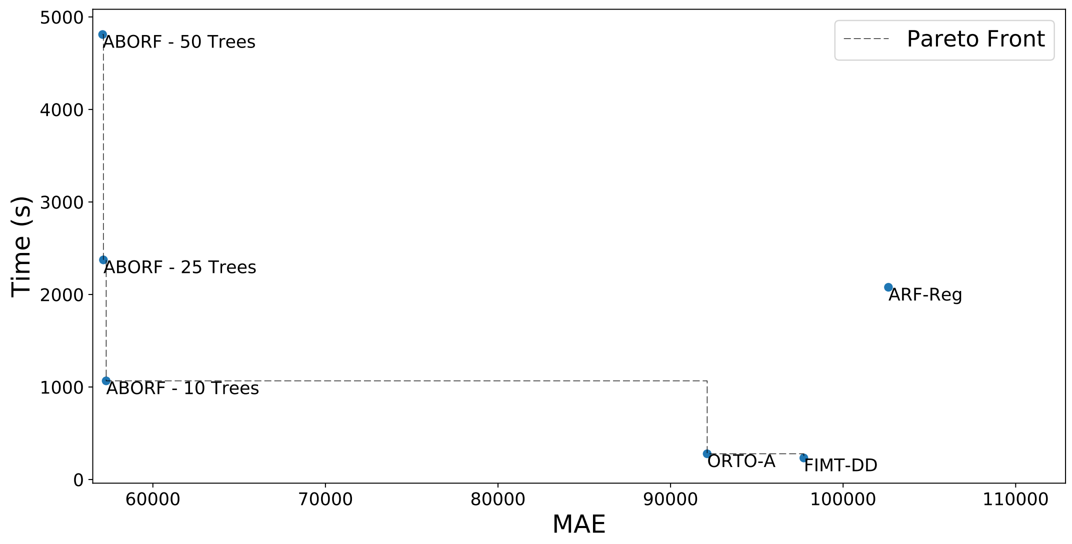


Figure A.12. The effectiveness of different models and the pareto front on the HOUSING dataset.

A.3 Additional Results for Section 6.5.3: MAE Comparison between Axis-Aligned and Multivariate Splits

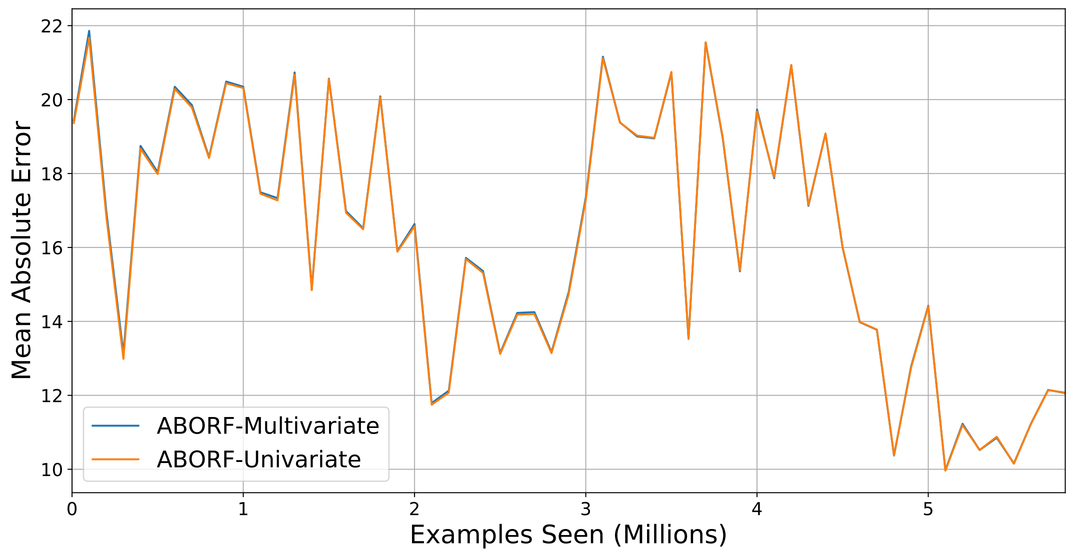


Figure A.13. A comparison between multivariate and univariate splitting on the AIRLINE08 dataset.

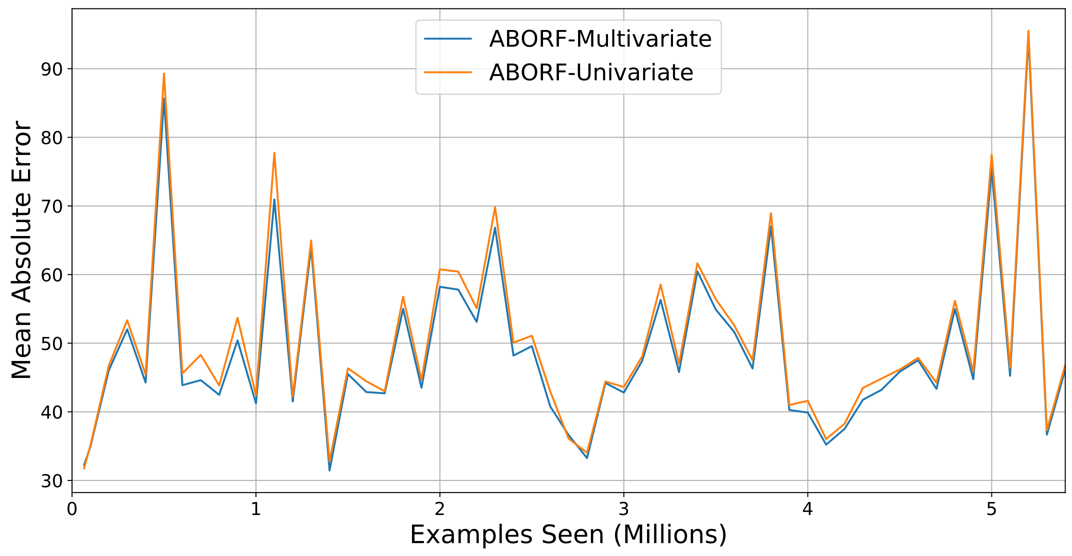


Figure A.14. A comparison between multivariate and univariate splitting on the ZURICH dataset.

