School of Computer Science
The University of Auckland
New Zealand

Concept Drift in Medical Referrals Triage

Hamish Huggard

July 2020

Supervisors:

Yun Sing Koh

Gillian Dobbie

A thesis submitted in partial fulfillment of the requirements of
Master of Science Degree

Abstract

In many machine learning applications, the relationship being modelled may change over time, a phenomenon called concept drift. Most existing approaches to concept drift have assumed an artificially narrow specification of the problem. In this thesis we explore some new approaches to concept drift which introduce several new "tricks" which help bridge the gap between academic concept drift detection and real data science applications. As a motivating example for our investigation, we consider a medical clinic where a decision support system is helping clinicians triage patients referred by GPs. As the triage decision making process evolves, the decision support system should be able to detect that its model has become outdated, and signal to a data scientist that it requires retraining.

We first introduce the Multiple Drift Detector framework, in which a detector for several different types of concept drift is constructed out of a single "narrow" drift detector. This allows for a more complete and interpretable monitoring of concept drift. Changes in the accuracy, precision, recall, label distribution, or instance distribution can be detected, whereas typically only changes in accuracy are detected by existing drift detectors. We also present a graphical interface for MDD to assist understanding of how a data stream is evolving.

Next, we introduce the calibrated drift detection method (CDDM), an algorithm which makes use of probabilistic predictions of models to detect increases in the reducible error, and not the irreducible error, of a model. Both of these are detected by conventional drift detectors, which can result in unnecessary and expensive model retraining.

Next, we present Bayesian drift detection method (BDDM), an algorithm which computes exact posterior probability distributions over possible drift locations and over the error rate of the model. This allows decisions about whether to retrain a model to be made on a rational, expected utility basis. We also introduce Bayes with adaptive forgetfulness (BWAF), which is a heuristic approximation of BDDM.

Finally, we experimentally validate our novel drift detection methods. We demonstrate the circumstances under which CDDM is useful. We also demonstrate that BWAF is competitive on most metrics on standard benchmarks. Finally, we show BWAF is competitive on a synthetic medical triage data stream.

Acknowledgements

I would like to acknowledge and thank the following people.

The Precision Driven Health Partnership for funding this work. Special thanks are due to **Kelly Atkinson** for being my point of contact and support.

Edmond Zhang for introducing me to this project and mentoring me in the field of medical data science.

Yun Sing Koh and **Gillian Dobbie** for being generally swell thesis advisors, as well as encouraging me to submit a paper to SIGIR, and putting just enough pressure on me so that I actually got work done without being overwhelmed by guilt and anxiety.

My past self, for not procrastinating too much. Although he didn't remotely pull his weight, he did at least get the ball rolling, and he did do some good work on side-projects.

List of Publications

Parts of Chapter 4 have been published in:

- Hamish Huggard, Yun Sing Koh, Gillian Dobbie, and Edmond Zhang. Detecting Concept Drift in Medical Triage. In Proceedings of Special Interest Groups of Information Retrieval (SIGIR), 2020.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In a machine learning application, a change in the data distribution is known as concept drift. Adapting to concept drift is one of the core problems in data stream learning [41]. Over the last few decades, considerable research has been dedicated to detecting and adapting to concept drift [24][10]. Because concept drift is such a broad category of phenomenon, this research has necessarily focused on a specific formulation of concept drift. The most common formulation is roughly

> At regular time intervals, new instances become available. A model must predict the label corresponding to each new instance. Immediately after the prediction has been made the true label becomes available. The learner then incrementally updates the model based on the new instance-label pair in an online manner.

We will call this the **standard formulation of the concept drift problem**, or simply the standard formulation.

The standard formulation is attractive from an academic perspective. It is simple and covers a wide class of problems. It is easily rendered in synthetic benchmark datasets against which progress in the field can be measured. It can be naturally expressed in formal mathematics, so is easily tractable to theoretical study.

There are broadly two approaches to handling concept drift. "Blind" approaches do not explicitly model drift, instead allowing the model to gradually adapt to the new environment. "Informed" approaches, instead employ **drift detectors** to explicitly detect

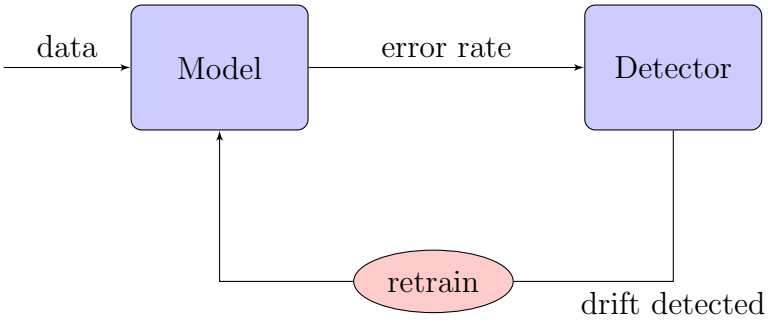


Figure 1.1: Informed approach to concept drift adaptation.

when concept drift has occurred so that the model can be retrained [24]. Blind approaches are often inadequate, as it can take too long for the model to adapt to the new environment. Informed approaches work roughly as follows:

> The drift detector monitors some performance metrics of the model. When it detects a degradation in performance, it infers that the model no longer reflects the current data distribution, and signals that concept drift has occurred. It then retrains the model using data which arrived after the drift was detected.

We will call this the **standard approach to concept drift adaptation**, and it is illustrated in Figure 1.1.

1.1 Motivation

In many applications the standard formulation of the concept drift problem diverges from reality in important ways. In this thesis we explore some algorithms which help bridge the gap between academic concept drift detection and practical data science applications.

We are motivated by a concrete problem from medical data science. We will return to this motivating example for illustration throughout the thesis. The problem is as follows.

> When a patient is referred to a medical facility, the referral is documented with free text and structured data, containing such information as condition, comorbidities, and demographics. From this data a clinician will make a decision about how urgently the patient needs to be addressed, and assign the patient a triage priority label. Some examples of triaging manuals are publicly available [12][55][3].
>
> Referral documents are often electronic, reflecting a broad trend of medical facilities switching from paper documents to electronic health records (EHR) [34]. These electronic documents present an opportunity [67]: supervised learning can be used to train a model which predicts triage labels

for referral documents. This model can then be incorporated into a decision support system to help clinicians make more efficient and systematic triage decisions.

An obvious benefit of such a support system is helping timely triaging of referrals. Automatic support system triage can provide a first pass on referrals, which can then be manually reviewed by clinicians. Another area in which this decision support system is expected to benefit public health is by countering potential bias in the healthcare system [1].

Staff, resources, policy, and medical best practices evolve over time, and the decision support system must be able to detect and adapt to these changes. For instance, suppose a condition is discovered to be particularly dangerous for some demographic. Clinicians will likely increase the average priority of patients with the condition in this demographic. A decision support system must be able to promptly detect that such a change has occurred, and trigger appropriate actions for the model to be corrected. That is, a decision support system must be sensitive to *concept drift*.

Within the medical context, a high degree of reliability is required. When concept drift is detected, a human expert is required to oversee and assist in the retraining of the model, as shown in Figure 1.2. When a GP makes a referral, the electronic referral document is sent to the model, a clinician, and a drift detector. The model predicts the triage label of the referral, and this prediction is sent to both the clinician, for decision support, and the drift detector. When the clinician decides on an "official" triage label, this is also sent to the drift detector. The detector monitors the incoming referral documents, predictions, and labels for signs of concept drift. If drift is detected, a data scientist is alerted. The data scientist inspects the output of the drift detector, and the distribution of the data stream, and decides if the model is still fit for purpose (FFP). If not, the data scientist may decide to retrain the model, or recall the model if its performance cannot be recovered to an acceptable level.

If the model is to be retrained, the data scientist must decide what data can be included in the new training set. For example, in the previous scenario, the model should be retrained on the full corpus of data, minus the referral documents of patients from the demographic group with the condition from before the priority change. The drift detector should help the data scientist figure out which data to use in retraining.

This setting diverges from the standard concept drift problem formulation in that there may be an arbitrarily long delays between new instances arriving and correct labels

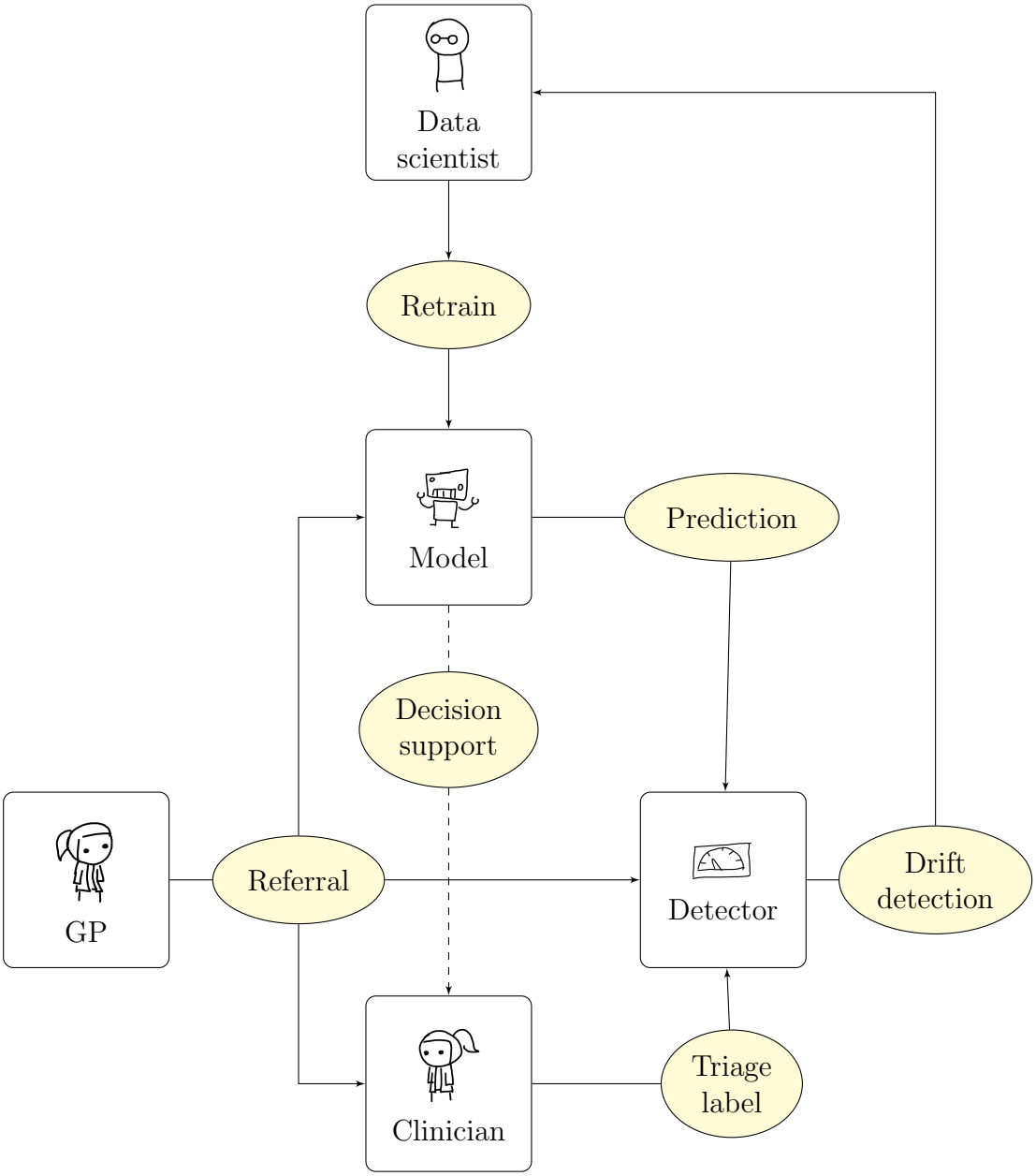


Figure 1.2: The proposed method for handling concept drift in GP referrals triage.

becoming available. Furthermore, the labels may become available in a different order to instances.

Also, unlike in the standard approach where the detector automatically retrains the model when drift is detected, in the GP referrals context, a human in the loop oversees and assists in model retraining.

How should a concept drift detector be designed to meet these challenges?

1.2 Problem Statement

To assist a human expert in making effectively adapting a model to concept drift, we would like to be answer the following questions:

1. If there is a delay between having access to new instances and their corresponding labels, is it possible to get early warnings that the data distribution has changed so that the model is no longer fit for purpose?

2. Sometimes model performance will degrade *irreducibly*, meaning that the average difficulty of the prediction task has become more difficult such that retraining the model will not be useful. Can we differentiate between reducible and irreducible error, so that the human expert can evaluate whether a model performance can be recovered after concept drift?

3. Given that it is not possible to know exactly when concept drift occurred or how severe it is, can we compute probability distributions over times when drift may have occurred, or over how much a model's performance has degraded due to concept drift? This would allow the expert to make retraining decisions informed by expected utility.

1.3 Objectives

The aims of our research are:

1. To develop a system which can provide early warnings of concept drift based on instance values when labels have not yet become available.

2. To develop drift detection algorithms which only detects reducible, and not irreducible, degradation in performance.

3. To develop a drift detection method which computes probability distributions for drift timing and model degradation.

1.4 Contributions

The main contributions of our work are:

1. A framework for providing early warnings that a model requires retraining or recall, the multiple drift detector (MDD). MDD monitors the instance distribution, the label distribution, and model performance metrics for indicators of concept drift.

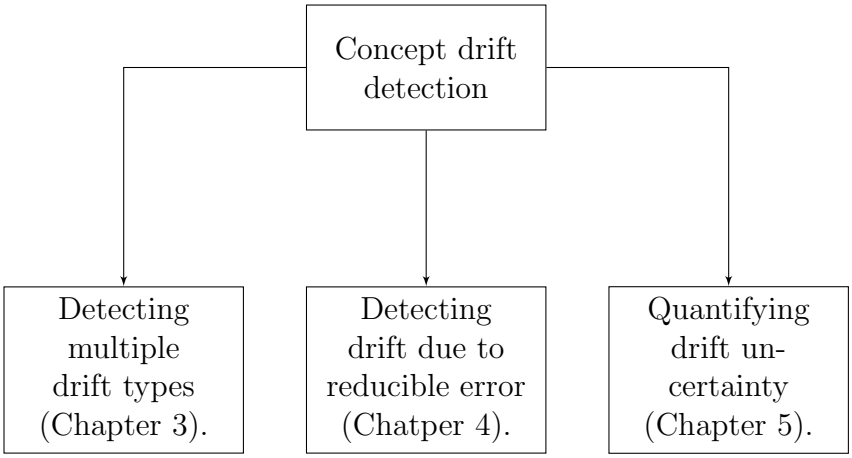


Figure 1.3: Overview of the topics covered in this thesis.

We also present a graphical interface for visualising the history of the status of MDD.

2. A drift detector which only detects reducible, and not irreducible, performance degradation, the calibrated drift detection method (CDDM).

3. A drift detection method which computes which computes probability distributions for drift timing and model degradation, the Bayesian drift detection method (BDDM). Additionally, we introduce beta with adaptive forgetfulness (BWAF), an efficient heuristic approximation of BDDM.

1.5 Overview of Research

The topics discussed in this thesis are illustrated in Figure 1.3. To understand how our contributions would assist the human expert in adapting the decision support system of our motivating example to concept drift, consider Figure 1.4.

If the distribution of instances changes, MDD will provide an early warning that "feature drift" has occurred. The expert inspects the distributional change using the MDD graphical interface, and decides whether the instance distribution has changed sufficiently that the model is no longer fit for purpose. If it the model is not fit for purpose, then it is recalled from the decision support system, so that it is not contributing clinically harmful predictions. Otherwise, no action is required.

If the relationship between instances and labels changes (i.e., if the triaging rules change), then MDD will warn that concept drift has occurred. At this point, the expert can consult the BDDM for a probability distribution over performance metrics, to determine if the model has sufficiently degraded to require retraining.

If the model *does* require retraining, then the expert can consult CDDM for whether

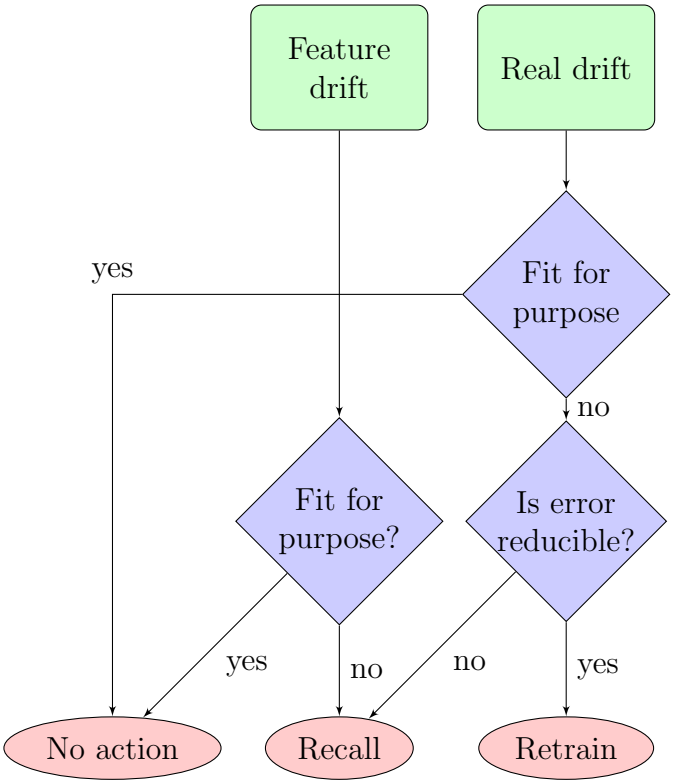


Figure 1.4: The decision making of the human expert.

the degradation is reducible or irreducible. If the change is irreducible, then by definition, model retraining will not recover model performance and the model must be recalled from the decision support system. If the degradation *is reducible*, then the expert can again consult BDDM for a probability distribution over drift timing. With this information, the expert can determine a suitable data set to retrain the model. If the retrained model performs adequately, then it can re-deployed in the decision support system.

1.6 Structure of this Thesis

This thesis is structured into the following chapters.

- Chapter 2 provides background on machine learning and data streams, and surveys related work on concept drift detection.
- Chapter 3 introduces multiple drift detector (MDD), a framework for providing early warnings that a model requires retraining or recall.
- Chapter 4 introduces calibrated drift detection method (CDDM), a drift detector which only detects reducible, and not irreducible, performance degradation.
- Chapter 5 introduces Bayesian drift detection method (BDDM), a method for exactly calculating posterior probabilities of drift timing and performance degradation.

We also introduce beta with adaptive forgetfulness (BWAF), an efficient heuristic approximation of BDDM.

- Chapter 6 describes experiments in which validate our novel drift detection methods. These involve a Bernoulli data stream, a battery of benchmark data streams, and a synthetic GP referrals triage data stream.
- Chapter 7 concludes this thesis by summarising the key points and discussing directions for future research.

2
Background

In this chapter we provide background and related work for this thesis. Section 2.1 introduces the setting of this work in machine learning and data streams, including notation and definitions. Section 2.2 discusses the related work on concept drift. Section 2.3 summarises the related work.

2.1 Setting

In this section we outline the setting of this thesis. Specifically, we provide the definitions and notation which will be used to describe machine learning and data streams.

A **label** is a random variable to be predicted by the machine learning model, denoted $y \in dom(y)$. In general, a label may be (non-exhaustively) binary, numeric, or categorical. However, in this thesis we will only be concerned with binary and categorical labels.[1]

A **binary label** may only take the values 0 and 1. We will denote binary labels with y. Because this is also the symbol for a generic label, we will explicitly note when a label is binary. A **multiclass label** may take on any of n values, $dom(y) = \{c_1, c_2, \ldots, c_n\}$.

An **instance** is a vector concatenation of one or more variables called **features**, and is denoted $x \in dom(x)$. Features may be (non-exhaustively) binary, numeric, categorical, or free text, and an instance may have any combination of feature types. The n-th feature

[1] An astute reader will note that we equivocate between y being a random variable and y being a specific value *of* that variable, and similarly with x, q, z, etc. This is to simplify notation.

of an instance is denoted $x^{(n)}$.

A **concept** is a joint probability of instances and labels:

$$\Pr(x, y). \tag{2.1}$$

It is often convenient to express this in terms of a "true" relationship between instances and labels, plus some amount of noise, as in:

$$y = f(x) + \epsilon(x) \tag{2.2}$$

where f is the **relationship**, and ϵ is **noise** random variable which may or may not vary with x.

The marginal probabilities of labels and instances are called the **label distribution** and the **instance distribution**, and denoted

$$P(y) \text{ and } P(x). \tag{2.3}$$

A **learner** is an algorithm designed to infer from a set of instance label pairs a **model** which approximates the relationship between instances and labels. The output of a model is called a **prediction** and is denoted $\hat{y}$.

There are two kinds of predictions we are concerned with. The first is a **point prediction**, in which the model simply outputs the prediction $\hat{y}$. The second is a **probabilistic prediction**, in which the model outputs a probability distribution over possible labels.

Due to noise, stochasticity, or the limited inferential capabilities of the learner, there is some probability that the model will make the wrong prediction. We thus denote

$$q = \Pr(y = \hat{y}) \tag{2.4}$$

as the **reliability** of the model, or the probability that the model will make the correct prediction, for the given instance.

For a model which makes probabilistic predictions, the **confidence** is the probability assigned by the model to its predicted label, denoted

$$\hat{q} = \hat{\Pr}(y = \hat{y}). \tag{2.5}$$

The **residual** of a prediction is a metric of the disparity between predictions and labels. For our purposes, it is given by

$$res = \mathbb{1}[y = \hat{y}] = \begin{cases} 1 & \text{if } y = \hat{y} \\ 0 & \text{if } y \neq \hat{y}. \end{cases} . \tag{2.6}$$

A **time series** is a sequence of values which become successively available as time progresses. We denote these as

$$Z = z_0, z_1, z_2, \ldots \tag{2.7}$$

A **Bernoulli time series** consists of samples of a random variable where:

$$z_i = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1 - p \end{cases} \tag{2.8}$$

where p is the **rate** of the series. The rate of the residual series is called the **error rate** of the model.

The times at which each value in the time series becomes available are denoted by

$$\tau_0, \tau_1, \tau_2, \ldots \tag{2.9}$$

If the differences in time between all instances are equal, or formally

$$\tau_{i+1} - \tau_i = c \tag{2.10}$$

for all $i \in \mathbb{N}$ and some constant c, then the time series is an **evenly spaced time series**. Otherwise it is an **unevenly spaced time series**.

An example of an evenly spaced time series is hourly measurements of air quality [33]. An example of an unevenly spaced time series is our motivating example of GP referrals triage, in which new referral documents or clinician labels may become available at any time.

Some specific time series we are interested in are **instance series** or **feature series**, the **label series**, the **prediction series**, and the **residual series**, which are denoted, respectively,

$$X = x_0, x_1, x_2 \ldots \tag{2.11}$$

$$Y = y_0, y_1, y_2 \ldots \tag{2.12}$$

$$\hat{Y} = \hat{y}_0, \hat{y}_1, \hat{y}_2 \ldots \tag{2.13}$$

$$RES = res_0, res_1, res_2 \ldots \tag{2.14}$$

An instance time series and a label time series together are called a **data stream**. A **drift** is a change in the distribution of a series

$$\Pr_t(z) \neq \Pr_{t+1}(z) \tag{2.15}$$

where $\Pr_t(z)$ is the distribution of z at time t. The point at which a drift occurs is called

a **drift point** or **drift location**. In Equation 2.15, the drift point is $t + 1$.

Concept drift is any change in the joint distribution of instances and labels.

$$\Pr_{t}(x, y) \neq \Pr_{t+1}(x, y) \tag{2.16}$$

There are several sub-types of concept drift. Note that there are many variations in the terminology used to describe concept drift [43][63], so the definitions provided here are not universal. **Feature drift**, also known as and **virtual drift** or **instance drift**, is a change in the distribution of feature values.

$$\Pr_{t}(x) \neq \Pr_{t+1}(x) \tag{2.17}$$

Label drift is a change in the distribution of label values.

$$\Pr_{t}(y) \neq \Pr_{t+1}(y) \tag{2.18}$$

Real drift is a change in the distribution of labels conditional on instances. These are the most consequential type of concept drift as they can require a change in the decision boundary of the model.

$$\Pr_{t}(y|x) \neq \Pr_{t+1}(y|x) \tag{2.19}$$

These types of concept drift are not mutually exclusive, and in fact will often occur together.

We further differentiate feature drift into **on-manifold** and **off-manifold** feature drift. In on-manifold feature drift, the relative probability of instance values increase or decrease, but the set of possible instance values remains the same. In off-manifold feature drift, the set of possible instances changes. For example, in the domain of classifying emails into spam and non-spam, if a new type of "Nigerian prince" spam email begins circulating, then this is off-manifold feature drift. Conversely, if the relative frequency of "Nigerian prince" spam email increases from one in one thousand to one in one hundred, then this is on-manifold drift.

A rich taxonomy of different types of drift has been explored, as illustrated in Figure 2.1. For our purposes, we need only differentiate between **abrupt** drift, where the stream distribution changes from one stable distribution to another stable distribution over a short period of time, and **gradual** drift, where the data stream goes through many "intermediate distributions" over a long period of time before stabilising on a new distribution.

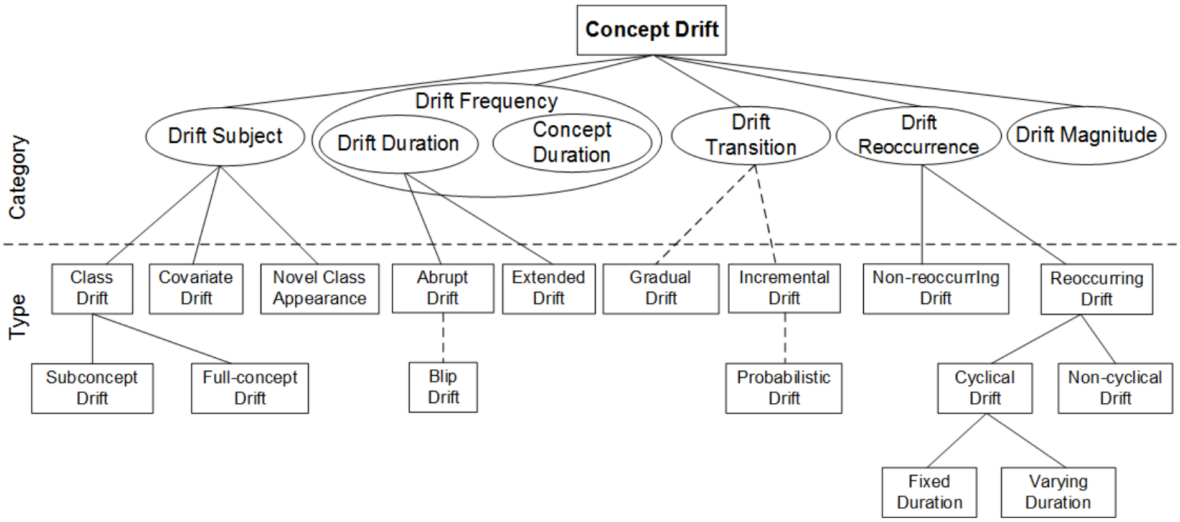


Figure 2.1: Taxonomy of drift types. Solid lines denote mutually exclusive subcategories. Dashed lines denote non-mutually exclusive subcategories. Illustration originally appeared in Webb et al. [63].

2.2 Related Work

There are broadly two approaches to handling concept drift. **Blind** approaches do not explicitly model drift, instead allowing the model to gradually adapt to the new environment. **Informed** approaches, instead employ **drift detectors** to explicitly detect when concept drift has occurred so that the model can be retrained on data from the drift point onward [24].

A drift detector is an algorithm which predicts whether drift has occurred for a given data stream

$$D(Z) \approx \begin{cases} 1 & \text{if drift has occurred on } Z \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

The following are desirable properties for a drift detector:

- **Adaptiveness** The ability to detect when concept drift has occurred in a short span of time so that model retraining can commence quickly.
- **Accuracy** The system should have a low rate of false-positives and false negatives, so that unnecessary training isn't invoked, and necessary training isn't neglected.
- **Robustness** The system should be robust to noise.

We now present a survey of the field of concept drift detection research.

2.2.1 Early Algorithms

Page [47] introduced the field of change detection, the parent field of concept drift detection. Page proposed the Page-Hinkley Test (PHT), also known as CUSUM algorithm, in the context of industrial quality control. We imagine testing fixed-size samples of some product from various batches. The number of faulty products from batch i is given by x_i. We would like to raise the alarm that the industrial process has gone awry whenever the following condition is met:

$$\begin{cases} x_n > 1 & \text{or} \\ x_n + x_{n-1} > 2 & \text{or} \\ \vdots & \\ \sum_{i=0}^{k} x_{n-i} > k+1 & \text{for some } k \end{cases} \tag{2.21}$$

This is equivalent to recording the cumulative sum $S_n = \sum_{i=0}^{n} x_i$ and raising the alarm whenever

$$S_n - \min_i S_i > h \tag{2.22}$$

for some h. In practice, this means we need only keep two registers. First, S_n which accumulates x_i, and S_{min}, which stores the minimum value of S_i encountered so far.

Work on concept drift often describes adaptations of CUSUM to the problem of concept drift detection [10][24]. These works cite the original paper by Page [47], making the author of these variations unclear.

The study of concept drift *per se* appears to originate with the STAGGER algorithm [56]. This work was heavily influenced by cognitive psychology, so the original usage of "concepts drift" denoted changes in conjunctive definitions of words. For example, a concept drift could be `bachelor = unmarried AND man` changing to `bachelor = unmarried AND woman`. This work also introduced the STAGGER dataset, which has become a standard benchmark in the field.

The FLORA family of algorithms [66][64][65] expanded the study of symbolic concept drift, and introduced many ideas to the field including recurring concepts, sliding windows of recent examples, and dynamic changes to the window size depending on the stability of the concepts.

Klinkenberg and Joachim subsequently expanded the study of concept drift to real-valued domains [36]. Their approach was to train an SVM online, using a sliding window whose size is chosen to minimise generalisation error, which is estimated with the $\zeta\alpha$ metric. Although this approach was efficient and theoretically well motivated, it was tied specifically to SVMs, and so was not a general solution to the problem of concept drift detection.

The sequential probability ratio test (SPRT) [59] detects when a time series drifts from one distribution $P_1(z)$ to a new distribution $P_2(z)$. Let $z_1, z_2, \ldots, z_n$ be some sequence. SPRT monitors the metric

$$T_1^n = \log \frac{P_1(z_1, z_2, \ldots, z_n)}{P_2(z_1, z_2, \ldots, z_n)} = \sum_{i=1}^{n} \log \frac{P_1(z_i)}{P_2(z_i)} \tag{2.23}$$

When $T_1^n > L$, for some parameter L, SPRT signals that drift has occurred.

2.2.2 Drift Detection Method

Drift detection method (DDM) [23] maintains a running observed mean of the residual time series p. The uncertainty around the true error rate is estimated via a normal distribution (the true posterior is a beta distribution) with standard deviation $s = \sqrt{p(1-p)/i}$. A register of the lowest values of p and s are maintained, denoted p_{min} and s_{min}. If $s + p > s_{min} + 2 \cdot s_{min}$, DDM emits a drift warning, and all new instances from that point are stored in a buffer. If $s + p > s_{min} + 3 \cdot s_{min}$, DDM emits a signal that drift has occurred, and the model is retrained on the instances in the buffer.

RDDM [9] is a variant of DDM with periodic resets to become more reactive to changes in the error-rate. EDDM [8] is another variant, which monitors for changes in the distribution of *gaps* between errors, rather than changes in the error rate itself.

2.2.3 Sliding Window Methods

STEPD essentially approaches concept drift detection by testing the hypothesis "the error-rate changed n time steps ago for some fixed, and pre-set n" at each time step [46]. This is achieved by partitioning the data stream history into 1) a sliding window of the most recent n values, and 2) the preceding values. The statistical test of equal proportions is then used to test whether the rates of the two partitions are significantly different.

FTDD , FPDD, and FSDD [16] are variants of STEPD, which uses Fischer's exact test in place of the test of equal proportions, because the latter is inappropriate for small or imbalanced data.

WSTD is another variant, which uses the Wilcoxon rank sum statistical test instead of the test of equal proportions, and additionally limits the size of the second partition [15].

PL (paired learners) is another sliding window method [7]. However, rather than testing for differences in error rates of a single model between the two partitions, it instead tests for differences in the error rates of two models. One model, the "stable learner" is trained online on the entire data stream, and the other, the "reactive learner" is trained only on the contents of the sliding window. If the proportion of instances in

the window that are classified correctly by the reactive learner, but not the stable learner, exceeds some threshold, then drift is indicated.

FHDDM may be considered a hybrid of DDM and sliding window methods [50]. At each time step the error rate in the sliding window is estimated. If this error rate significantly exceeds its lowest value, then drift is indicated.

SEQDRIFT2 [48] uses the Bernstein bound to compare a window of the most recent instances with a reservoir of the preceding instances.

MDDM [51] uses both a sliding window *and* geometric or linear weighting, and uses the McDiarmid bound. It detects increases in the weighted error rate in the window.

2.2.4 Adaptive Windowing

A major flaw of the sliding window approaches is that they are brittle with respect to the window size parameter. If the window size is too large then there will be a large delay in the drift detection. If the window is too small then the detector will not be able to detect small drifts over random noise. ADWIN combats this problem by dynamically increasing or decreasing the window size [13]. The authors explain:

> The idea is simple: whenever two "large enough" subwindows of W exhibit "distinct enough" averages, one can conclude that the corresponding expected values are different, and the older portion of the window is dropped. In other words, W is kept as long as possible while the null hypothesis "μ_t has remained constant in W" is sustainable up to confidence δ.

This involves testing for drift at each time step in the window, which requires a multiple comparisons correction. One of the main advantages of ADWIN is that it provided theoretical guarantees on its false positive and negative rates.

SEED [32] is another drift detection method which considers multiple drift points. Unlike ADWIN, the data are processed in batches or "blocks", so most possible drift points are automatically excluded. When contiguous blocks are evaluated via the Hoeffding inequality to have the same rate, then the two are merged, thus eliminating one potential drift point.

2.2.5 EWMA

An exponentially weighted moving average (EWMA) [53] is an estimation of a variable x, which gives exponentially more weighting to more recent examples:

$$\hat{x}_t = \lambda \hat{x}_{t-1} + (1 - \lambda) x_t \tag{2.24}$$

where λ is the decay factor. EWMA is employed in several drift detection methods as it provides a way of estimating the current error rate of the model without knowing how far back in time a drift occurred. When the EWMA estimation of the error rate is significantly larger than the overall mean error rate, then we may conclude drift has occurred. ECDDM is a straightforward application of EWMA to concept drift detection, deriving p-values from a lookup table [54]. LFR also uses EWMA, although p-values are derived from Monte Carlo estimation [60].

2.2.6 HDDM

Frías-Blanco et al. [22] cast the drift detection problem as follows. There are n examples $x_1, \ldots, x_n$, and a hypothesised drift point d. The instances before the drift point are denoted $X = x_1, \ldots, x_{d-1}$, and their mean is denoted μ_X. Similarly, those instances after the drift point are denoted $Y = x_d, x_{d+1}, \ldots, x_n$, and their mean μ_Y. These means are estimated using either the sample average or exponentially weighted average. Depending on which estimator is used, the method is known as HDDM$_A$ or HDDM$_W$. The estimations are denoted $\hat{\mu}_X$ and $\hat{\mu}_Y$, respectively. The authors derive bounds for $P(\hat{\mu}_X + \epsilon < \hat{\mu}_Y | \mu_X \geq \mu_Y)$ and $P(\hat{\mu}_X > \hat{\mu}_Y | \mu_X + \epsilon < \mu_Y)$, thus bounding the false positive and false negative rates, respectively.

This is incorporated into an online drift detection algorithm as follows. At time t, the hypothesised drift point is set to $d = t$, and is fixed at this value until enough instances are accumulated to the right of the drift point that either $P(\hat{\mu}_X + \epsilon < \hat{\mu}_Y | \mu_X \geq \mu_Y)$ is statistically insignificant, in which case a drift is signalled, or $P(\mu_X + \epsilon < \mu_Y)$ is statistically insignificant, in which case d is set to the current time step. A major advantage of HDDM is that the derived bounds do not assume that the data stream to be monitored is Bernoulli, so can be used to detect drift on real-valued loss streams.