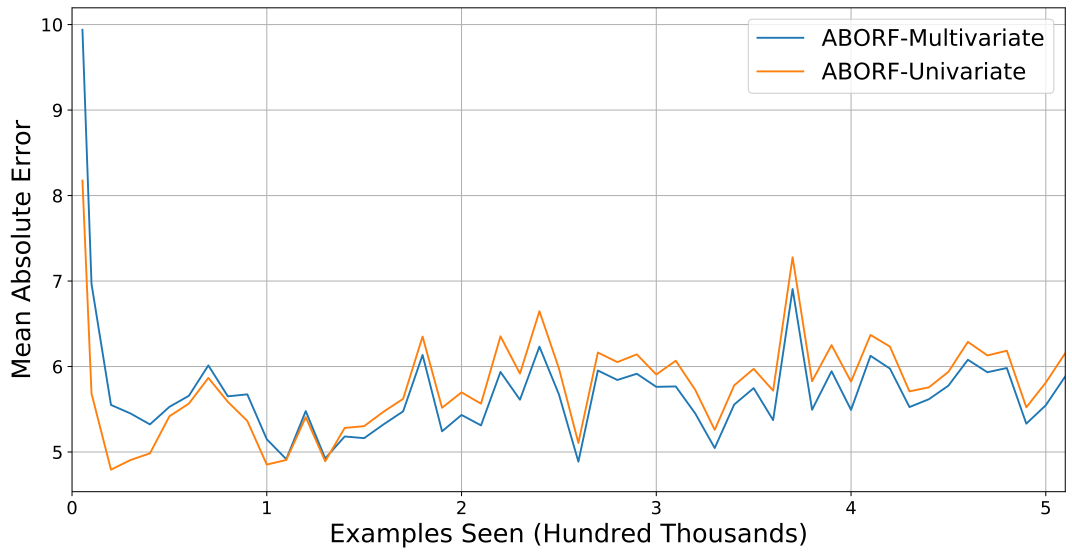


Figure A.15. A comparison between multivariate and univariate splitting on the YEARMSD dataset.

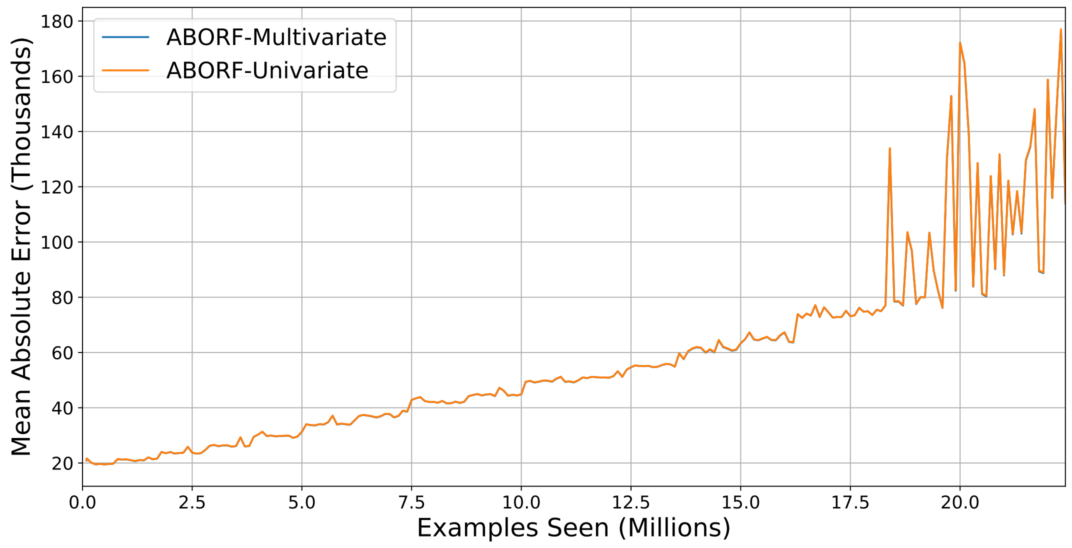


Figure A.16. A comparison between multivariate and univariate splitting on the HOUSING dataset.

A.4 Additional Results for Section 6.5.4: Effect of Ensemble Size on MAE

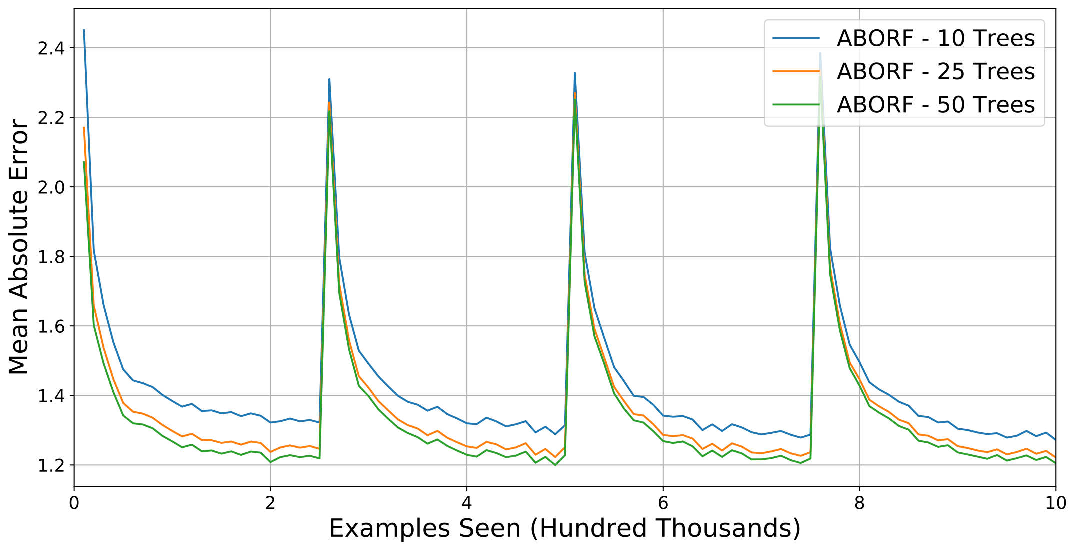


Figure A.17. The effect of ensemble size on the FRIEDMAN dataset.

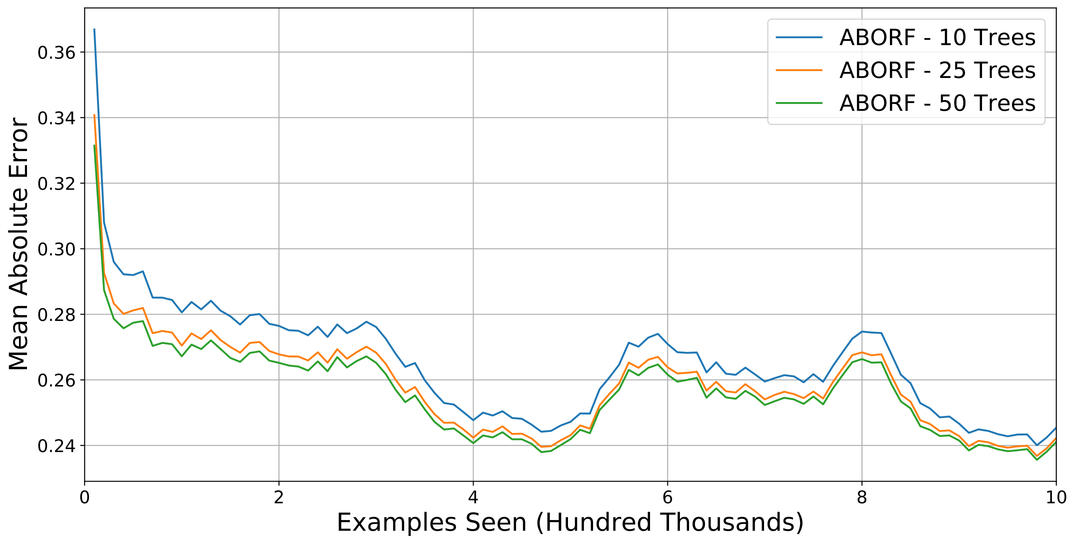


Figure A.18. The effect of ensemble size on the HYPERPLANE dataset.