2.2.7 Region Drift

Most of the drift detectors discussed so far have supposed that when concept drift occurs, all the data before the drift becomes irrelevant. Liu et al. [40] introduced the notion of **region drift**, in which concept drift only affects a "region" of instance space. Thus all the data outside of this region from before the drift can be reused when retraining the model. The authors present a new drift detector, local drift degree (LLD), which processes data in batches and uses the nearest neighbour methods to determine if drift has occurred in each region since the previous batch.

2.2.8 Data Batches

Degree of drift (DoF) [57] considers data in batches. If the most recent two batches have a heterogeneous Euclidean/overlap metric (HEOM) which is s standard deviations above the mean of past batches, then drift is indicated.

2.2.9 Unbalanced Classes

Wang et al. [62][61] argue that most drift detectors, which detect concept drift via monitoring for increases in the error rate are unsuitable for data streams with unbalanced classes, because

> the minority class contributes too little to these performance measures compared to the majority class [and] too few examples from the minority classes can make the time arbitrarily long until the concept drift is detected.

Wang et al. thus propose drift detection method for online class imbalance (DDM-OCI). This monitors for decreases in the recall of the minority class, rather than decreases in the overall accuracy of the model.

Linear four rates (LFR) [60] expands on this idea. In addition to monitoring the recall of the minority class, it also monitors the recall of the majority class and the precision of both. It uses an EWMA-style [53] estimator with lookup tables to detect changes in these rates.

PerfSim [5] similarly monitors all the components of the confusion matrix, and uses the cosine similarity test to detect changes between batches of data. CSDD [29] uses the same test, but uses a sliding window rather than batches of data.

2.2.10 Bayesian Concept Drift Detection

Many researchers have proposed Bayesian approaches to change point detection, a problem closely related to concept drift detection. Barry and Hartigan [11] derived an $O(n^3)$ algorithm for detecting *multiple* change points within a time series. Fearnhead [21] developed an $O(n^2)$ algorithm to achieve the same task.

Adams and MacKay [4] adopted a Bayesian approach to change point detection for online contexts. This approach considers a hypothesis space of $l = 1, l = 2, \ldots$, where $l = i$ indicates that a drift point occurred i time steps in the past. At each new time step, the posterior $\Pr(l = i)$ for each i is recalculated.

Bach and Maloof [6] adopted the Adams-MacKay approach to concept drift adaptation. They proposed the Bayesian conditional model comparison (BCMC) algorithm, which makes predictions according to an ensemble of Bayesian learners, one for each value of $l = i$. The models are weighted according to the $\Pr(l = i)$ posteriors. The authors

also propose a more efficient algorithm which approximates BCMC, called the pruned Bayesian conditional model comparison (PBCMC).

2.3 Conclusion

With some abstraction, we can consider drift detection methods of consisting of three components. First is the method of generating drift point candidates. The common approaches are using a *sliding window*, in which case the candidate drift point is the start of the window, *aggregation*, in which case the data stream is aggregated into blocks and the spaces between these blocks are used as drift point candidates, *batching*, in which the spaces between batches of data are drift point candidates, and *thresholding*, in which case the candidate drift point is derived by monitoring for a summary statistic exceeding some threshold.

The second component of a drift detector is a statistical test. Many detectors make use of Hoeffding's test, Bernstein's test, McDiarmid's test, Fischer's exact test, or the cosine test.

The third component is the operationalisation of concept drift. The general problem of concept drift detection is virtually insoluble in its most general form, so drift detectors operationalise the problem to make it tractable. The most common operationalisation is to detect increases in the error rate of the model. Other approaches include monitoring for increases in the precision and recall. All the existing drift detectors we have surveyed are summarised according to these components in Table 2.1.

Table 2.1: Summary of existing drift detectors

Detector	Drift Points	Statistical Test	Operationalisation
ADWIN	aggregation	Hoeffding	Accuracy
CSDD	cosine	batches	Recall, Precision
CUSUM	threshold	-	Accuracy
DDM	threshold	-	Accuracy
DDM-OCI	threshold	-	Recall
DoF	batches	-	Accuracy
ECDDM	threshold	-	Accuracy
EDDM	threshold	Hoeffding	Accuracy
FHDDM	window	Hoeffding	Accuracy
FLORA	-	-	Accuracy
HDDM	window	Hoeffding	Accuracy
LDD	batches	-	Accuracy
LFR	threshold	-	Recall, Precision
MDDM	window	McDiarmid	Accuracy
PerfSim	window	cosine	Recall, Precision
PL	window	-	Accuracy
RDDM	threshold	Hoeffding	Accuracy
SEED	Blocks	Hoeffding	Accuracy
SeqDrift2	window	Bernstein	Accuracy
SPRT	threshold	-	-
STAGGER	-	-	Accuracy
STEPD	window	-	Accuracy

3

Multiple Drift Detector

In this chapter we introduce multiple drift detector (MDD), a framework early detection of multiple types of concept drift. Section 3.1 discusses the motivation for MDD. Section 3.2 describes the setting of MDD. Section 3.3 provides pseudocode for MDD. Section 3.4 introduces a graphical interface for MDD. Section 3.5 illustrates the utility of MDD for our motivating example. Section 3.6 summarises this chapter and discusses future work.

3.1 Motivation

Existing drift detectors typically take the following approach. The time series of one or more performance metrics is monitored, and if a decline in the mean or rate of metric can be detected, then drift is signalled. The most common metric is accuracy [23][8][13], although some detectors also use precision or recall [60][37][61]. The rationale is that under stable conditions the performance of the model should either improve or remain steady, so a decline in performance must indicate concept drift has occurred.

We posit that in some applications, a better approach to concept drift detection is to not only monitor the time series of performance metrics, but also the time series of feature values and labels. Multiple drift detection (MDD) achieves this via multiple instantiations of a drift detector to monitor for feature drift, label drift, and real drift.

The main benefit of this approach is it can provide an early warning of concept drift. Specifically, if *feature* drift occurs, a drift detector which is monitoring the feature time

series can alert the user to this change before all the labels become available. The user will then have the opportunity of recalling the model if it will no longer be able to perform adequately under the new distribution.

The second benefit of this approach is that it helps the user obtain a more complete picture of how the data stream is evolving. A drift detector which simply signals when one or more performance metrics has declined affords little interpretability. By monitoring each of precision, recall, accuracy, label frequencies, and feature values, the user can better understand what exactly is driving a decline in model performance and how the decline is manifesting. This can help the user in choosing a dataset for model retraining.

3.2 Setting

Let $Z = z_1, z_2, \ldots, z_n$ be some timer series. We can abstractly describe a drift detector as a function which takes time series and outputs an assessment of whether the distribution of the time series changes between z_1 and z_n.

Specifically, the drift detector takes the time series of a performance metric, and determines whether the expected value of the metric *declines*, as an improvement of the metric may simply indicate normal learning. Often the metric is restricted to binary values [46][13][8], although there are some drift detection methods, such as HDDM [22], which allow real-valued performance metrics. The most common metric used is accuracy (or equivalently, error rate), obtained via the time series of the model's residual.

We can thus represent drift detectors as functions whose domain is arbitrary length time series, and whose output is a binary evaluation of whether the expected value of the time series has decreased. We denote this as:

$$D_{-}(Z) = \begin{cases} 1 & \text{if } \mathbb{E}[z_1] > \mathbb{E}[z_n] \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

This function is then incorporated into a loop as in Algorithm 1 for practical drift adaptation. Often the drift detector is parameterised with a confidence value α. The drift detector only indicates that drift has occurred if the expected value of the metric has declined with confidence α.

$$D_{-}(Z;\alpha) = \begin{cases} 1 & \text{if } \Pr(\mathbb{E}[z_1] \leq \mathbb{E}[z_n]) \leq \alpha \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Given a drift detector as described above, we wish to construct a more general concept drift detector which can detect changes in the distribution of features, labels, *and* performance metrics. We will demonstrate how to construct such a drift detector by

Algorithm 1 The typical approach to adapting to concept drift with a drift detector.

Require: Drift detector $D(Z)$
Require: Model
  for $t = 1, 2, 3, \ldots$ **do**
    $M = m_1, m_2, \ldots, m_t$ ▷ The time series of the performance metric up to t
    if $D(M) = 1$ **then**
      model.retrain()
    else
      model.update(x_t, y_t)
    end if
  end for

The first step of this construction is to notice that we can construct a detector of *increases* in the expected value of a time series by simply applying a drift detector to one minus the time series:

$$D_+(Z;\alpha) = \begin{cases} 1 & \text{if } \Pr(\mathbb{E}[z_1] \geq \mathbb{E}[z_n]) \leq \alpha \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

$$= D_-(1 - Z;\alpha) \tag{3.4}$$

where $1 - Z = (1 - z_0), (1 - z_1), \ldots, (1 - z_t)$. Note that if the time series is real-valued, it suffices to take the negative of the time series:

$$D_+(Z) = D_-(-Z). \tag{3.5}$$

Equation 3.4 covers both the real and binary valued cases.

We can also construct a bidirectional drift detector, which indicates both increases and decreases of the expected value of the time series:

$$D_\pm(Z;\alpha) = \begin{cases} 1 & \text{if } \Pr(\mathbb{E}[z_1] = \mathbb{E}[z_n]) \leq \alpha \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

$$= D_+(Z;\alpha/2) \vee D_-(Z;\alpha/2). \tag{3.7}$$

Note that we have halved the confidence threshold as a Bonferonni correction for multiple comparisons.

Detecting changes in the distribution of a categorical variables can be achieved by

detecting changes in the rates of each of the categories:

$$D(Z;\alpha) = \begin{cases} 1 & \text{if } \Pr(z) \text{ has changed (with } p < \alpha) \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

$$= D(Z^{(0)};\alpha/n_c) \vee D(Z^{(1)};\alpha/n_c) \vee \cdots \vee D(Z^{(n_c)};\alpha/n_c) \tag{3.9}$$

where n_c is the number of categories, and

$$Z^{(i)} = z_1^{(i)}, z_2^{(i)}, \ldots, z_t^{(i)} \tag{3.10}$$

$$= \mathbb{1}[z_1 = i], \mathbb{1}[z_2 = i], \ldots, \mathbb{1}[z_t = i] \tag{3.11}$$

.

Detecting changes in multi-dimensional variables can be achieved by detecting changes in each of the dimensions separately:

$$D(Z;\alpha) = \begin{cases} 1 & \text{if } P(z) \text{ has changed (with } p < \alpha) \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

$$= D(Z^{(0)};\alpha/n_d) \vee D(Z^{(1)};\alpha/n_d) \vee \cdots \vee D(Z^{(n_c)};\alpha/n_d) \tag{3.13}$$

where n_d is the number of dimensions, and $Z^{(i)} = z_0^{(i)}, z_2^{(i)}, \ldots, z_t^{(i)}$.

Using these compositions, we can construct a multi-drift detector out of an arbitrary base drift detector.

3.3 Pseudocode

The multiple drift detector framework essentially consists of five procedures. First, is feature preprocessing. This converts categorical features into dummy variables and free text into bag of words representations. This allows all features to be processed as binary variables. The preprocessing pseudocode is given in Algorithm 2. Second, there is the initialisation procedure for constructing the multiple drift detector out of a base drift detector, following the procedure set out in the previous section. The initialisation pseudocode is given in Algorithm 3. Third, there is the procedure for updating the multiple drift detector when a new instance becomes available, whose pseudocode is given in Algorithm 4. Fourth, there is the procedure for updating the multiple drift detector when a new prediction becomes available, whose pseudocode is given in Algorithm 5. Finally, there is the procedure for updating the multiple drift detector when a new label becomes available, whose pseudocode is given in Algorithm 6. Algorithm 7 shows how all these algorithms are used together, by describing the entire updating loop of the multiple drift

detector.

Note that this Algorithms 4, 6, 5 use a single threshold to indicate when drift has occurred. Many drift detectors use two thresholds [23][8][22]. Passing the first threshold results in a drift warning, after which all new instance-label pairs are placed in a buffer. Passing the second threshold results in a drift signal, at which point the model is retrained on the content of the buffer. It is trivial to extend the given code to similarly accommodate two thresholds, which may or may not be useful depending on context.

Algorithm 2 Preprocess features for multiple drift detector

function PREPROCESS($x^{(0)}, x^{(1)}, \ldots, x^{(n)}$)
Require: The domains of each of the features $X^{(0)}, X^{(1)}, \ldots, X^{(n)}$
  $x' \leftarrow []$
  for $x^{(i)}, X^{(i)}$ **do**
    if $X^{(i)} = \mathbb{R}$ **then**
      $x' \leftarrow x' \cup x^{(i)}$ ▷ Leave real-valued features as-is.
    else if $X^{(i)} = \{0, 1\}$ **then**
      $x' \leftarrow x' \cup x^{(i)}$ ▷ Leave binary features as-is.
    else if $X^{(i)} = k \in \mathbb{N}$ **then**
      for i=1,2,...,k **do** ▷ Convert categorical features to dummy variables.
        $x' \leftarrow x' \cup \mathbb{1}[x^{(i)} = k]$
      end for
    else if $X^{(i)}$ is free-text with vocabulary V **then**
      for $v \in V$ **do** ▷ Convert free-text features to bags-of-words
        $x' \leftarrow x' \cup (v \in x^{(i)})$
      end for
    end if
  end for
end function

Algorithm 3 Initialise multiple drift detector

function INITIALISE
Require: Drift threshold α
Require: Instance dimensionality n_x
Require: Number of feature labels n_y
Require: Base drift detector $D_-(Z; \alpha)$
  $D_+(Z; \alpha) = D_-(1 - Z; \alpha)$ ▷ Construct each of the drift detectors
  $D_\pm(Z; \alpha) = D_-(1 - Z; \alpha/2) \vee D_+(Z; \alpha/2)$
  $D_x(X) = D_\pm(X^{(0)}; \alpha/4n_x) \vee D_\pm(X^{(1)}; \alpha/4n_x) \vee \cdots \vee D_\pm(X^{(n_x)}; \alpha/4n_x)$
  $D_y(Y) = D_\pm(X^{(0)}; \alpha/4n_y) \vee D_\pm(X^{(1)}; \alpha/4n_y) \vee \cdots \vee D_\pm(X^{(n_y)}; \alpha/4n_y)$
  $D_p(P) = D_\pm(X^{(0)}; \alpha/4n_y) \vee D_\pm(X^{(1)}; \alpha/4n_y) \vee \cdots \vee D_\pm(X^{(n_y)}; \alpha/4n_y)$
  $D_r(R) = D_\pm(X^{(0)}; \alpha/4n_y) \vee D_\pm(X^{(1)}; \alpha/4n_y) \vee \cdots \vee D_\pm(X^{(n_y)}; \alpha/4n_y)$
  $X, Y, P, R, \hat{Y} \leftarrow [], [], [], [], []$ ▷ Initialise each of the data streams as empty
  $\hat{y}$_lookup $\leftarrow \{\}$
end function

Algorithm 4 Update multiple drift detector when a new instance becomes available.

function AddInstance(x_t)
  $X \leftarrow X \cup x_t$
  if $D_x(X)$ **then**
    `status` $\leftarrow$ `Featuredrift`
  end if
end function

Algorithm 5 Update multiple drift detector when a new prediction becomes available.

function AddPrediction($\hat{y}_t$)
  $\hat{Y} \leftarrow \hat{Y} \cup \hat{y}_t$
  $\hat{y}$_lookup$[t] \leftarrow \hat{y}_t$
  if $D_y(\hat{Y})$ **then**
    `status` $\leftarrow$ `Labeldrift`
  end if
end function

Algorithm 6 Update multiple drift detector when a new label becomes available.

function AddPrediction(y_t)
  $\hat{y}_t \leftarrow \hat{y}$_lookup$[t]$
  $R^{(y_t)} \leftarrow P \cup \mathbb{1}[y_t = \hat{y}_t]$ ▷ Update recall
  $P^{(\hat{y}_t)} \leftarrow P \cup \mathbb{1}[y_t = \hat{y}_t]$ ▷ Update precision
  if $D_p(P)$ or $D_r(R)$ **then**
    `status` $\leftarrow$ `Realdrift`
  end if
end function

Algorithm 7 Main loop of multiple drift detector

Initialise
while No drift detected. **do**
  if new x_t **then**
    $\hat{y} \leftarrow$ model.predict(x_t)
    $x_t \leftarrow$Preprocess(x_t)
    AddPrediction($\hat{y}_t$)
    AddInstance(x_t)
  else if new y_t **then**
    AddLabel(y_t)
  end if
end while

3.4 Graphical Interface

To assist a human expert in interpreting the evolution of the data stream, we have implemented a graphical interface for MDD. In our implementation of MDD, a trace of each time series, along with the status of each component drift detector, is written to a CSV file. Each trace is then rendered as a time series using the Dash framework [2]. A

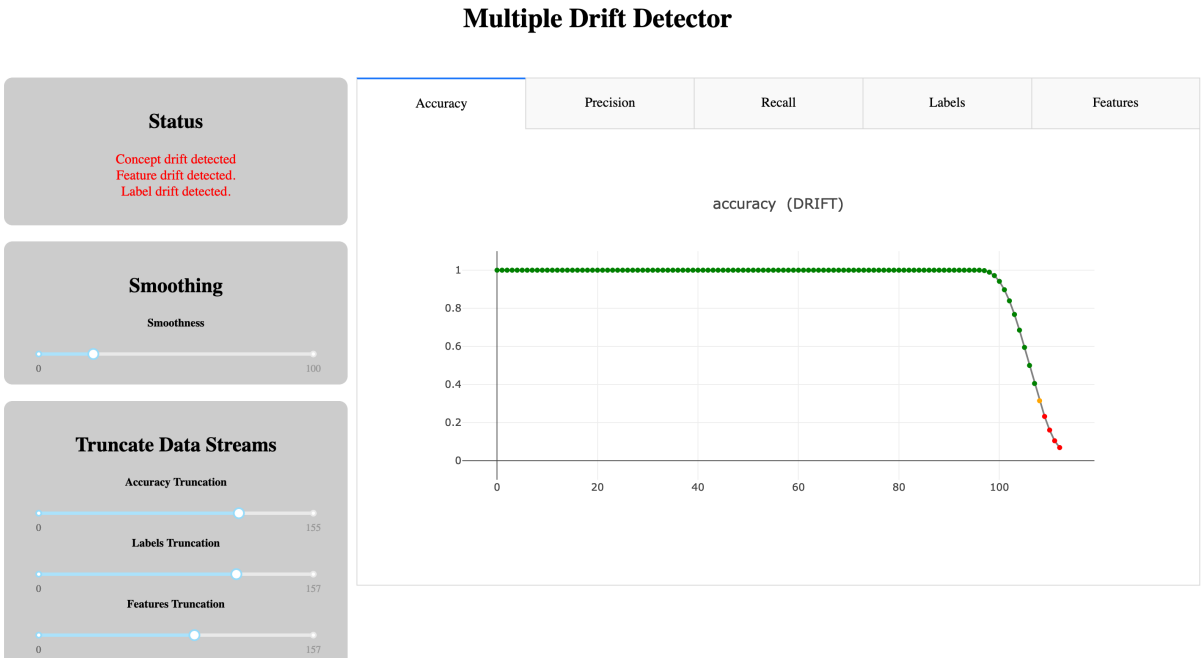


Figure 3.1: The graphical interface for multiple drift detector.

screenshot of the interface is shown in Figure 3.1.

Each time series is smoothed using a Hanning kernel [18], whose width can be adjusted. Each of accuracy, precision, recall, labels, features has a tab in which these time series can be inspected. The data points are coloured to indicate the drift status at that point in time, green points indicate that he time series is normal, orange that it is suspected of drift (i.e., the drift detector is in the "warning" state), and red indicates that the drift detector has been triggered. A summary of state of the data stream is given in the top right corner of the app, where it is stated whether concept drift, feature drift, label drift, or real drift have been detected by MDD. Within the individual tabs, one is given a break-down of the specific label values or features for which drift has been detected.

3.5 Illustration

In this section we illustrate MDD framework using our GP referrals triage motivating example. We explore three drift scenarios, and describe how MDD will facilitate responding to the drift appropriately.

3.5.1 Scenario 1: Retrain

The human expert commissioned to maintain the GP referrals triage decision support system receives a message from MDD indicating that model recall has decreased for priority 4, and precision has decreased for priority 3, as shown in Figure 3.2. Upon

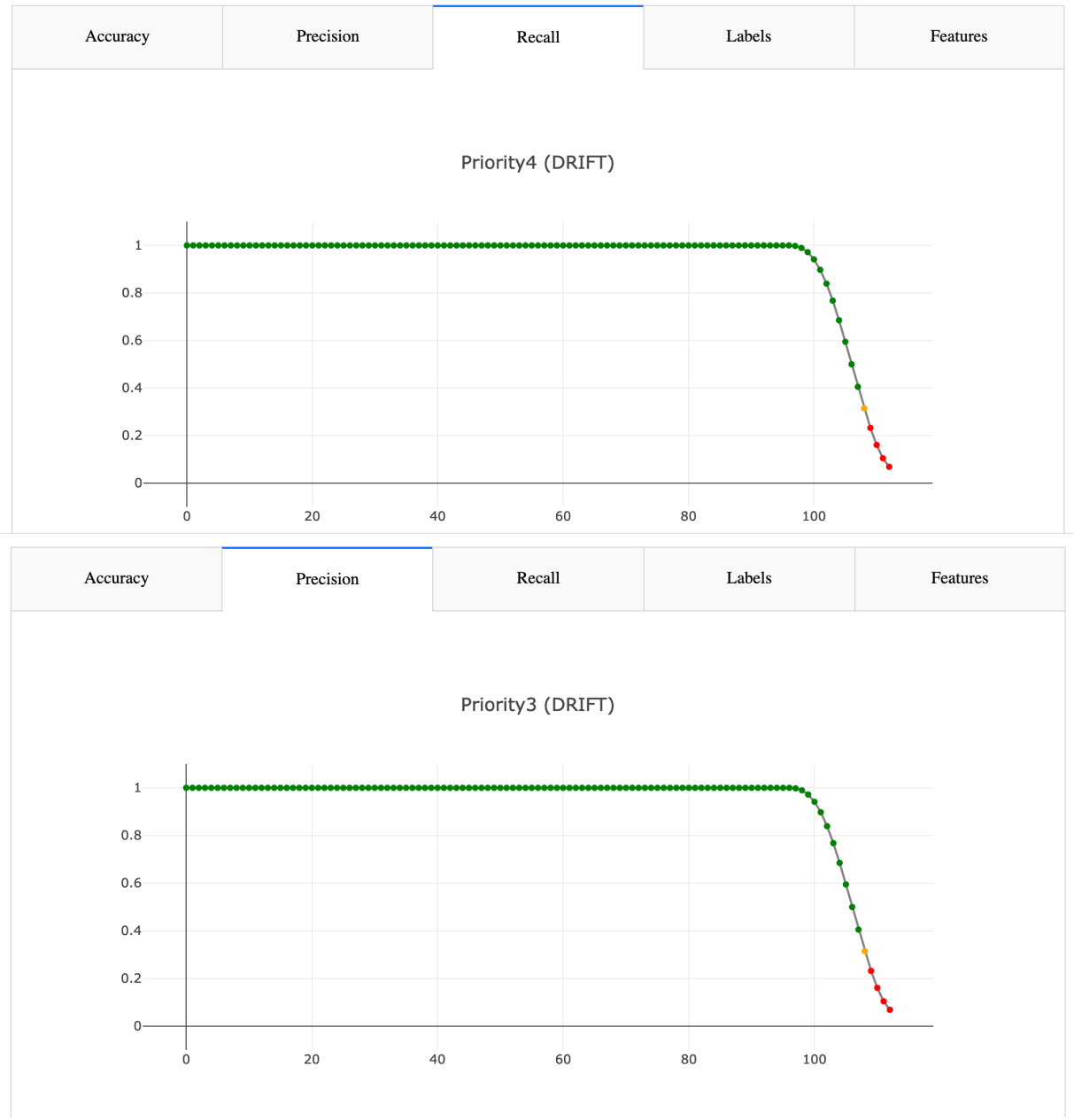


Figure 3.2: Scenario 1: Declining precision for triage priority 3 and declining recall for triage priority 4.

investigation, it turns out that a new study has been released showing that coronavirus is more dangerous than previously thought. Patients with this condition are now given priority 4 rather than priority 3.

The expert responds by removing all referral documents from the dataset of patients with coronavirus from before the study was released. The model is retrained on the new dataset, and is redeployed.

3.5.2 Scenario 2: No Action

The human expert receives a message indicating that feature drift and label drift have both occurred. The expert inspects the feature time series in the graphical interface, and it appears that there has been an increase in referrals for patients with coronavirus, as shown in Figure 3.3. Because these instances are given priority 4, there has also been an increase in priority 4 labels, hence the label drift. Because the model has been trained on an adequate amount of referral documents which include coronavirus, it is able to correctly predict the priorities and this is correct behaviour so no action is required.

The expert checks in on model performance a week later when the clinicians have caught given priority labels to the referrals under the new distribution. The performance metrics haven't changed significantly, and the model was indeed adjusting to the change in distribution correctly.

3.5.3 Scenario 3: Recall

The human expert receives a message indicating that feature drift has occurred. The expert inspects the feature time series in the graphical interface, and it appears that there has been an increase in referrals for patients with coronavirus, as shown in Figure 3.4. Because there were no coronavirus patients in the training data, the experts conclude that the model will be unable to make sensible triage predictions for this current wave of patients.These predictions are therefore clinically unsafe, and so should be withdrawn from the decision support system.

The expert therefore adds a rule to the decision support system stating that if a patient has coronavirus, the model should refrain from making a prediction, and annotate the referral as requiring a clinician to label it. Once a sufficient dataset of labels for coronavirus patients has accumulated, the model is retrained to handle this new class of patient referral.

3.6 Conclusion

In this chapter we introduced a framework for providing early warnings that a model requires retraining or recall, the multiple drift detector (MDD). MDD monitors the instance distribution, the label distribution, and model performance metrics for indicators of concept drift. We also present a graphical interface for visualising the evolution of the data stream and the state of MDD.

In future work, we would like to validate that MDD works well in real applications by performing a user study.

Figure 3.3: Scenario 2: Increase in the rate of triage priority 4 and increase in the rate of feature "Corona virus".

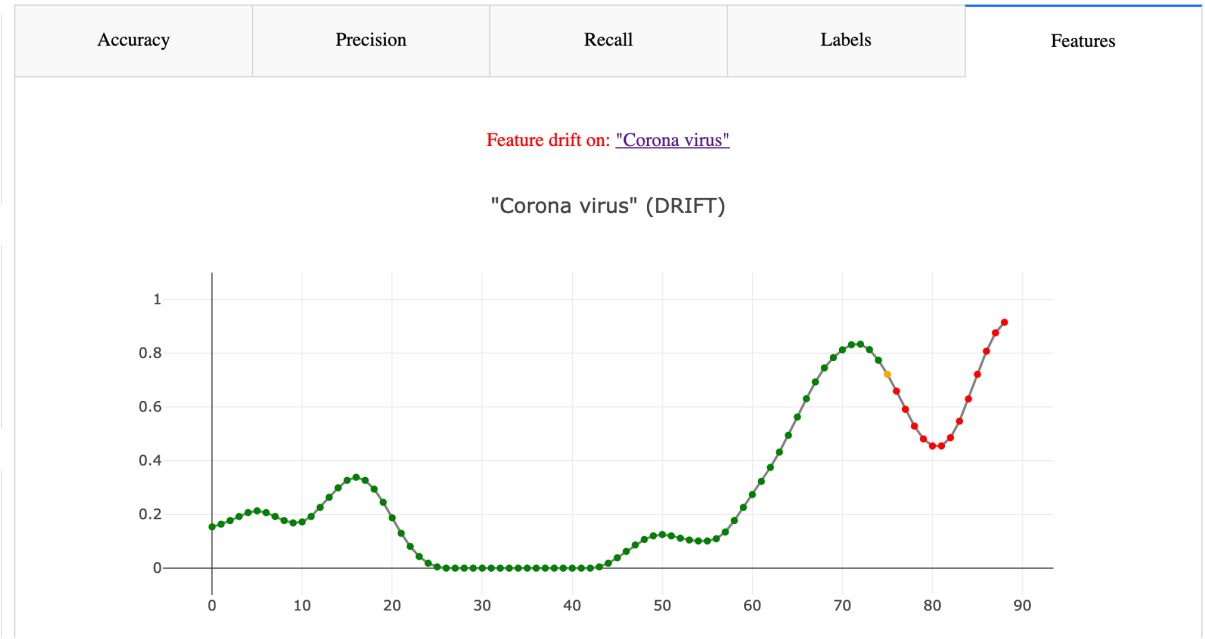


Figure 3.4: Scenario 3: Increase in the rate of feature “Coronavirus” only.

4

Calibrated Drift Detection Method

In this chapter we introduce calibrated drift detection method (CDDM), a drift detection method which detects increases in the irreducible error rate, rather than the overall error rate. Section 4.1 motivates the approach to concept drift detection taken by CDDM. Section 4.2 discusses the setting and notation of CDDM. Section 4.3 provides the actual CDDM algorithm. Section 4.4 discusses theoretical and practical limitations of CDDM. Section 4.5 summarises this chapter and discusses future work.

4.1 Motivation

Most, if not all, extant drift detectors assume that significant increases in the error rate of the base learner are due to concept drift and require model retraining. For example, Gama et al. [23] state

> Statistical [sic] theory [42] guarantees that while the class distribution of the examples is stationary, the error rate of the learning algorithm (p_i) will decrease when i increases. A significant increase in the error of the algorithm, suggest a change in the class distribution, and that the actual decision model is not appropriate.

However, this assumption is invalid in environments when 1) some labels can be predicted more easily than others, and 2) feature drift occurs. Consider the following (fictional) example from our motivating domain of medical triage:

> At a coronavirus emergency clinic, patients with coronavirus symptoms are being triaged. Young patients have a low mortality rate from coronavirus so are given low priority. Old patients have a high mortality rate so are given high priority. Middle aged patients, however, may have high or low mortality depending on other factors, so may be given a high, low, or medium priority.
>
> The learner is easily able to discover the relationship between age and priority, but fails to make use of other features. The model thus has a higher error rate for middle aged patients than young or old patients. If there is an increase in the number of middle aged patients with coronavirus, then the overall error rate of the model will increase.
>
> A concept drift detector will detect the increase in the error rate and signal that the model requires retraining. However, because the actual relationship between instances and labels has not changed, retraining the model would at best be a waste of time, and at worst result in an inferior model trained on a smaller dataset.
>
> Conversely, suppose that instead of an increase in the middle aged patients there is a *decrease in middle aged patients*. This will reduce the average difficulty of the prediction task, and reduce the error rate of the model.
>
> Suppose that shortly after this occurs, the clinic decides to change its triage policy for dealing with middle aged patients. Because the model is trained to implement the old policy, its error rate for middle aged patients will further increase, but this increase may be cancelled out by the decrease from the reduced number of middle aged patients.
>
> Thus, a concept drift detector may fail to notice the real drift which has occurred, and the model will not be retrained despite the change in the decision boundary. This situation may therefore result in a false negative.

This example illustrates that a concept drift detector should not monitor for increases in the error rate of the model *per se*. Instead it should monitor for increases in rate of *reducible* error due to *real* drift which is *not* predictable *ex ante* from the prediction task becoming more difficult on average. An increase in the *irreducible* error due to *virtual* drift which *is* predictable *ex ante* is a confounder. It can cause false negatives and false positives, such that the model may not be retrained when it should be, and may be retrained when it does not need to be.

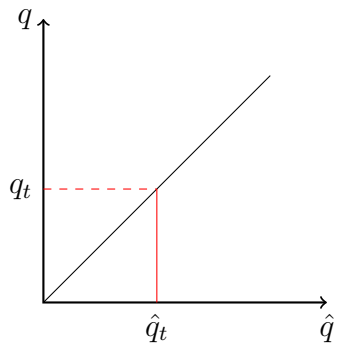


Figure 4.1: Calibration Graph

4.2 Setting

Let y be the true label given to some instance x, and $\hat{y}$ be the label predicted by the model. Due to noise, stochasticity, or the limited inferential capabilities of the learner, there is some probability that the model will make the wrong prediction. We thus denote

$$q = \Pr(y = \hat{y}) \tag{4.1}$$

as the **reliability** of the model, or the probability that the model will make the correct prediction, for the given instance.

We assume our model makes probabilistic predictions. We can thus talk about the model **confidence** as the probability assigned by the model to its predicted label, denoted

$$\hat{q} = \hat{\Pr}(y = \hat{y}). \tag{4.2}$$

A plot of a model's confidence versus its reliability is called a **reliability diagram** [45], and is illustrated in Figure 4.1.

When a model's confidence is equal to its reliability, $q = \hat{q}$, we say that the model is **calibrated** [58][26][45]. For example, if a calibrated model makes 10 predictions, and assigns a 0.9 confidence to each of them, then in expectation, one of those predictions will turn out to be incorrect. The reliability diagram in Figure 4.1 is of a calibrated model, as it shows an identity relationship between confidence and reliability.

When a model is calibrated, we can estimate the accuracy and error rate of the model from its confidence. Figure 4.2 shows a reliability diagram plus the distribution over confidence values, and thus the distribution over reliability. The accuracy is the expected value of the reliability

$$Acc = \mathbb{E}[q] \tag{4.3}$$

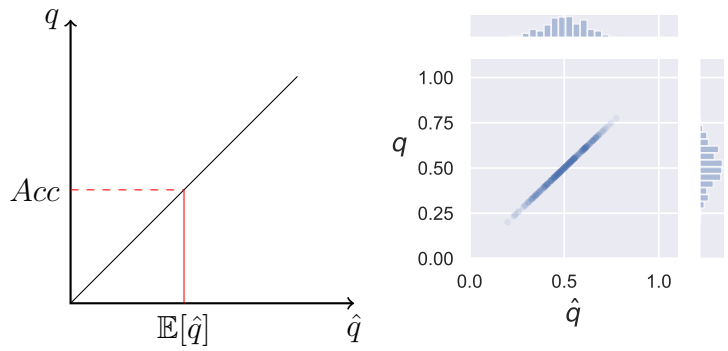


Figure 4.2: A calibrated model.

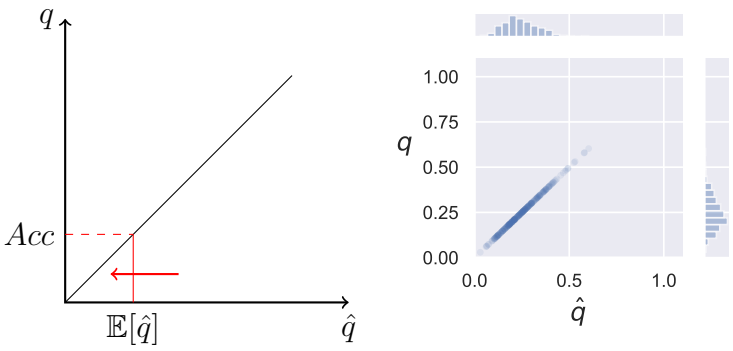


Figure 4.3: An increase in irreducible error rate.

and the error rate is one minus the accuracy:

$$Err = 1 - Acc = \mathbb{E}[q]. \tag{4.4}$$

In this manner we may derive an *ex ante* estimation of the model's accuracy based on its confidence scores. If the actual *ex post* accuracy significantly deviates from this accuracy, then we have evidence that the model is miscalibrated, which we take as evidence of concept drift.

Suppose the average difficulty of the prediction task increases, as described in Section 4.1, and illustrated in Figure 4.2. A calibrated model will decrease in its mean confidence, and our *ex ante* expected error rate will increase. When we observe an *ex post* increase in the error rate we will know that this can be attributed to feature drift rather than due to real drift, and so the model should not be retrained. In other words, we have observed an increase in the irreducible error rather than reducible error. We can thus avoid false positive.