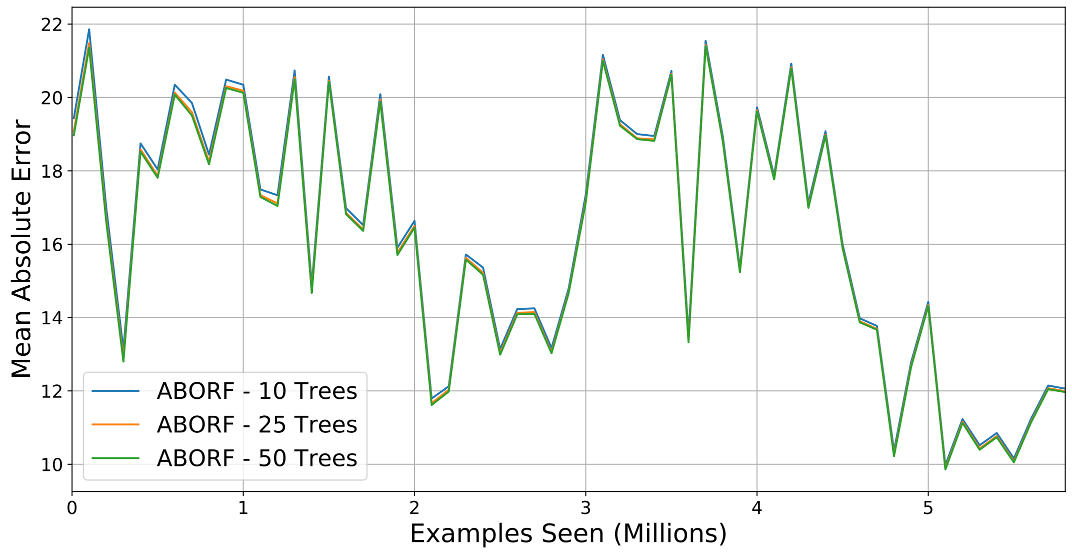


Figure A.19. The effect of ensemble size on the AIRLINE08 dataset.

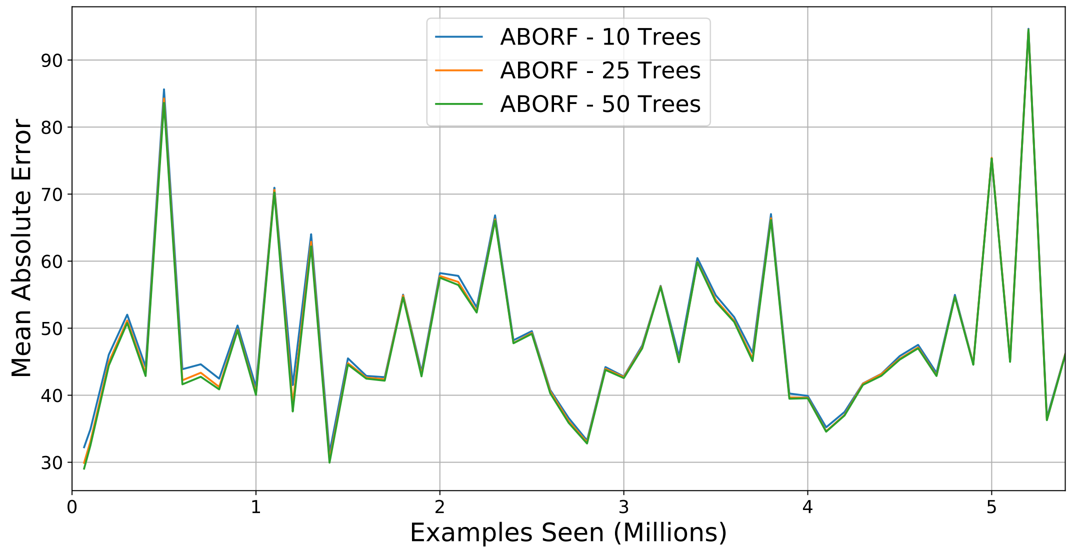


Figure A.20. The effect of ensemble size on the ZURICH dataset.

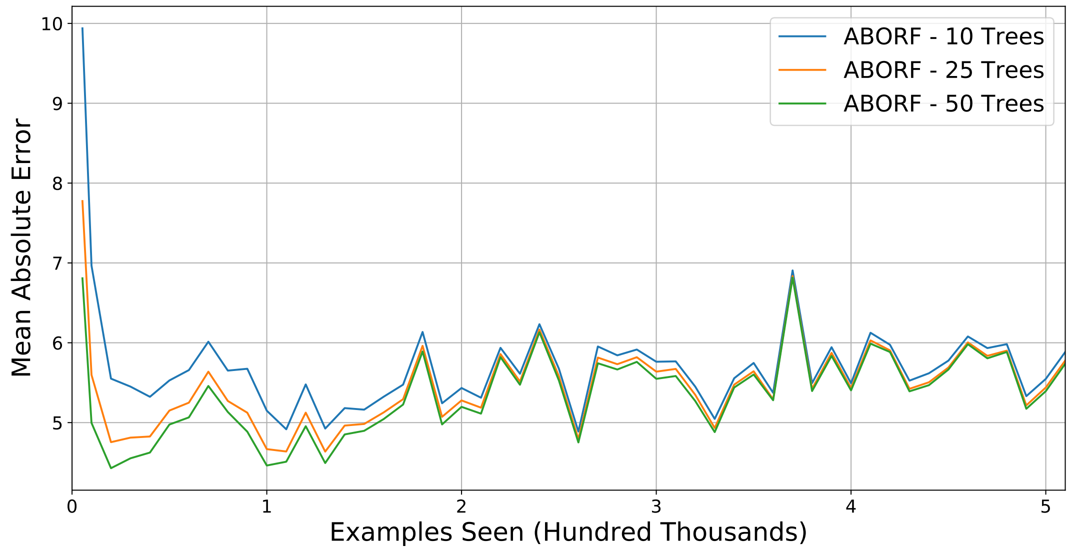


Figure A.21. The effect of ensemble size on the YEARMSD dataset.