Conversely, consider the other case described in Section 4.1 and illustrated in Figure 4.4, where the average difficulty of the prediction task decreases around the same time as real drift occurs. In this case the mean confidence of the model decreases, and so *ex ante* error rate decreases. If the *ex post* error rate does not decrease, this indicates that

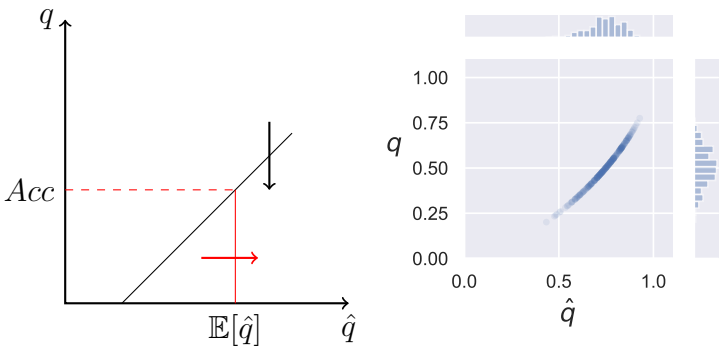


Figure 4.4: A decrease in irreducible error masking an increase in reducible error.

concept drift has occurred. In other words, there has been an increase in the reducible error rate, but it has been obscured by a decrease in the irreducible error rate. We can thus detect that the model *should* be retrained despite no increase in the overall error rate and thus avoid a false negative.

4.3 Algorithm

CDDM detects increases in reducible error rather than overall error by monitoring for differences in the *ex ante* accuracy and the *ex post* accuracy.

Intuitively, we can consider CDDM to be placing a "bet" on each prediction. If the model has confidence $\hat{q}$ in its prediction, then it will buy a bet that it is correct for $\$\hat{q}$. If the prediction turns out to be correct, the model will receive a payoff of \$1, otherwise it will receive a payoff of \$0. The payoff is denoted:

$$\gamma = \begin{cases} 1 - \hat{q} & \text{if } y = \hat{y} \\ -\hat{q} & \text{if } y \neq \hat{y} \end{cases} \tag{4.5}$$

The expected value of the payoff under normal, calibrated conditions is zero:

$$\begin{aligned} \mathbb{E}[\gamma] &= \Pr(y = \hat{y}) \cdot (1 - \hat{q}) - \Pr(y \neq \hat{y}) \cdot \hat{q} && (4.6) \\ &= q \cdot (1 - q) - (1 - q) \cdot q && (4.7) \\ &= 0. && (4.8) \end{aligned}$$

If we have observed n payoffs, $\gamma_1, \gamma_2, \ldots, \gamma_n$, then we can use Hoeffding's inequality [31]

to bound the probability of seeing a large average payoff:

$$\Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n}\gamma_i - \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\gamma_i\right]\right| \geq k\right) = \Pr\left(\left|\frac{1}{n}\sum_{i=1}^{n}\gamma_i\right| \geq k\right) \tag{4.9}$$

$$\leq 2\exp\left(-\frac{2n^2k^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right) \tag{4.10}$$

Where a_i and b_i are lower and upper bounds on γ_i, respectively. Because $0 \leq \hat{q} \leq 1$, we have $a_t = -1 \leq \mathbb{1}[y_t = \hat{y}_t] - \hat{q}_t \leq b_t = 1$.

$$\leq \exp\left(-\frac{2n^2k^2}{\sum_{i=1}^{n}4}\right) \tag{4.11}$$

$$\leq \exp\left(-\frac{nk^2}{2}\right). \tag{4.12}$$

CDDM maintains a sliding window of the most recent n payoff values. Equation 4.12 gives a p-value for an average payoff across this window at least as extreme as observed. If this value falls below some critical threshold, we may reject the null hypothesis and conclude that the model is miscalibrated. In this case, CDDM will signal that drift has occurred.

Similar to several other drift detectors [23][22][66], we use two critical thresholds: α_{warn}, a warning threshold, and α_{drift}, a drift threshold. When the p-value falls below α_{warn}, CDDM emits a warning that drift may be occurring, and all new instances are placed in a buffer. When the p-value falls below α_{drift}, CDDM indicates that concept drift has occurred. The model is then retrained on the instances in the buffer. The full pseudocode is given in Algorithm 8.

4.4 Limitations

CDDM suffers from two major limitations. The first is that CDDM requires learners to be calibrated, but most machine learning algorithms are not calibrated out of the box. The second is that small deviations from calibration will not be detectable by CDDM.

4.4.1 Calibration

CDDM requires the base learner to be calibrated. In fact few machine learning models are calibrated out of the box, and require post-hoc transformations of their probabilistic predictions to become calibrated.

Some models are "overconfident", meaning that the model's confidence tends to exceed its reliability, $\hat{q} > q$. Some deep learning models such as LeNet [39] suffer from

Algorithm 8 CDDM algorithm

Require: Warning threshold α_{warn}
Require: Drift threshold α_{drift}
Require: Window size N
  Window $\leftarrow$ []
  `status` $\leftarrow$ `normal`
  for $y_t, \hat{q}_t, \hat{y}_t$ in the data stream **do**
    $\gamma_t \leftarrow \mathbb{1}[y = \hat{y}] - \hat{q}$
    Window $\leftarrow \{\gamma_t\} \cup$ Window
    $n =$ Window.length
    if $n > N$ **then**
      Window.pop()
    end if
    $k \leftarrow |\text{Window.mean}|$
    $p \leftarrow 2\exp\left(-\frac{nk^2}{2}\right)$
    if $p \leq \alpha_{drift}$ **then**
      `status` $\leftarrow$ `drift`
    else if $p_{min} \leq \alpha_{warn}$ **then**
      `status` $\leftarrow$ `warn`
    end if
  end for

overconfidence [27].

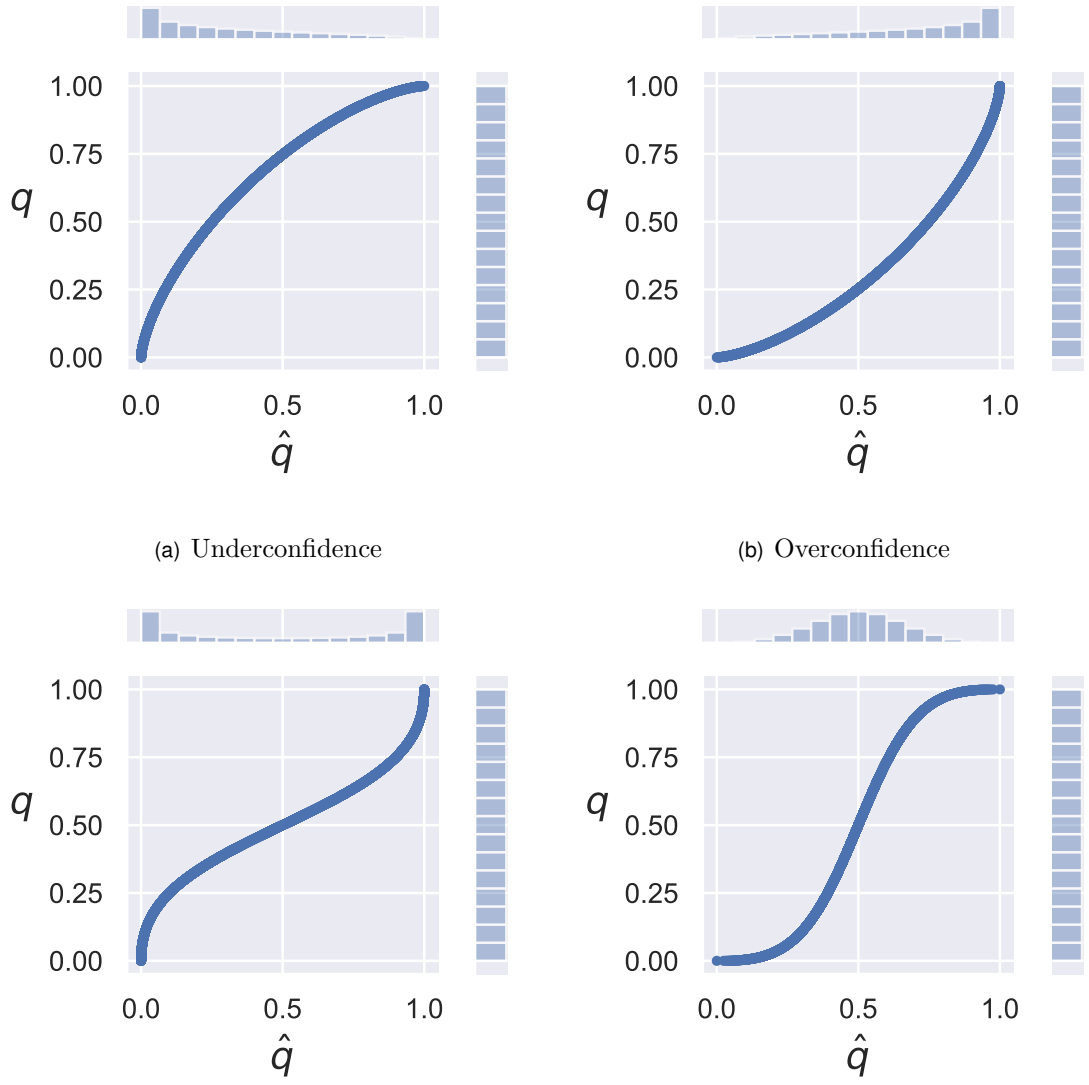
Conversely, some models are "underconfident", meaning that the model's reliability tends to exceed its confidence, $\hat{q} < q$. Some deep learning models such as ResNet [28] suffer from underconfidence [27].

Other models tend to "extremism", meaning that they are underconfident for confidences close to 0, and overconfident for confidences close to 1. Extremist models can be considered as pushing $\hat{q}$ values towards 0 and 1 with respect to q values. Maximum margin methods such as boosted trees and boosted stumps are prone to extremism [45].

The opposite behaviour to extremism is "moderatism", meaning the model is underconfident for confidence values close to 1 and overconfident for confidence values close to 0. Thus, $\hat{q}$ values are pushed *away from* 0 and 1 with respect to q values. Naïve Bayes models are prone to moderatism due to making unrealistic independence assumptions [45].

The terms "overconfident" and "underconfident" are standard in the prediction literature [58]. "Extremism" and "moderatism" are our own terminology. These behaviours are illustrated with reliability diagrams in Figure 4.5.

Models are often not calibrated because they are designed to optimise some metric other than calibration, such as accuracy. However, even in cases where models *are* optimised for calibration, miscalibration can still occur. For example, minimising cross entropy is a standard optimisation objective in deep learning [25]. This is a proper scoring



(a) Underconfidence

(b) Overconfidence

(c) Extremism

(d) Moderatism

Figure 4.5: Reliability diagrams for a common model behaviours.

rule [26], so this optimisation encourages both calibration and accuracy. However, many deep learning models still fail to calibrate [27]. This may be due to lack of robustness against dataset shift [43], or overfitting the training data [27].

A model which is not calibrated can be remedied by applying a post-processing function to a model's probabilistic predictions. This function is called a **calibration map** [45]. A variety of methods for generating calibration maps have been developed. Some of the major methods are given below.

Logistic calibration - also called **Platt scaling** [52] fits a sigmoid function, given by

$$q = \frac{1}{1 + \exp(a\hat{q} + b)} \tag{4.13}$$

to model probabilities. The parameters a and b are fitted using maximum likelihood estimation on a calibration set. Logistic calibration tends towards exactly correct calibration when the model's confidence values are normally distributed [38].

Isotonic calibration is a non-parametric type of calibration map [69]. It is therefore more flexible than logistic calibration, but more prone to overfitting. The only assumption isotonic regression makes about the reliability diagram is that it is isotonic, that is, it is monotonically increasing.

Beta calibration is a generalisation of sigmoid calibration [38]. A beta calibration map takes the form

$$q = \frac{1}{1 + 1/\left(e^c \frac{\hat{q}^a}{(1-\hat{p})^b}\right)} \tag{4.14}$$

where $a, b \geq 0$ so that the map is monotonically non-decreasing. This map can be fitted as easily as a logistic map [38]. Beta calibration is intended to overcome some of the drawbacks of logistic calibration, such as distortions for many classifiers including Naive Bayes and Adaboost. Beta calibration tends towards exactly correct calibration when the model's confidence values are beta distributed. Experiments have found beta calibration to be superior to logistic calibration for Naïve Bayes, Adaboost, Random Forests, logistic regression, support vector machines, and multilayer perceptrons [38].

Other calibration map methods include histogram binning [68], Bayesian binning into quantile (BBQ) [44], matrix and vector scaling [27], and temperature scaling [27][30].

4.4.2 Small Deviations from Calibration

CDDM cannot detect small deviations from calibration. Suppose the base learner has been calibrated up to time $t - n$, and from $t - n$ to t it has been slightly overconfident (or underconfident). Specifically,

$$\delta_i = \begin{cases} 0 & 0 \leq i < t - n \\ |\hat{q}_i - q_i| & t - n \leq i < t \end{cases} \tag{4.15}$$

Given a window size N, the miscalibration will be on the cusp of detectability from CDDM when the p-value given by equation Equation 4.12 is equal to the drift detection threshold ϵ:

$$\epsilon = \exp\left(-\frac{n^2\delta^2}{2N}\right). \tag{4.16}$$

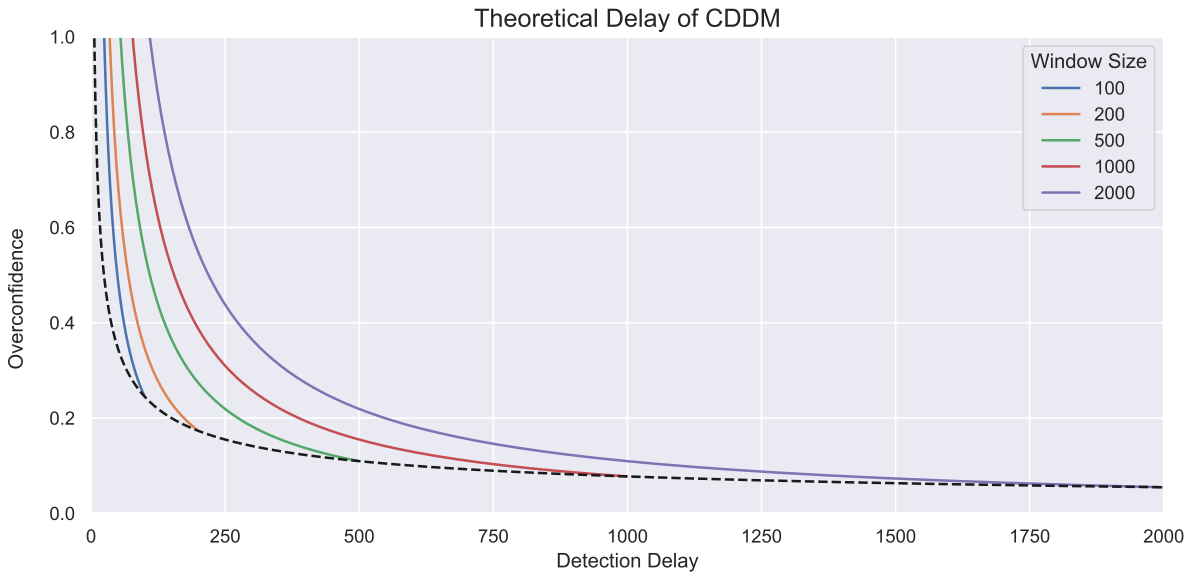


Figure 4.6: Detection delay versus overconfidence magnitude for CDDM.

Solving for δ gives the minimum overconfidence value which can be detected within n time steps for a given drift threshold and window size:

$$\delta = \sqrt{-\frac{2N}{n^2} \ln\left(\frac{\epsilon}{N}\right)}. \tag{4.17}$$

This relationship is plotted for a range of window sizes with drift threshold 0.01 in Figure 4.6. In this plot we can see that, for example, an overconfidence of $\delta = 0.4$ can be detected within 250 time steps for a window size of 2000.

The smallest overconfidence value which can possibly be detected by CDDM is found by setting $n = N$. That is, it is only detectable when the entire window has filled up with the miscalibrated payoffs. The range of detectable miscalibrations is thus given by

$$1 \geq \delta \geq \sqrt{-\frac{2}{N} \ln\left(\frac{\epsilon}{N}\right)}. \tag{4.18}$$

The dashed line in Figure 4.6 indicates this lower bound on detectable miscalibration. This shows that, for example, a miscalibration of magnitude $\delta = 0.1$ will be undetectable with a window size of $N = 200$.

4.5 Conclusion

In this chapter we introduced calibrated drift detection method (CDDM). We argued that a drift detector should detect increases in the reducible error rate rather than the overall error rate, so as to avoid unnecessary retraining. CDDM takes this approach

to concept drift detection by detecting when a model becomes miscalibrated. However, CDDM still has some significant limitations. Theoretically, CDDM can only detect mean overconfidence below a certain level determined by the window length. Practically, CDDM assumes that a learner is initially calibrated, whereas most models require prediction post-processing to achieve calibration.

A promising direction for future work would be building a calibration map online, and detecting when this calibration map changes, rather than assuming the learner is calibrated in the first place.

5

Bayesian Concept Drift Detection

In this chapter we explore Bayesian approaches to concept drift detection. Section 5.1 motivates a Bayesian approach to concept drift detection. Section 5.2 introduces Bayesian drift detection method (BDDM), an algorithm for computing posteriors probabilities of concept drift. Section 5.3 introduces Bayes with adaptive forgetfulness (BWAF), an efficient method for approximating posterior probabilities of concept drift. Section 5.4 summarises this chapter and discusses future work.

5.1 Motivation

In this chapter we consider methods for deriving a posterior probability distribution over concept drift locations, and of the current error rate of the model. We claim that such a derivation is valuable for many practical data science applications where a human is in the loop and there are strict requirements on performance. As illustration, we return to our motivating example of GP referrals triage. The details of this description are fictional.

> A model has been trained to predict triage labels for GP referrals documents. A drift detector is has been installed to detect when the error rate of the model increases, at which point it will alert a data scientist that the model requires retraining.
>
> Suppose the data scientist receives an alert that the error rate of the model has increased. The data scientist now has to decide 1) whether the model really

requires retraining, and if so 2) how far back in time should the new training data extend so that the new model is trained only on post-drift instances.

The model is required to have an error rate of at most 5%, or it is not considered fit for purpose. The cost of retraining the model is \$500, as it requires the data scientist to spend several days on site collecting data, and retraining the new model, validating the new model, and deploying the new model. The cost of keeping a model which is not fit for purpose deployed is estimated at \$5000 due to errors in the triage support leading to inefficient allocation of medical staff. If $\Pr(err > 0.05)$ is the probability that the error rate is greater than 5%, then it is worthwhile retraining the model as long as

$$\Pr(err > 0.05) \times \$5000 > (1 - \Pr(err > 0.05)) \times \$500. \tag{5.1}$$

This illustrates that it is not only important for the data scientist to know that the error rate has probably increased, but to also have a probability distribution over *how much* it has increased.

If the data scientist has access to a posterior probability distribution over possible error rates at the current time, as in Figure 5.1, then they can make a more informed decision about whether the model requires retraining or not.

Let us suppose the data scientist decides that the model *does* require retraining. They must now decide which data to use to retrain the model. Other things being equal, the data scientist would like to use more data rather than less so that the model will generalise better. However, the less recent the data, the more likely it is to have become outdated by the concept drift.

Again, the data scientist would be assisted by a posterior probability mass function, this time over possible times at which the concept drift occurred, as in Figure 5.2. This gives the posterior probability of a concept drift having occurred at each of the time steps. For time step t, this probability is denoted $\Pr(d = t)$.

Figure 5.2 also gives a cumulative mass function. This illustrates the posterior probability of concept drift having occurred *by* each time step. This is denoted as $\Pr(d \leq t)$.

With these posterior distributions, the data scientist can decide which instances to use in the training data to maximise expected utility.

For example, suppose the cost of including outdated instances is equal to the cost of not including relevant instances in the training data. In this case the data scientist will want to include all instances after the time at which the

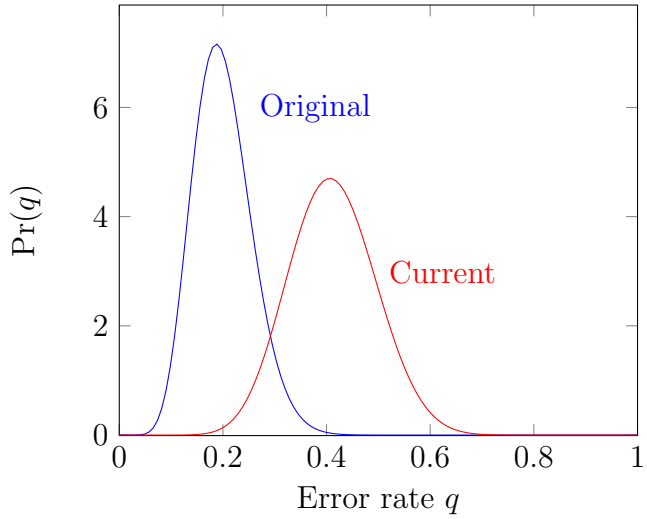
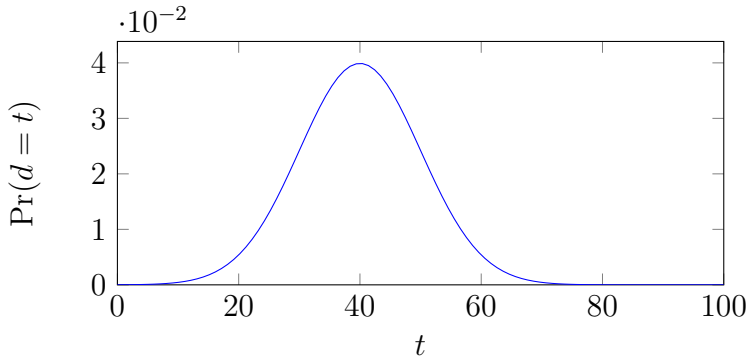
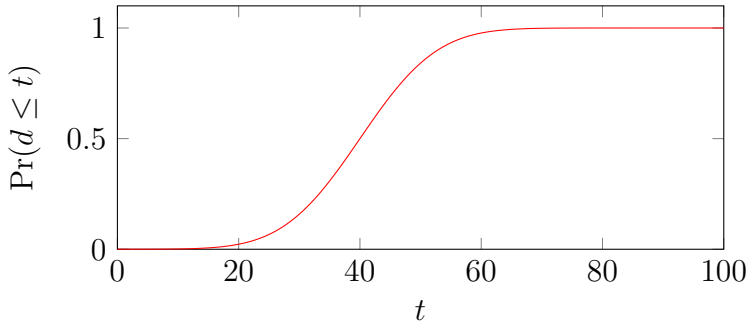


Figure 5.1: Posterior distribution over the original and current error rates of the model when concept drift has occurred.



(a) Probability mass function



(b) Cumulative mass function

Figure 5.2: Probability mass function and cumulative mass function over times at which a concept drift may have occurred.

probability of concept drift having occurred was 50%. Thus in Figure 5.2, the model would be retrained using instances which became available after $t = 40$.

5.2 Bayesian Drift Detection Method

In this section we introduce the Bayesian drift detection method (BDDM). BDDM precisely computes posterior distributions over drift points, as well as the error rates at the start and end of the time series. It is therefore more precise than other drift detectors, but at the cost of having $O(t^2)$ time complexity and $O(t)$ space complexity.

5.2.1 Setting

We pose the problem of computing a posterior over drift points as follows. Given a Bernoulli time series

$$Z = z_0, z_1, \ldots, z_t \tag{5.2}$$

we would like to compute the probabilities that the rate of the series changed at each time step, as well as the probability no change occurred. The time series in question is the series of residuals of the model. A change in the rate of this series indicates that the error rate has changed which indicates drift. This is the same strategy as used by most other drift detectors [23][13][6], and is referred to as the described as the accuracy-operationalisation in Chapter 2.

We denote the probability that a drift occurred at time step i as $\Pr(d = i)$. Specifically, $\Pr(d = i)$ is the prior probability. For convenience, we denote the probability that no drift has occurred as $\Pr(d = 0)$. The posterior probability is the probability of drift conditioned on the observed time series, denoted $\Pr(d = i|Z)$.

We can express the posterior probability of drift at a given time step using Bayes' rule:

$$\Pr(d = i|Z) = \frac{\Pr(Z|d = i)}{\Pr(Z)} \Pr(d = i) \tag{5.3}$$

$$= \frac{\Pr(Z|d = i)}{\sum_{j=0}^{t} \Pr(Z|d = j) \Pr(d = j)} \Pr(d = i) \tag{5.4}$$

For convenience, we will denote the joint probability of a given sequence with a given drift point as

$$P_i = \Pr(Z, d = i) \tag{5.5}$$

$$= \Pr(Z|d = i) \Pr(d = i). \tag{5.6}$$

The posterior probability that drift has occurred is given by

$$\Pr(d = 0|Z) = \frac{\sum_{i=1}^{t} P_i}{\sum_{i=0}^{t} P_i}. \tag{5.7}$$

5.2.2 Priors

We require a method for setting unitary priors on possible drift points.

$$\sum_{i=0}^{t} \Pr(d = i) = 1 \tag{5.8}$$

We propose two systems of priors for BDDM, one for evenly spaced time series and one for unevenly spaced time series. Our motivating example of GP referrals triage is an example of an unevenly spaced time series as GP referral documents or clinician labels can arrive at any time.

A natural prior for evenly spaced time series is a geometric distribution [21][6]. We suppose that there is a constant rate of concept drift, and that drift may only occur once. This produces the prior

$$\Pr(d = i) = \begin{cases} (1-\lambda)^t & \text{if } i = 0 \\ (1-\lambda)^{d-1}\lambda & \text{if } i > 0 \end{cases} \tag{5.9}$$

where $\lambda \in [0, 1]$ is the drift rate, which is the only parameter of BDDM.

We can generalise this strategy to unevenly spaced time series by modelling drift as a Poisson process. Again, we have a "rate" of drift, but now the rate is per unit time rather than per time step.

$$\Pr(d = i) = \begin{cases} P[N(\tau_t) = 0] & \text{if } i = 0 \\ P[N(\tau_{i-1}) = 0]P[N(\tau_i) > 0] & \text{if } i > 0 \end{cases} \tag{5.10}$$

$$= \begin{cases} e^{-\lambda\tau_t} & \text{if } i = 0 \\ e^{-\lambda\tau_{i-1}}\left(1 - e^{-\lambda(\tau_i - \tau_{i-1})}\right) & \text{if } i > 0 \end{cases} \tag{5.11}$$

$$= \begin{cases} e^{-\lambda\tau_t} & \text{if } i = 0 \\ e^{-\lambda\tau_{i-1}} - e^{-\lambda\tau_i} & \text{if } i > 0 \end{cases} \tag{5.12}$$

where $\lambda > 0$ is the drift rate, τ_i is the time at which the i-th value arrives, and $N(\tau)$ is the number of drifts which have occurred at time τ. Note that if the instances are evenly spaced in time then this is equivalent to the geometric prior.

5.2.3 Likelihoods

Suppose the error rate of the model x is drawn from a uniform distribution over $[0, 1]$. Then the probability of observing a given sequence of residuals is given by the beta

function.

$$\Pr(Z;x) = \int_0^1 x^{\sum_{i=0}^{t} z_i}(1-x)^{\sum_{i=0}^{t}(1-z_i)}dx \tag{5.13}$$

$$= B\left(1+\sum_{i=0}^{t} z_i, 1+\sum_{i=0}^{t}(1-z_i)\right) \tag{5.14}$$

$$= \frac{\left(\sum_{i=0}^{t} z_i\right)!\left(\sum_{i=0}^{t}(1-z_i)\right)!}{(n+1)!} \tag{5.15}$$

where $B(a,b)$ is the beta function.

We assume that if concept drift occurs, the error rates for before and after the drift are drawn independently. Thus

$$\Pr(Z|d=k) = \Pr(Z_{0:k-1})\Pr(Z_{k:t}) \tag{5.16}$$

$$= B\left(1+\sum_{i=0}^{k-1} z_i, 1+\sum_{i=0}^{k-1}(1-z_i)\right) B\left(1+\sum_{i=k}^{t} z_i, 1+\sum_{i=k}^{t}(1-z_i)\right). \tag{5.17}$$

5.2.4 Pseudocode

By combining the priors from Equation 5.12 and likelihoods from Equation 5.17 into Equation 5.4, we obtain the posterior probabilities of drift at each time step. BDDM is a straightforward computation of this equation, as shown in Algorithm 9. Clearly, BDDM is $O(t^2)$ in time complexity and $O(t)$ in space complexity.

Algorithm 9 BDDM algorithm

Require: Drift rate λ
Require: Warning threshold α_{warn}
Require: Drift threshold α_{drift}
  for $t = 0, 1, 2, \ldots$ **do**
    $c_t \leftarrow (z_t, 1-z_t)$ ▷ We store z values in dummy vectors.
    for $k = 0, 1, 2, \ldots, t$ **do**
      $P_i \leftarrow B\left(1+\sum_{i=0}^{k-1} c_i\right) B\left(1+\sum_{i=k}^{t} c_i\right) \Pr(d=\tau_k;\lambda)$
    end for
    $P(drift) \leftarrow \frac{\sum_{i=1}^{t} P_i}{\sum_{i=0}^{t} P_i}$
    if $P(drift) > \alpha_{warn}$ **then**
      `status` $\leftarrow$ `warning`
    else if $P(drift) > \alpha_{drift}$ **then**
      `status` $\leftarrow$ `drift`
    end if
  end for

5.2.5 Comparison with other Drift Detectors

BDDM is closely related, but distinct, from other Bayesian approaches to concept drift detection and change point detection. The only other Bayesian method which is intended to be directly applied to machine learning on volatile data streams is BCMC [6]. However, this algorithm requires maintaining an ensemble of one model per potential drift point, and so is too computationally expensive for many applications. The work of Adams and MacKay [4], Barry and Hartigan [11], and Fearnhead [21] are concerned with the more general problem of Bayesian multiple change-point detection in data streams, and so cannot be directly applied to our problem.

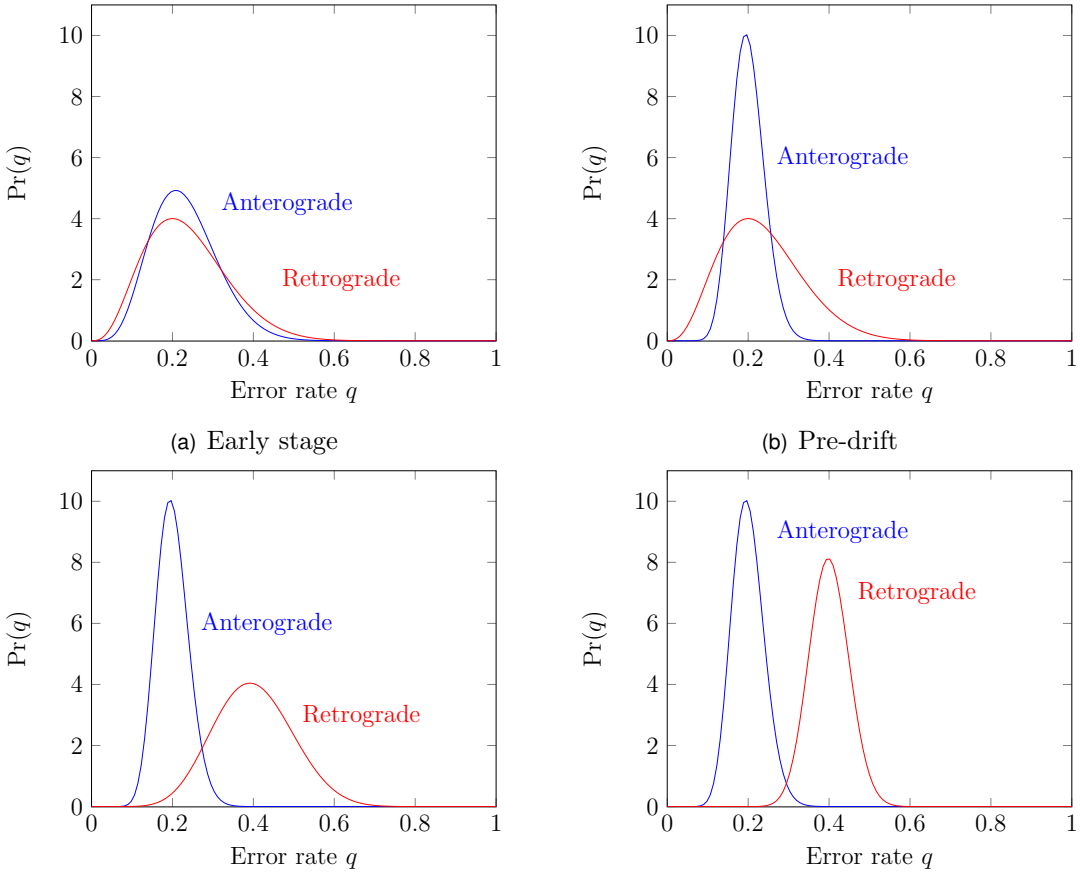
Modelling drift in a Bayesian manner has many advantages compared to other drift detection methods. By encoding information about time intervals in our priors, we can apply drift detection to both unevenly spaced time series and evenly spaced time series, a feature which most other drift detectors do not possess [20]. This allows the Bayesian calculations to have more common sense behaviour, such as assigning a higher probability to a concept drift occurring between consecutive instances separated by one day, than consecutive instances separated by one hour.

Another advantage of BDDM is that it does not require parameter tuning. The only parameter is the rate of drift used to calculate priors. This has an intuitive meaning, and its approximate magnitude should be apparent in most domains. This is particularly important in concept drift detection, as we often do not have annotated historical data for parameter tuning. There are few other drift detectors which have such simple parameters [23][47]. Other drift detectors require tuning thresholds [13][22], window sizes [46][7], or decay rates [22][53], which do not have intuitive meanings.

Because we are assigning precise probabilities to each candidate drift point, we can graphically plot the probability mass function (PMF) and cumulative mass function (CMF) of concept drift, as shown in Figure 5.2. This makes the drift detector more interpretable. Plotting a PMF or CMF is possible in principle for other drift detectors which explicitly test several candidate drift points, such as ADWIN [13] and SEED [32]. However, because these detectors use frequentist statistics and bound probabilities rather than estimating them, these distributions would be imprecise and non-unitary.

Another benefit of assigning probabilities to candidate drift point is it allows us to model the posterior distribution over the rates of Bernoulli streams before and after the drift point. The posterior distribution over rates in a stable Bernoulli time series is given by the beta distribution.

$$\Pr(q|Z) = \frac{x^{\sum_{i=0}^{t} z_i}(1-x)^{\sum_{i=0}^{t} 1-z_i}}{B\left(1+\sum_{i=0}^{t} z_i, 1+\sum_{i=0}^{t} 1-z_i\right)} \tag{5.18}$$



(a) Early stage

(b) Pre-drift

(c) Immediate post-drift

(d) Later post-drift

Figure 5.3: Progression of BWAF algorithm.

where q is the rate of the Bernoulli stream (the error-rate). Thus, the pre-drift posterior distribution over rates can be calculated from a sum of beta distributions, weighted by the posterior probabilities of each drift point.

$$\overrightarrow{\Pr}(q) = P_0 \Pr(q|Z_{0:t}) + \sum_{i=1}^{t} P_i \Pr(q|Z_{0:i-1}). \tag{5.19}$$

We can similarly derive the post-drift posterior distribution over rates.

$$\overleftarrow{\Pr}(q) = \sum_{i=0}^{t} P_i \Pr(q|Z_{i:t}). \tag{5.20}$$

An illustration of these posterior distributions is given in Figure 5.1. These posterior distributions add to the interpretability of BDDM, and allow retraining decisions to be made on the basis of expected utility.

5.3 Bayes With Adaptive Forgetfulness

We now introduce Bayes With Adaptive Forgetfulness (BWAF). This is a drift detector which uses a simple heuristic update rule to approximate the behaviour of BDDM. BWAF is very efficient: it is $O(1)$ space complexity and $O(t)$ in time complexity.