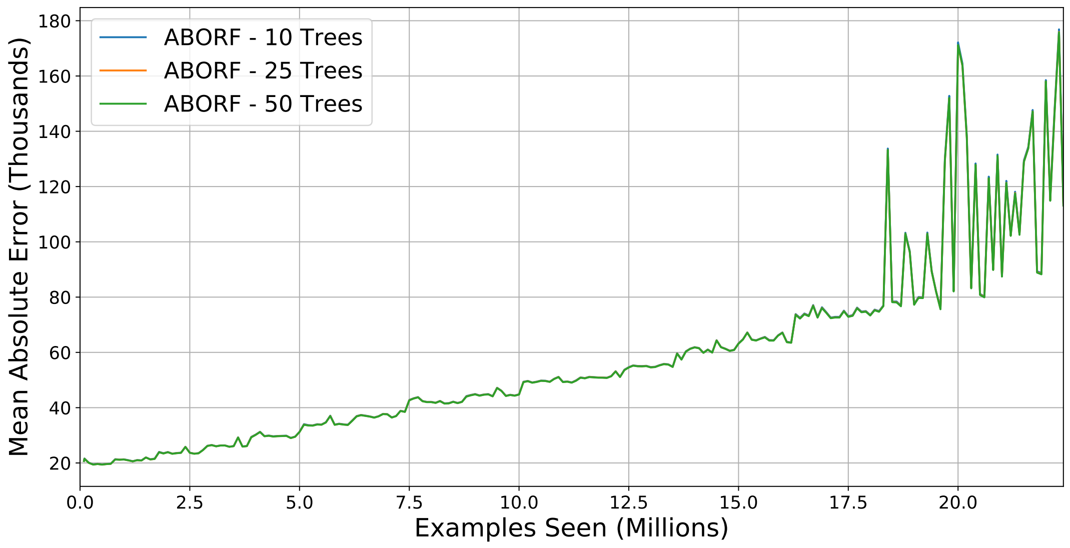


Figure A.22. The effect of ensemble size on the HOUSING dataset.

A.5 Additional Results for Section 6.5.5: MAEs for 50 Tree ATSER and ABORF Ensembles

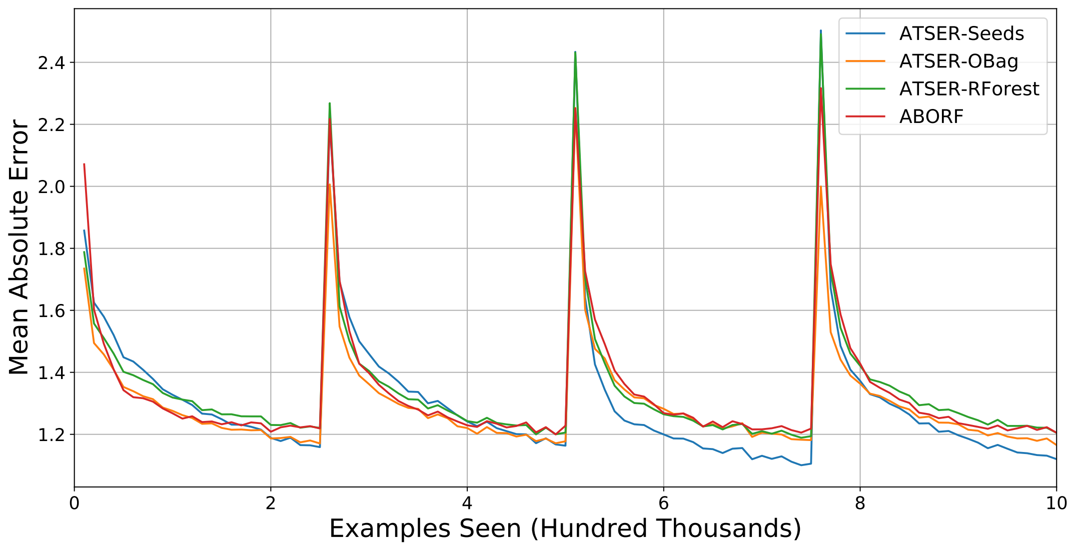


Figure A.23. A comparison between 50 tree ensembles of ATSER and ABORF on the FRIEDMAN dataset.

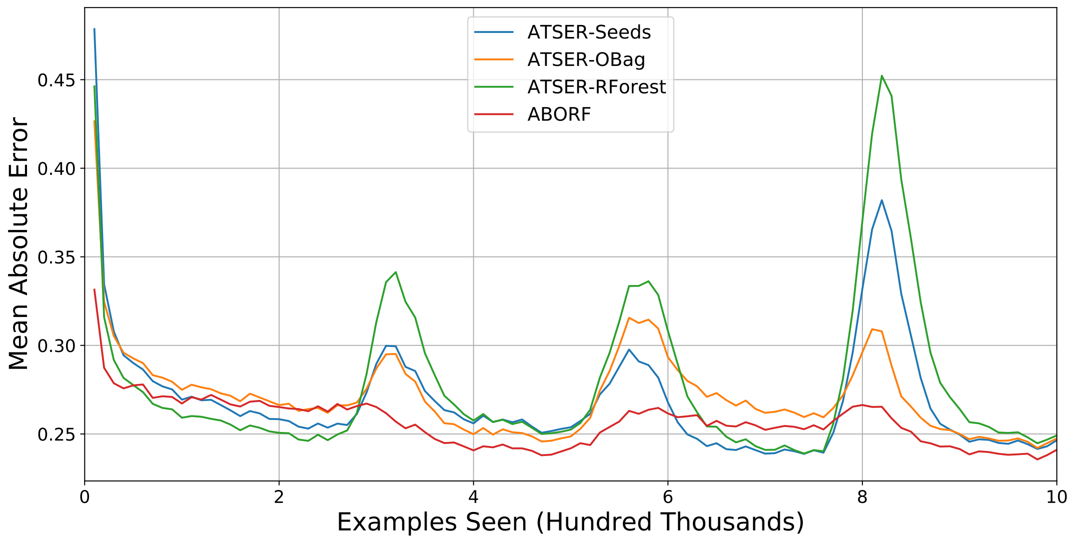


Figure A.24. A comparison between 50 tree ensembles of ATSER and ABORF on the HYPERPLANE dataset.