5.3.1 Amnesiac Distributions

From the Equation 5.18, we saw that:

$$\Pr(q|Z) \propto \prod_{i=0}^{t} x^{z_i}(1-x)^{1-z_i}. \tag{5.21}$$

or in log form

$$\ln\left(\Pr(q|Z)\right) \propto \sum_{i=0}^{t} z_i z \ln(x) + \sum_{i=0}^{t}(1-z_i)\ln(1-x). \tag{5.22}$$

In this formulation, it is natural to think of the posterior as accumulating update terms from each new observed value. We wish to estimate the current rate of the Bernoulli series at time t. Because drift may have occurred at any point in the past, we should give greater weights to the more recent updates. A natural way to do this is with exponentially decaying weights. Hence, we can heuristically estimate the posterior distribution over rates as:

$$\ln\left(\overleftarrow{\Pr}(q|Z)\right) \propto \sum_{i=0}^{t} \gamma^{\tau_t-\tau_i}(1-z_i)\ln(1-x) + \sum_{i=0}^{t} \gamma^{\tau_t-\tau_i} z_i z \ln(x) \tag{5.23}$$

where $0 \leq \gamma \leq 1$ is the **forgetfulness parameter**. Converting this back out of log form gives

$$\overleftarrow{\Pr}(q|Z) \propto \prod_{i=0}^{t}(1-x)^{(1-z_t)\gamma^{\tau_t-\tau_i}} \prod_{i=0}^{t} x^{z_t\gamma^{\tau_t-\tau_i}}. \tag{5.24}$$

Which can be expressed more compactly as

$$\overleftarrow{\Pr}(q) = \frac{x^a(1-x)^b}{B(a+1,b+1)} \tag{5.25}$$

with

$$a = \sum_{i=0}^{t} z_t\gamma^{\tau_t-\tau_i} \tag{5.26}$$

$$b = \sum_{i=0}^{t}(1-z_t)\gamma^{\tau_t-\tau_i}. \tag{5.27}$$

Thus the current posterior distribution over rates at time t can be derived from Equation 5.25 and the following update rules

$$a_t = \begin{cases} z_t & \text{if } t = 0 \\ z_t + \gamma^{\tau_t - \tau_{t-1}} a_{t-1} & \text{if } t > 0 \end{cases} \tag{5.28}$$

$$b_t = \begin{cases} (1 - z_t) & \text{if } t = 0 \\ (1 - z_t) + \gamma^{\tau_t - \tau_{t-1}} b_{t-1} & \text{if } t > 0 \end{cases} \tag{5.29}$$

We call this PMF the retrograde amnesiac distribution, as it "forgets" about about older values from the time series.

We now wish to estimate the initial rate of the Bernoulli stream. Naturally, we should use the opposite weighting scheme as for the retrograde amnesiac distribution. This can be easily achieved with

$$\overrightarrow{\Pr}(q) = \frac{x^{A-a}(1-x)^{B-b}}{B(A-a+1, B-b+1)} \tag{5.30}$$

where

$$A = \sum_{i=0}^{t} z_t \tag{5.31}$$

$$B = \sum_{i=0}^{t} (1 - z_t). \tag{5.32}$$

These have the update rules:

$$A_t = \begin{cases} z_t & \text{if } t = 0 \\ z_t + A_{t-1} & \text{if } t > 0 \end{cases} \tag{5.33}$$

$$B_t = \begin{cases} (1 - z_t) & \text{if } t = 0 \\ (1 - z_t) + B_{t-1} & \text{if } t > 0. \end{cases} \tag{5.34}$$

We call this PMF the anterograde amnesiac distribution, as it forgets about about *recent* values of the time series.

5.3.2 Adaptive Forgetfulness

With posterior distributions over the initial rate and current rate of the Bernoulli stream, we can compute the probability that the rate has increased.

$$\Pr(q_t > q_0) = \int_0^1 \overrightarrow{\Pr}(q_0) \int_{q_0}^1 \overleftarrow{\Pr}(q_t)\ dq_t\ dq_0. \tag{5.35}$$

We call this the **drift probability**. If the drift probability exceeds some critical threshold, BWAF will signal that drift has occurred.

How do we choose the forgetfulness parameter? We face a trade-off: if the parameter is small then the retrograde amnesiac distribution is more forgetful. This allows it to shift to a new distribution more quickly, and thus have a shorter detection delay. But it also means that it never remembers enough data to know the current rate with much precision. That is, the entropy of the posterior distribution will always be quite high. Thus, small changes in the rate will never be detected.

A good way to get around this problem is to dynamically adjust the forgetfulness parameter. When the drift probability is large, we want the retrograde amnesiac to be updated much more than the anterograde amnesiac. Thus the forgetfulness parameter should increase with the drift probability. The simplest way to achieve this is with the following update rule

$$\gamma \leftarrow \begin{cases} 0.5 & \text{when } t = 0 \\ \Pr(q_t > q_0) & \text{when } t > 0 \end{cases} \tag{5.36}$$

The progression of the anterograde and retrograde amnesiac distributions is illustrated in Figure 5.3. If no drift has occurred, the anterograde amnesiac distribution will converge towards the current rate. The retrograde amnesiac distribution will oscillate around the current rate, and the drift probability will oscillate around 0.5. The retrograde amnesiac distribution will thus remain high-entropy. When drift occurs, the the drift probability will increase. The forgetfulness parameter will also increase, allowing the retrograde amnesiac to converge towards the current distribution while the anterograde amnesiac continues to estimate the original distribution, thereby further increasing the forgetfulness parameter. The drift probability and the forgetfulness parameter will continue to mutually reinforce one another, until the drift probability reaches the critical threshold.

The pseudocode for the full BWAF procedure is given in Algorithm 10

5.3.3 Comparison with Other Drift Detectors

BWAF retains several of the advantages of BDDM. It retains the ability to view the posterior distribution over the current and initial rates of the Bernoulli stream. It does not retain the ability to plot a PMF or CMF of drift occurring, although one can create a diagram analagous to a CMF by plotting the drift probability over time. BWAF is therefore not as interpretable than BDDM, but is still more interetable than most drift detectors.

As with BDDM, BWAF can accomodate unevenly spaced time series. The choice of units for these intervals will affect the performance of BWAF, and choosing appropriate units is not straightforward. One might adopt the heuristic that the units should be

such that the mean interval is one, so that BWAF will behaviour similarly with unevenly spaced time series and evenly spaced time series.

A major advantage of BWAF is that it does not have any parameters. By dynamically adjusting the forgetting parameter it can in principle detect drifts of any magnitude or any level of abruptness. It is not clear if the same is true of any other drift detectors. DDM has no parameters, but cannot detect drifts which are small compared to sigma [23]. In this respect, BWAF is superior even to BDDM, which has the drift rate parameter. As aforementioned, in the case of unevenly spaced time series there is an implicit parameter in the choice of interval units.

It is interesting to note that BWAF has parallels with many disparate approaches to drift detection. The Bayesian connection has already been mentioned. Like DDM and EDDM, BWAF derives p-values from changes in the distribution over the rates of Bernoulli series [23][8]. Similar to ADWIN, a parameter of the model adapts online depending on drift conditions [13]. Finally, the exponential decay of distribution updates is very similar to EWMA [53].

Algorithm 10 BWAF algorithm

Require: Warning threshold α_{warn}
Require: Drift threshold α_{drift}
  $\gamma \leftarrow 0.5$
  $a, b, A, B \leftarrow 0, 0, 0, 0$
  for $t = 0, 1, 2, \ldots$ **do**
    $a \leftarrow (1 - z_t) + \gamma^{\tau_t - \tau_{t-1}} a$
    $b \leftarrow z_t + \gamma^{\tau_t - \tau_{t-1}} b$
    $A \leftarrow (1 - z_t) + A$
    $B \leftarrow z_t + B$
    $\Pr(drift) \leftarrow \frac{1}{B(a+1,b+1)B(A-a+1,B-b+1)} \int_0^1 (1-q_0)^{A-a} q_0^{B-b} \int_{q_0}^1 (1-q_t)^a q_t^b \, dq_t \, dq_0$
    $\gamma \leftarrow \Pr(drift)$
    if $\Pr(drift) > \alpha_{warn}$ **then**
      `status` $\leftarrow$ `warning`
    else if $\Pr(drift) > \alpha_{drift}$ **then**
      `status` $\leftarrow$ `drift`
    end if
  end for

5.4 Conclusion

In this chapter we have introduced BDDM, a drift detector which computes exact posterior probabilities of drift. We have also introduced BWAF, a heuristic drift detection method inspired by BDDM. These detectors offer many advantages compared to other drift detectors. They accommodate unevenly spaced time series as well as evenly spaced

time series. They simplify the task of parameter tuning. They are also more transparent than other drift detectors.

For future research, it would be useful to theoretically explore BWAF. Can performance guarantees be obtained? Are there different heuristics for updating the forgetfulness parameter which work as well or better?

6
Experiments

In this chapter we experimentally validate the novel drift detectors we have introduced, and investigate the performance of drift detectors on a synthetic medical triage dataset. Section 6.1 specifies the details which of the experiments which will be run in this chapter. Section 6.2 describes some experiments with Bernoulli data streams which validate the approach taken by CDDM. Section 6.3 investigates the performance of our drift detectors on a battery of benchmark data streams. Section 6.4 introduces a data stream which simulates a medical referral triage environment, and investigates the performance of our drift detectors on this dataset. Section 6.5 summarises this chapter and discusses future work.

6.1 Experiment Details

This section provides details of the experiments which will be performed in the following sections. The experiments are implemented using a modified version of the Tornado framework [49]. The experiments are performed on a 2.5 GHz Intel Core i7 with 16GB RAM. The operating system is macOS Mojave version 10.14.6.

Within each experiment suite, we run all the drift detectors which are in Tornado. These are ADWIN [13], CUSUM [47], DDM [23], EDDM [8], EWMA [53], FHDDM [50], FHDDMS [50], FHDDMS$_{add}$ [50], HDDM$_A$ [22], HDDM$_W$ [22], MDDM$_A$ [22], MDDM$_E$ [51], MDDM$_G$ [51], PH [47], RDDM [9], and SeqDrift2 [48]. CUSUM and PH are not the

original procedure proposed by Page [47], but the modified versions described in [24]. We compare these existing drift detectors with our novel detectors, BWAF and CDDM. We do not test BDDM due to it being $O(t^2)$ and therefore inappropriate for data streams.

This covers the most popular drift detection methods, although there are some notable omissions. All of these methods use the error-rate operationalisation of concept drift, so LFR [60] which is a state-of-the-art detector with an alternative operationalisation would have been a sensible comparison to CDDM. Similarly, BCMC is a Bayesian approach to concept drift detection, so would have been useful to compare to BWAF. However, to our knowledge, code for these detection methods is not publicly available.

Unless otherwise stated, the experiments are run using the three most popular models implemented in Tornado. These are the perceptron [14], the naïve Bayes, and the Hoeffding tree [19].

We evaluate the performance of detectors using the following metrics. A drift signal emitted by the drift detector within 250 time steps of a concept drift, is interpreted as a true positive (TP). A failure to emit a drift signal within 250 time steps of a concept drift is interpreted as a false negative (FN). A drift signal which is emitted when a concept drift has not occurred in the last 250 time steps is interpreted as a false positive (FP).

The **precision** of the detector is given by

$$\text{Precision} = \frac{N_{TP} + 1}{N_{TP} + N_{FP} + 2} \tag{6.1}$$

where N_{TP} is the number of true positives, and similarly for N_{FP} and N_{FN}. Note that this formula makes use of Laplace smoothing to avoid division-by-zero errors. The **recall** of the detector is given by

$$\text{Recall} = \frac{N_{TP} + 1}{N_{TP} + 1 + N_{FN} + 1} \tag{6.2}$$

The $\mathbf{F}_1$ score of the drift detector is given by

$$F_1 = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}. \tag{6.3}$$

The **mean delay** is the mean number of time steps between a concept drift occurring and a drift signal being emitted. Because any signals after 250 time steps are considered false positives, this is the maximum value for mean delay. **Memory** is the memory footprint of the drift detector in bytes measured at the end of an experimental trial. **Runtime** is the number of milliseconds between the beginning and end of the experimental trial.

Results are reported in tables in the form "mean (std)", where "std" is the sample standard deviation across the experimental trials. The best detector on each metric is rendered in bold. The results are additionally visualised using CD diagrams [17].

6.2 Bernoulli Experiments

In this section we describe experiments on a data stream which validates the benefit of CDDM's approach to concept drift detection. Specifically, we show how the pathologies discussed in Section 4.1 of false positive and false negative drift detections can occur from feature drift and variation in the difficulty of in prediction tasks.

In this data stream, instances consist of a single Bernoulli feature x. Labels y are likewise Bernoulli variables, whose rate depends on the feature value:

$$\Pr(x = i) = \begin{cases} 1 - \gamma & \text{if } x = 0 \\ \gamma & \text{if } x = 1. \end{cases} \tag{6.4}$$

$$\Pr(y = 1 | x = i) = \begin{cases} \alpha & \text{if } x = 0 \\ \beta & \text{if } x = 1. \end{cases} \tag{6.5}$$

We will call γ the feature rate, and α and β the label rates. The instance-label joint probability of the data stream is illustrated in Figure 6.1.

We are interested in two kinds of concept drift. First we have feature drift, in which the feature rate changes, but the label rates remain the same.

$$(\gamma, \alpha, \beta) \Rightarrow (\gamma', \alpha, \beta) \tag{6.6}$$

Second, we have real drift concurrent with virtual drift, in which the feature rate, *and* the label rates change.

$$(\gamma, \alpha, \beta) \Rightarrow (\gamma', \alpha', \beta') \tag{6.7}$$

When a drift detector is triggered by real drift, this is a true positive. When a drift detector is triggered by feature drift, this is a false positive. Our experiments with Bernoulli data streams consist of 1000 time steps under the pre-drift distribution, followed 1000 time steps in the post-drift distribution.

We use naïve Bayes as our learner. Naïve Bayes' are known to be poorly calibrated [45], so are typically a poor choice to use with CDDM. However, given this data stream has one-dimensional instances, the usual issues of independence assumptions between features does not arise.

In this data stream, we have one feature value whose label is hard to predict, and one feature value whose label is easy to predict. Specifically, a feature an $x = 0$ instance is consistently labelled as $y = 0$, with no noise. A feature value of $x = 1$ is also labelled as $y = 0$, but with a noise rate 0.2. The feature rate is uniformly sampled for each instance

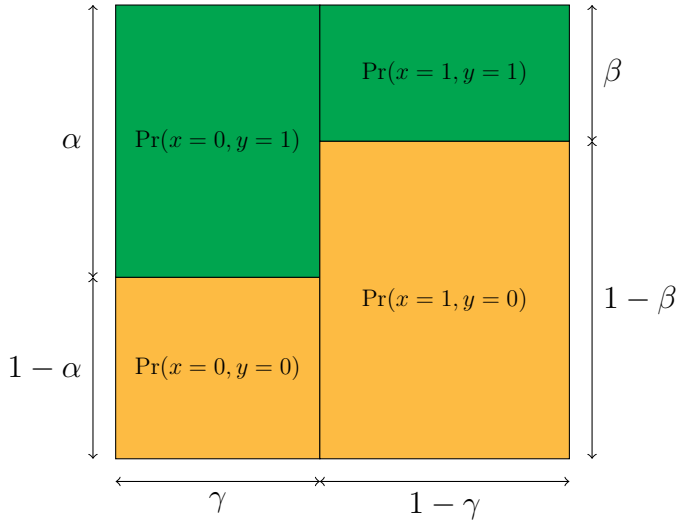


Figure 6.1: Distribution of the Bernoulli data stream.

of the data stream.

$$\alpha = 0 \tag{6.8}$$

$$\beta = 0.2 \tag{6.9}$$

$$\gamma \sim U[0, 1] \tag{6.10}$$

Virtual drift is implemented by reversing the feature rate.

$$(\gamma, \alpha, \beta) \Rightarrow (1 - \gamma, \alpha, \beta) \tag{6.11}$$

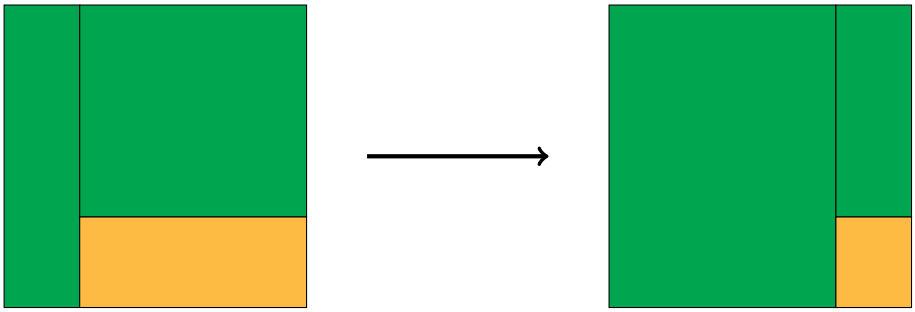
Real drift is implemented by reversing the feature rate *and* reversing the first label rate.

$$(\gamma, \alpha, \beta) \Rightarrow (1 - \gamma, 1 - \alpha, \beta) \tag{6.12}$$

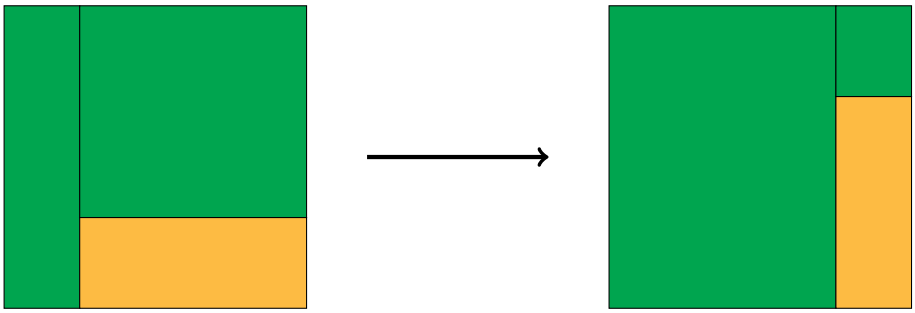
These two scenarios are illustrated in Figure 6.2.

If the initial feature rate is less than 0.5, then after virtual drift we will see an increase in the error rate of the model due to an increase in the rate of "hard problems".