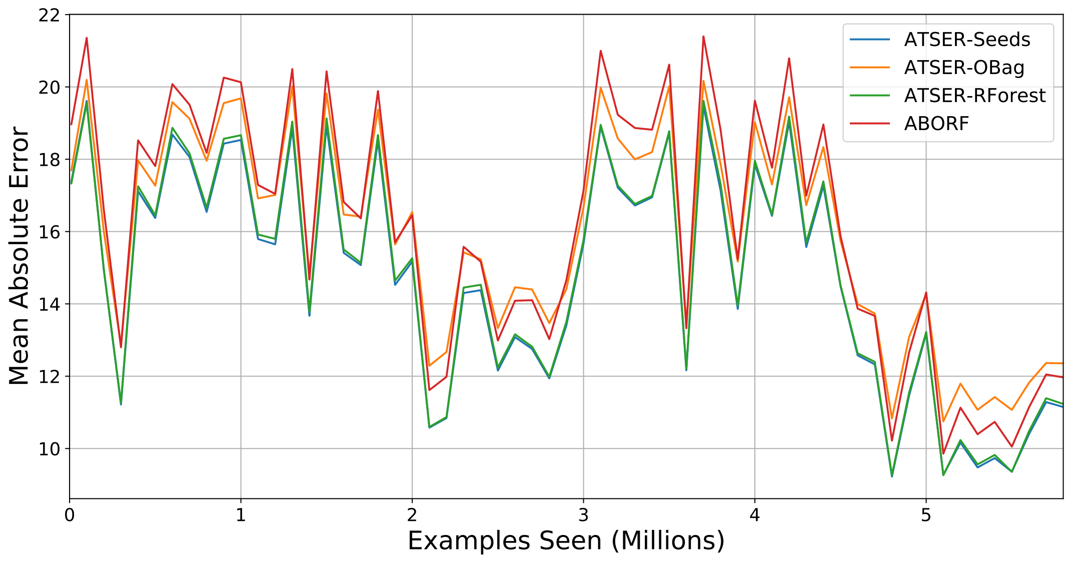


Figure A.25. A comparison between 50 tree ensembles of ATSER and ABORF on the AIRLINE08 dataset.

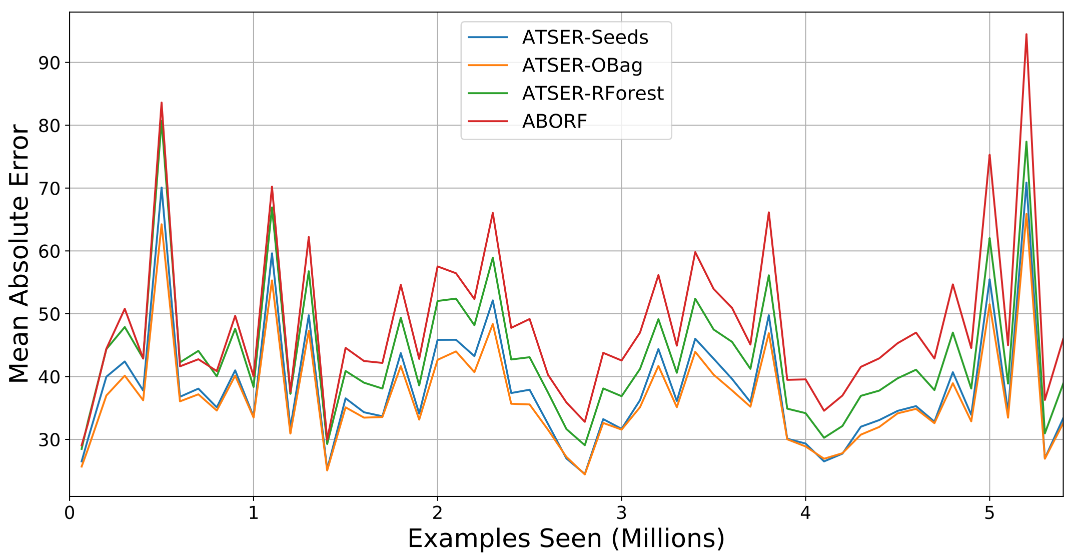


Figure A.26. A comparison between 50 tree ensembles of ATSER and ABORF on the ZURICH dataset.

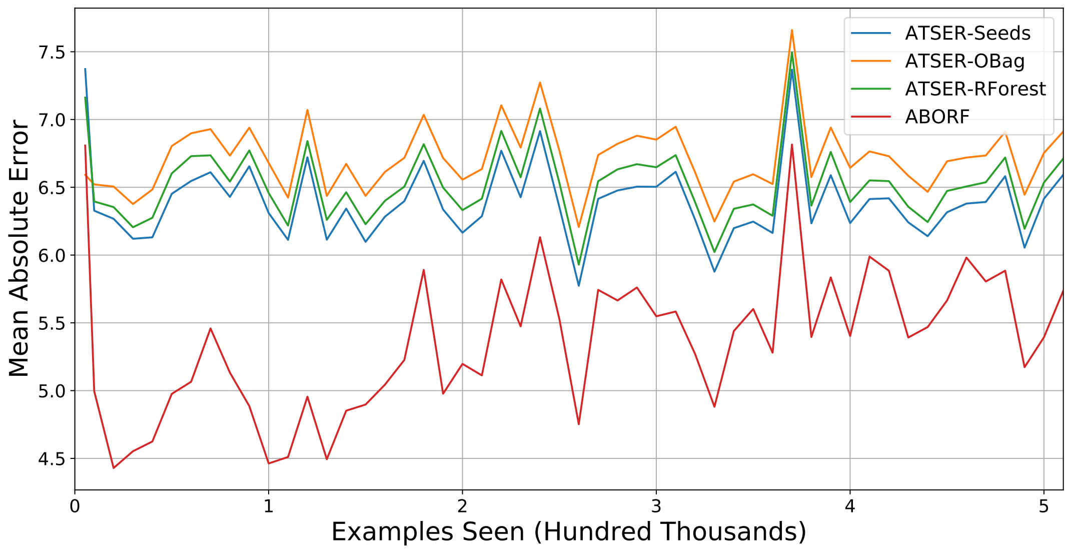


Figure A.27. A comparison between 50 tree ensembles of ATSER and ABORF on the YEARMSD dataset.

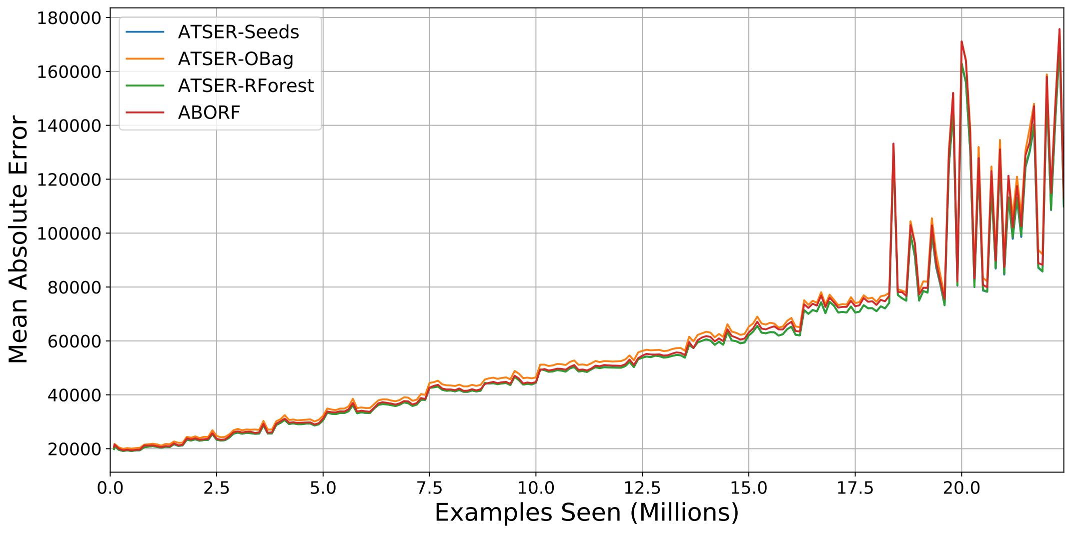


Figure A.28. A comparison between 50 tree ensembles of ATSER and ABORF on the HOUSING dataset.

Bibliography

Abramowitz, M. and Stegun, I. (1974). *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Inc., USA.

Aggarwal, C. C. (2006). *Data Streams: Models and Algorithms*. springer.

Ahmed, E., Clark, A., and Mohay, G. (2008). A novel sliding window based change detection algorithm for asymmetric traffic. *Proceedings - 2008 IFIP International Conference on Network and Parallel Computing, NPC 2008*, pages 168–175.

Akbani, R., Kwek, S., and Japkowicz, N. (2004). Applying Support Vector Machines to Imbalanced Datasets. *Lnai*, 3201:39–50.

Alippi, C., Boracchi, G., and Roveri, M. (2009). Just in time classifiers: Managing the slow drift case. *Proceedings of the International Joint Conference on Neural Networks*, pages 114–120.

Armstrong, J. and Collopy, F. (1992). Error measures for generalizing about forecasting methods: Empirical comparisons. *International Journal of Forecasting*, 8(1):69–80.

Basseville, M. and Nikiforov, I. (1993). *Detection of Abrupt Change Theory and Application*, volume 15. PTR Prentice-Hall.

Bernardo, J. and Smith, A. (2000). *Bayesian Theory*, volume 15. Wiley.

Bickel, S., Brückner, M., and Scheffer, T. (2007). Discriminative learning for differing training and test distributions. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 81–88, New York, New York, USA. ACM Press.

Bickel, S., Brückner, M., and Scheffer, T. (2009). Discriminative Learning Under Covariate Shift. *Journal of Machine Learning Research*, 10:2137—-2155.

Bifet, A., Frank, E., Holmes, G., and Pfahringer, B. (2012). Ensembles of Restricted Hoeffding Trees. *ACM Transactions on Intelligent Systems and Technology*, 3(2).

Bifet, A. and Gavald, R. (2009). Adaptive Parameter-free Learning from Evolving Data Streams. *Advances in Intelligent Data Analysis VIII*, pages 249–260.

Bifet, A., Gavalda, R., and Gavaldà, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. *Sdm*, 7:2007.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010a). MOA: Massive Online Analysis. *Journal of Machine Learning Research*, 11:1601–1604.

Bifet, A., Holmes, G., and Pfahringer, B. (2010b). Leveraging Bagging for Evolving Data Streams. *ECML PKDD*, 6321:135–150.

Bifet, A., Holmes, G., Pfahringer, B., and Gavaldà, R. (2009a). Improving adaptive bagging methods for evolving data streams. In Zhou, Z.-H. and Washio, T., editors, *Advances in Machine Learning*, pages 23–37, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., and Gavaldà, R. (2009b). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 139–148, New York, NY, USA. Association for Computing Machinery.

Bifet, A., Morales-bueno, R., Baena-Garcia, M., Campo-Avila, J. D., Fidalgo, R., Bifet, A., Gavalda, R., and Morales-bueno, R. (2006). Early Drift Detection Method. *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, 6:77–86.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blackard, J. A. and Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, vol.24:131–151. Data available from `https://archive.ics.uci.edu/ml/datasets/Covertype`.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45:5–32.

Brodley, C. E. and Utgoff, p. E. (1995). Multivariate Decision Trees. *Machine Learning*, 19:45–77.

Brzeziński, D. and Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6679 LNAI(PART 2):155–163.

Brzezinski, D. and Stefanowski, J. (2014a). Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, 265:50–67.

Brzezinski, D. and Stefanowski, J. (2014b). Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):81–94.

Chipman, H. A., George, E. I., and McCulloch, R. E. (1998). Bayesian cart model search. *Journal of the American Statistical Association*, 93(443):935–948.

Chipman, H. A., George, E. I., and McCulloch, R. E. (2006). Bayesian Ensemble Learning. In *Advances in Neural Information Processing Systems*.

Coppersmith, D. and Winograd, S. (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280. Computational algebraic complexity editorial.

Cortes, C., Mohri, M., Riley, M., and Rostamizadeh, A. (2008). Sample Selection Bias Correction Theory. *International Conference on Algorithmic Learning Theory*, pages 38–53.

Dasgupta, S. and Freund, Y. (2008). Random projection trees and low dimensional manifolds. In *STOC '08: Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546.

Dasu, T., Krishnan, S., Venkatasubramanian, S., and Yi, K. (2006). An information-theoretic approach to detecting changes in multi-dimensional data streams. In *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*.

Dawid, A. (1984). Statistical Theory : The Prequential Approach Author. *Journal of the Royal Statistical Society*, 147(2):278–292.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. In *Multiple Classifier Systems. MCS 2000*, pages 1–15.

Domingos, P. and Hulten, G. (2000). Mining High-Speed Data Streams. *Proceedings of The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 6(1):71–80.

Duarte, J. a., Gama, J. a., and Bifet, A. (2016). Adaptive model rules from high-speed data streams. *ACM Transanctions on Knowledge Discovery from Data*, 10(3):1–22.

Elwell, R. and Polikar, R. (2009). Incremental learning in nonstationary environments with controlled forgetting. In *2009 International Joint Conference on Neural Networks*, pages 771–778. IEEE.

Elwell, R. and Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 22(10):1517–31.

Estabrooks, A., Jo, T., and Japkowicz, N. (2004). A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20(1):18–36.

Faria, E. R., Gama, J., and Carvalho, A. C. P. L. F. (2013). Novelty Detection Algorithm for Data Streams Multi-Class Problems. In *SAC '13: Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 795–800.

Friedman, J. H. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1 – 67.

Frías-Blanco, I., Campo-Ávila, J. d., Ramos-Jiménez, G., Morales-Bueno, R., Ortiz-Díaz, A., and Caballero-Mota, Y. (2015). Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823.

Gama, J. and Castillo, G. (2006). Learning with local drift detection. In Li, X., Zaïane, O. R., and Li, Z., editors, *Advanced Data Mining and Applications*, pages 42–55, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. In Bazzan, A. L. C. and Labidi, S., editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, 15(1):329.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, 90:317–346.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Gomes, H. M., Barddal, J. P., Boiko, L. E., and Bifet, A. (2018). Adaptive random forests for data stream regression. *ESANN 2018 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 1(1):25–27.

Gomes, H. M., Barddal, J. P., Enembreck, F., and Bifet, A. (2017a). A Survey on Ensemble Learning for Data Stream Classification. *ACM Computing Surveys*, 50(2).

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., and Abdessalem, T. (2017b). Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495.

Gomes, H. M., Read, J., Bifet, A., Barddal, J. P., and Gama, J. (2019). Machine learning for streaming data : state of the art , challenges , and opportunities. *SIGKDD Explorations*, 21(1):6–22.

Gouk, H., Pfahringer, B., and Frank, E. (2019). Stochastic gradient trees. In Lee, W. S. and Suzuki, T., editors, *Proceedings of The Eleventh Asian Conference on Machine Learning*, volume 101 of *Proceedings of Machine Learning Research*, pages 1094–1109, Nagoya, Japan. PMLR.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. J. (2006). A kernel method for the two-sample-problem. *Advances in Neural Information Processing Systems*, 19(157):0–43.

Harel, M., Crammer, K., El-Yaniv, R., and Mannor, S. (2014). Concept Drift Detection Through Resampling. *International Conference on Machine Learning (ICML)*, pages 1009–1017.

Heckman, J. (1979). Sample Specification Bias as a Selection Error. *Econometrica*, 47(1):153–162.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning lecture 6a.

Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30.

Hoens, T. R., Polikar, R., and Chawla, N. V. (2012). Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 1(1):89–101.

Holmes, G., Kirkby, R., and Pfahringer, B. (2005). Stress-testing Hoeffding trees. In Jorge, A. M., Torgo, L., Brazdil, P., Camacho, R., and Gama, J., editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 495–502, Berlin, Heidelberg. Springer Berlin Heidelberg.

Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2006a). Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501.

Huang, J., Smola, A. J., Gretton, A., Borgwardt, K. M., and Scholkopf, B. (2006b). Correcting sample selection bias by unlabeled data. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, page 601–608, Cambridge, MA, USA. MIT Press.

Huerta, R., Mosqueiro, T., Fonollosa, J., Rulkov, N., and Rodriguez-Lujan, I. (2016). Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157. Data available from `https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring`.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 97–106.

Ikonomovska, E., Gama, J., and Džeroski, S. (2011a). Incremental multi-target model trees for data streams. In *SAC*, number 11 in Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11, pages 988–993.

Ikonomovska, E., Gama, J., and Džeroski, S. (2011b). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168.

Ikonomovska, E., Gama, J., and Džeroski, S. (2015). Online tree-based ensembles and option trees for regression on evolving data streams. *Neurocomputing*, 150:458–470.

Ikonomovska, E., Gama, J., Zenko, B., and Dzeroski, S. (2011c). Speeding-up Hoeffding-based regression trees with options. In *ICML*, number 28 in Proceedings of the 28th International Conference on Machine Learning, pages 537–544.

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449.

Kaggle Competition (2012). Give Me Some Credit - Historic credit data from 150,000 borrowers. Data available from `https://www.kaggle.com/c/GiveMeSomeCredit/`.

Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., and Khaled, G. (2018). Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, 9(1):1–23.

Kifer, D., Ben-david, S., and Gehrke, J. (2004). Detecting Change in Data Streams. *Proceedings of the 30th VLDB Conference*, pages 180–191.

Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8:281–300.

Klinkenberg, R. and Joachims, T. (2000). Detecting Concept Drift with Support Vector Machines. In *International Conference on Machine Learning*, pages 487–494.

Kohavi, R. and Kunz, C. (1997). Option decision trees with majority votes. *In Proc. 14th International Conference on Machine Learning*, pages(1996):161–169.

Kolter, J. and Maloof, M. (2003). Dynamic weighted majority: a new ensemble method for tracking concept drift. *Third IEEE International Conference on Data Mining*, pages 123–130.

Koychev, I. (2000). Gradual forgetting for adaptation to concept drift. *Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*, pages 101–106.

Krause, O. and Igel, C. (2015). A more efficient rank-one covariance matrix update for evolution strategies. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*, FOGA '15, page 129–136, New York, NY, USA. Association for Computing Machinery.

Krawczyk, B., Minku, L. L., and Stefanowski, J. (2017). Ensemble learning for data stream analysis : A survey. *Information Fusion*, 37:132–156.

Kull, M. and Flach, P. (2014). Patterns of dataset shift. *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD*.

Kuncheva, L. (2004). Classifier ensembles for changing environments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3077:1–15.

Kuncheva, L. I. and Žliobait, I. (2009). On the window size for classification in changing environments. *Intelligent Data Analysis*, 13(6):861–872.

Lakshminarayanan, B., Roy, D., and Teh, Y. W. (2013). Top-down particle filtering for bayesian decision trees. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 280–288, Atlanta, Georgia, USA.

Latinne, P., Saerens, M., and Decaestecker, C. (2001). Adjusting the outputs of a classifier to new a priori probabilities may significantly improve classification accuracy: evidence from a multi-class problem in remote sensing. *Proceedings of the 18th International Conference on Machine Learning (ICML 2001)*, pages 298–305.

Lazarescu, M., Venkatesh, S., and Bui, H. (2004). Using multiple windows to track concept drift. *Intelligent Data Analysis*, 8(1):1–28.

Littlestone, N. and Warmuth, M. (1994). The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261.

Lockhart, J. W., Weiss, G. M., Xue, J. C., Gallagher, S. T., Grosner, A. B., and Pulickal, T. T. (2011). Design considerations for the wisdm smart phone-based sensor mining architecture. In *Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '11, page 25–33, New York, NY, USA. Association for Computing Machinery. Data available from `http://www.cis.fordham.edu/wisdm/dataset.php`.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2019). Learning under Concept Drift : A Review. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 31(12):2346–2363.

Maloof, M. a. and Michalski, R. S. (1995). Learning evolving concepts using a partial memory approach. *Working Notes of the 1995 AAAI Fall Symposium on Active Learning*, pages 70–73.

Manapragada, C., Webb, G. I., and Salehi, M. (2018). Extremely Fast Decision Tree. In *KDD '18: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1953—-1962.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Minku, L. L., White, A. P., and Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742.

Minku, L. L. and Yao, X. (2012). Ddd: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):619–633.

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530.

Mouss, H., Mouss, D., Mouss, N., and Sefouhi, L. (2004). Test of Page-Hinckley, an approach for fault detection in an agro-alimentary production system. *Proceedings of the 5th Asian Control Conference*, 5(1):815–818.

Narasimhamurthy, a. and Kuncheva, L. (2007). A framework for generating data to simulate changing environments. *Proceedings of the 25th IASTED International Multi-Conference Artificial Intelligence and Applications*, pages 384–389.

OpenML (1999). KDD Cup 1999 - Network connection data. Data available from `https://www.openml.org/d/1110`.

OpenML (2007). Pokerhand - Normalised data on different poker hands. Data available from `https://www.openml.org/d/155`.

Osojnik, A., Panov, P., and Džeroski, S. (2016). Comparison of Tree-Based Methods for Multi-target Regression on Data Streams. In *New Frontiers in Mining Complex Patterns. NFMCP 2015*, volume 9607 of *Lecture Notes in Computer Science*, pages 17–31, Switzerland. Springer, Cham.

Osojnik, A., Panov, P., and Džeroski, S. (2017). Multi-label classification via multi-target regression on data streams. *Machine Learning*, 106(6):745–770.

Osojnik, A., Panov, P., and Džeroski, S. (2018). Tree-based methods for online multi-target regression. *Journal of Intelligent Information Systems*, 50(2):315–339.

Oza, N. C. and Russell, S. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, volume 136, pages 359–364, New York, New York, USA. ACM Press.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

Pfahringer, B., Holmes, G., and Kirkby, R. (2007). New Options for Hoeffding Trees. In *AI 2007: Advances in Artificial Intelligence*, pages 90–99. Springer Berlin Heidelberg, Berlin, Heidelberg.

Pham, X., Dang Manh, T., Sang, D., Hoang, S., Nguyen, T., and Liew, A. W.-C. (2017). Learning from data stream based on random projection and hoeffding tree classifier. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8.

Quadrianto, N. and Ghahramani, Z. (2015). A Very Simple Safe-Bayesian Random Forest. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1297–1303.

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset Shift in Machine Learning*. The MIT Press.

Reiss, A. and Stricker, D. (2012). Introducing a new benchmarked dataset for activity monitoring. In *2012 16th International Symposium on Wearable Computers*, pages 108–109. Data available from `https://archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring`.

Ross, G. J., Adams, N. M., Tasoulis, D. K., and Hand, D. J. (2012). Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198.

Rutkowski, L., Pietruczuk, L., Duda, P., and Jaworski, M. (2013). Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279.

Schetinin, V., Fieldsend, J. E., Partridge, D., Coats, T. J., Krzanowski, W. J., Everson, R. M., Bailey, T. C., and Hernandez, A. (2007). Confident interpretation of bayesian decision tree ensembles for clinical applications. *IEEE Transactions on Information Technology in Biomedicine*, 11(3):312–319.

Schlimmer, J. C., Granger Jr, R. H., and Granger, R. H. (1986). Incremental learning from noisy data. *Machine Learning*, 1(3):317–354.

Sebastião, R. and Gama, J. (2007). Change detection in learning histograms from data streams. In Neves, J., Santos, M. F., and Machado, J. M., editors, *Progress in Artificial Intelligence*, pages 112–123, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sethi, T. S. and Kantardzic, M. (2017). On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244.

Soares, S. G. and Araújo, R. (2015). An on-line weighted ensemble of regressor models to handle concept drifts. *Engineering Applications of Artificial Intelligence*, 37:392–406.

Spinosa, E. J., Carvalho, A. C. P. L. F., and Gama, J. (2007). OLINDDA : A cluster-based approach for detecting novelty and concept drift in data streams. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 448–452.

Stisen, A., Blunck, H., Bhattacharya, S., and Prentow, T. S. (2015). Smart Devices are Different : Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition. In *SenSys '15: Proceedings of the 13th ACM Conference on Embedded Networked Sensor SystemsNovember 2015*, pages 127–140. Data available from `https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition`.

Storkey, A. (2008). When Training and Test Sets Are Different: Characterizing Learning Transfer. In *Dataset Shift in Machine Learning*, pages 2–28. The MIT Press.

Storkey, A. J. and Sugiyama, M. (2007). Mixture Regression for Covariate Shift. *Advances in Neural Information Processing Systems 19*, pages 1337–1344.

Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, 4:377–382.

Sugiyama, M., Krauledat, M., and Muller, K.-R. (2007). Covariate Shift Adaptation by Importance Weighted Cross Validation. *Journal of Machine Learning Research*, 8:985–1005.

Sugiyama, M., Suzuki, T., Nakajima, S., Kashima, H., Von Bünau, P., and Kawanabe, M. (2008). Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics*, 60(4):699–746.

Tsuboi, Y., Kashima, H., Hido, S., Bickel, S., and Sugiyama, M. (2009). Direct Density Ratio Estimation for Large-scale Covariate Shift Adaptation. *Journal of Information Processing*, 17(Cv):138–155.

Tsymbal, A. (2004). The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 4(C):2004–15.

Vella, F. (1998). Estimating Models with Sample Selection Bias: A Survey. *The Journal of Human Resources*, 33(1):127.

Wang, H., Fan, W., Yu, P., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2(1):226–235.

Wang, S., Minku, L. L., Ghezzi, D., Caltabiano, D., Tino, P., and Yao, X. (2013). Concept drift detection for online class imbalance learning. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10.

Wang, S., Minku, L. L., and Yao, X. (2018). A Systematic Study of Online Class Imbalance Learning with Concept Drift. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):4802–4821.

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., and Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994.

Widmer, G. and Kubat, M. (1992). Learning Flexible Concepts from Streams of Examples : FLORA2. In *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, pages 463–467.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(505):241–259.

Xu, S. and Wang, J. (2017). Dynamic extreme learning machine for data stream classification. *Neurocomputing*, 238:433–449.

Zadrozny, B. (2004). Learning and evaluating classifiers under sample selection bias. *Twenty-first international conference on Machine learning - ICML '04*, pages 903–910.

Zhang, K., Muandet, K., Wang, Z., and Others (2013). Domain adaptation under target and conditional shift. *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 28:819–827.

Zhang, P., Zhu, X., and Shi, Y. (2008). Categorizing and mining concept drifting data streams. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 812–820, New York, NY, USA. Association for Computing Machinery.

Žliobait, I. E. (2010). Learning under Concept Drift: an Overview. *Training*, abs/1010.4:1–36.