Conversely, when the feature rate is greater than 0.5, when real drift occurs we will see a decrease in the rate of "hard problems", which will decrease the error rate. This may hide the increase in the error rate due to the real drift itself, and can lead to false negatives.



Feature drift



Real drift

Figure 6.2: Distributional changes in the Bernoulli data stream.

More specifically, the change in error rate due to feature drift will be positive when

$$0 < \Delta E \tag{6.13}$$
$$< (1-\gamma)\beta - \gamma\beta \tag{6.14}$$
$$< \beta - 2\gamma\beta \tag{6.15}$$
$$\gamma < 0.5. \tag{6.16}$$

Thus, any feature rate greater than 0.5 may trigger false positives in error-rate detectors. The change in error rate due to real drift is positive when

$$0 < \Delta E \tag{6.17}$$
$$< (1-\gamma)(1-\beta) - \gamma\beta \tag{6.18}$$
$$< 1 - \gamma - \beta \tag{6.19}$$
$$\gamma < 0.8. \tag{6.20}$$

Thus false negatives may occur when the feature rate is above 0.8.

The results of the trials on this data stream are given in Table 6.1, and illustrated in Figure 6.3. CDDM achieves the highest precision and F_1 score of any of the detectors

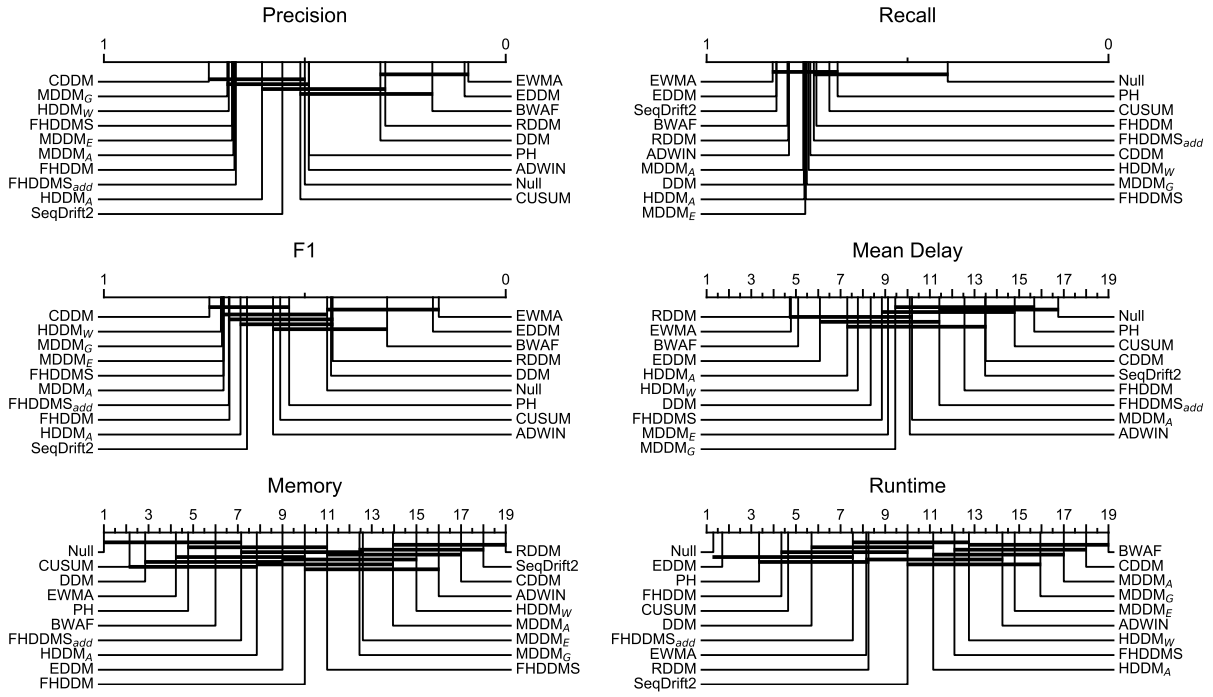


Figure 6.3: CD diagram of Bernoulli datasets.

(although not significantly greater than the runner up at $p < 0.05$ significance).

6.3 Benchmark Datasets

In this section we evaluate the concept drift detection methods we have introduced on a battery of benchmark datasets. The Tornado framework provides implementations of the following synthetic data streams:

- **CIRCLES** Each instance consists of two attributes $x_1, x_2 \sim U[0, 1]$. Each concept consists of three values, $r, x_{circ}, y_{circ} \in [0, 1]$, representing the radius of a circle, and x and y coordinates of the center of a circle. And instance is given a positive label if it falls within the circle, otherwise it is given a negative label. Sampling is done such that an equal number of positive and negative instances occur. The concept parameters cycle through the values of $[0.15, 0.2, 0.5]$, $[0.2, 0.4, 0.5]$, $[0.25, 0.6, 0.5]$ and $[0.3, 0.8, 0.5]$.

- **LED** Each concept in the LED task is a digit displayed on a 7-bit LED interface. There are 10 labels (the digits $0, 1, 2, \ldots, 9$) and 7 binary features (each of the display bits). For example, the label corresponding to the instance [1, 1, 1, 1, 1, 1, 0] is 0. In addition to the concept bits, there are also n irrelevant and random-valued binary attributes. The position of the irrelevant attributes changes with each concept, so the model must learn anew which attributes are decision-relevant.

Table 6.1: Results of Bernoulli datasets.

Detector	Precision	Recall	F1	Mean Delay	Memory (bytes)	Runtime (ms)
ADWIN	0.49 (0.29)	0.80 (0.20)	0.58 (0.28)	125.56 (94.84)	6612.64 (182.58)	67.49 (0.85)
BWAF	0.18 (0.07)	0.80 (0.20)	0.30 (0.11)	**86.77 (94.32)**	1023.98 (0.09)	103.07 (1.03)
CDDM	**0.74 (0.21)**	0.74 (0.23)	**0.74 (0.22)**	169.76 (71.85)	8944.00 (0.00)	98.20 (2.25)
CUSUM	0.51 (0.25)	0.69 (0.25)	0.56 (0.25)	184.09 (73.03)	880.00 (0.00)	38.25 (0.89)
DDM	0.31 (0.14)	0.76 (0.23)	0.43 (0.17)	115.78 (99.77)	893.68 (12.63)	39.17 (0.84)
EDDM	0.10 (0.04)	0.83 (0.17)	0.18 (0.07)	96.23 (84.15)	1465.38 (4.53)	35.50 (0.69)
EWMA	0.09 (0.03)	**0.84 (0.16)**	0.17 (0.06)	89.96 (88.83)	958.84 (1.72)	41.77 (0.72)
FHDDM	0.68 (0.25)	0.73 (0.24)	0.69 (0.25)	137.00 (98.44)	1740.28 (12.36)	38.02 (0.88)
FHDDMS	0.68 (0.22)	0.75 (0.22)	0.70 (0.22)	126.41 (102.09)	1918.56 (3.69)	53.98 (1.07)
FHDDMS$_{add}$	0.67 (0.23)	0.73 (0.23)	0.69 (0.23)	136.71 (99.32)	1223.52 (1.89)	41.36 (0.56)
HDDM$_A$	0.61 (0.20)	0.76 (0.23)	0.66 (0.20)	115.27 (103.69)	1257.18 (26.59)	52.48 (0.94)
HDDM$_W$	0.69 (0.22)	0.75 (0.22)	0.71 (0.22)	125.38 (102.07)	2518.44 (2.18)	54.51 (0.94)
MDDM$_A$	0.68 (0.21)	0.76 (0.22)	0.70 (0.21)	127.22 (96.81)	1973.48 (7.54)	93.39 (1.44)
MDDM$_E$	0.68 (0.22)	0.75 (0.21)	0.70 (0.22)	128.25 (100.73)	1964.82 (9.90)	68.95 (1.83)
MDDM$_G$	0.69 (0.22)	0.75 (0.22)	0.71 (0.22)	129.19 (101.69)	1965.44 (7.01)	71.94 (1.87)
Null	0.50 (0.00)	0.40 (0.00)	0.44 (0.00)	255.00 (0.00)	**320.00 (0.00)**	**35.10 (0.70)**
PH	0.49 (0.24)	0.67 (0.23)	0.54 (0.23)	205.31 (60.13)	960.00 (0.00)	37.18 (0.71)
RDDM	0.30 (0.12)	0.80 (0.20)	0.43 (0.15)	95.28 (100.18)	64014.80 (14.42)	41.91 (0.77)
SeqDrift2	0.56 (0.27)	0.83 (0.19)	0.64 (0.25)	210.20 (21.25)	41227.08 (10343.85)	47.75 (1.09)

- **MIXED** Each instance is a mix of two binary attributes $w, v \in \{0, 1\}$, and two real-valued attributes $x_1, x_2 \sim U[0, 1]$. Labels are assigned according to $y = 1[v \wedge w \wedge y < 0.5 + 0.3 \sin(3\pi x)]$. After each context change the classification is reversed.

- **SEA** Each instance consists of three attributes $x_1, x_2, x_3 \sim U[0, 10]$. Each concept consists of a threshold θ, such that $y = 1[x_1 + x_2 + x_3 > \theta]$. That is, if the binary label denotes whether the sum of the attributes exceeds the threshold. Thresholds cycle between the values $[8, 9, 7, 9.5]$.

- **SINE1** Each instance consists of two attributes $x_1, x_2 \sim U[0, 1]$. Binary labels are given by $y = 1[x_1 > sin(x_2)]$. When concept drift occurs, the labels are reversed.

- **SINE2** As with SINE1, the attributes are $x_1, x_2 \sim U[0, 1]$. Binary labels are given by $y = 1[x_1 > 0.5 + 0.3 \sin(3\pi x_2)]$. When concept drift occurs, the labels are reversed.

- **STAGGER** Instances consist of categorical attributes $size \in [small, medium, large]$, $color \in [red, green]$, $shape \in [circular, non - circular]$. Each concepts is a first order logic expression. Specifically, the following concepts are cycled: $y = 1[size = small \wedge color = red]$, $y = 1[color = green \vee shape = circular]$, and $y = 1[size = medium \vee size = large]$.

These data streams provide constitute the most popular benchmarks used in the concept drift detection literature. Each of these data streams has the following parameters:

- **Concept Length** The number of instances between concept drifts. If this number is small then the data stream is *volatile*, if this number is small then the data stream is *stable*.

- **Transition Length** The number of instances over which a concept drift occurs. If the transition length is n, and there have been i instances since the concept drift, then the probability that the new concept is used is $\Pr(new - concept) = i/n$, otherwise the previous concept is used. If the transition length is low then the drift is *abrupt*. Otherwise the stream is *gradual*.

- **Noise Rate** The rate at which any given label will be replaced with a different label. For binary labels, the label is simply inverted. Otherwise, a different label is chosen at random. If this quantity is high, then the data stream is *noisy*.

We are interested in how drift detector performance varies with noise and transition length. We therefore run experiments using each of the variations of the above datasets

- **High Noise** Noise rate is set to 0.4, transition length is set to 50 and concept length is set to 1000.

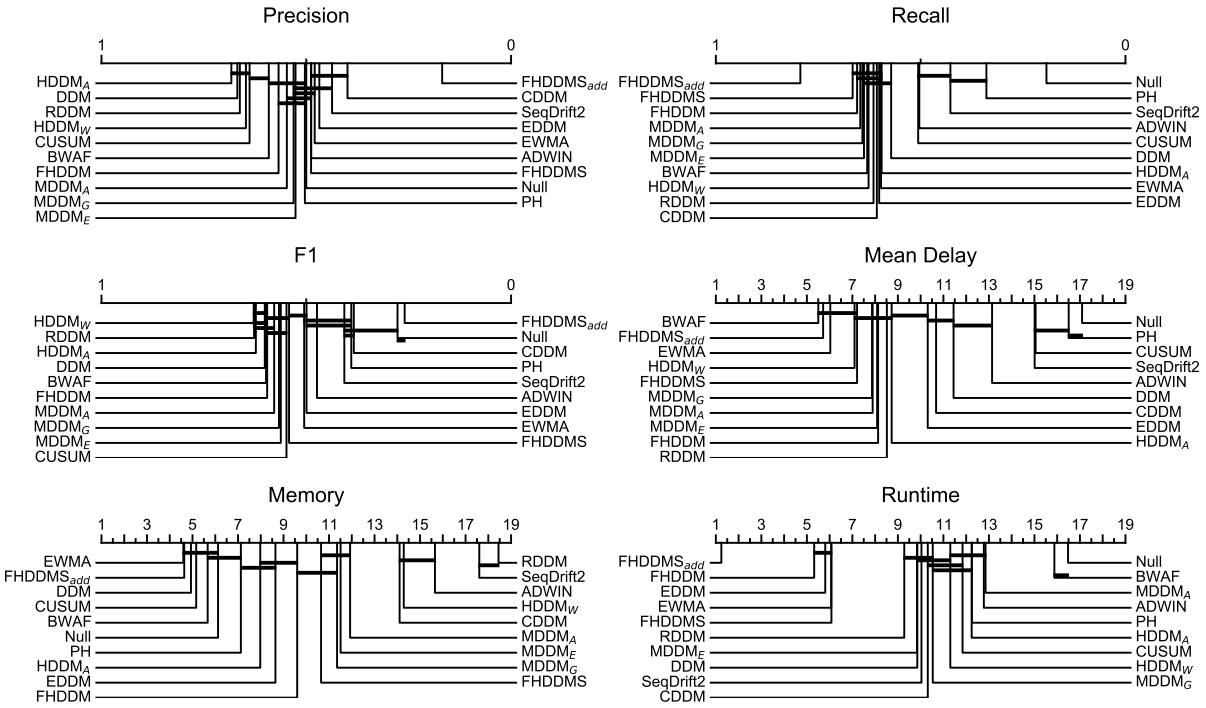


Figure 6.4: CD diagram of benchmark datasets.

- **Low Noise** Noise rate is set to 0.02, transition length is set to 50 and concept length is set to 1000.
- **Gradual Drift** Transition length is set to 250, noise rate is set to 0.1, and concept length is set to 1000.
- **Abrupt Drift** Transition length is set to 0, noise rate is set to 0.1, and concept length is set to 1000.
- **Long concepts** Transition length is set to 50, noise rate is set to 0.1, and concept length is set to 25000.
- **Short concepts** Transition length is set to 50, noise rate is set to 0.1, and concept length is set to 250.

We run each datastream with two base learners. Naïve Bayes and Hoeffding trees [19] are two of the most popular, so we use these.

We run two iterations of each of the 7 data streams, for each of the four variations, with both of the base learners. The results are given in Table 6.2 and visualised in Figure 6.4.

We see that BWAF performs well on benchmark datasets. It achieves the fifth highest precision (significantly less than the best detector at $p < 0.05$), the seventh best recall (not significantly less than the best detector at $p < 0.05$), the fourth best F_1 score (not

Table 6.2: Results of benchmark datasets.

Detector	Precision	Recall	F1	Mean Delay	Memory (bytes)	Runtime (ms)
ADWIN	0.49 (0.20)	0.50 (0.26)	0.47 (0.23)	173.56 (76.56)	59.78 (73.15)	60.27 (62.47)
BWAF	0.59 (0.17)	0.63 (0.23)	0.60 (0.19)	105.20 (84.93)	52.56 (76.53)	86.75 (109.54)
CDDM	0.40 (0.27)	0.61 (0.27)	0.38 (0.27)	132.92 (93.41)	78.19 (141.79)	81.65 (179.35)
CUSUM	0.64 (0.18)	0.51 (0.28)	0.55 (0.24)	194.15 (61.85)	68.20 (144.00)	84.61 (133.74)
DDM	0.67 (0.16)	0.57 (0.27)	0.60 (0.23)	149.43 (81.68)	61.87 (113.65)	78.30 (121.63)
EDDM	0.47 (0.20)	0.60 (0.25)	0.50 (0.21)	141.65 (76.42)	60.95 (115.77)	64.53 (111.05)
EWMA	0.48 (0.18)	0.60 (0.23)	0.51 (0.18)	112.58 (84.33)	47.32 (45.72)	70.33 (168.48)
FHDDM	0.57 (0.18)	0.66 (0.21)	0.60 (0.18)	114.97 (67.17)	54.17 (89.55)	49.37 (92.55)
FHDDMS	0.49 (0.20)	0.67 (0.19)	0.54 (0.18)	110.00 (64.19)	48.35 (49.73)	44.40 (80.51)
FHDDMS$_{add}$	0.17 (0.12)	**0.79 (0.06)**	0.26 (0.14)	**76.39 (37.31)**	**41.05 (41.47)**	**7.59 (13.95)**
HDDM$_A$	**0.68 (0.16)**	0.60 (0.26)	0.62 (0.23)	129.86 (88.14)	59.99 (108.88)	72.75 (103.00)
HDDM$_W$	0.65 (0.19)	0.63 (0.23)	**0.63 (0.21)**	116.21 (82.71)	60.97 (109.85)	65.47 (102.01)
MDDM$_A$	0.55 (0.18)	0.65 (0.20)	0.58 (0.17)	115.83 (65.88)	52.02 (80.13)	62.35 (94.20)
MDDM$_E$	0.53 (0.19)	0.64 (0.21)	0.56 (0.18)	118.31 (69.11)	47.42 (44.77)	54.25 (88.08)
MDDM$_G$	0.53 (0.19)	0.64 (0.20)	0.57 (0.18)	117.62 (66.90)	47.27 (44.61)	54.95 (88.30)
Null	0.50 (0.00)	0.19 (0.03)	0.28 (0.03)	249.98 (0.15)	84.39 (159.77)	200.90 (383.73)
PH	0.50 (0.19)	0.34 (0.23)	0.39 (0.21)	231.80 (31.77)	75.09 (150.22)	122.89 (231.97)
RDDM	0.66 (0.18)	0.62 (0.25)	0.63 (0.22)	127.33 (84.36)	113.81 (83.46)	66.14 (104.59)
SeqDrift2	0.44 (0.20)	0.43 (0.24)	0.41 (0.20)	204.78 (51.88)	117.58 (155.47)	56.49 (64.58)

significantly less than the best detector at $p < 0.05$), the best mean delay (although not significantly greater than the second best at $p < 0.05$), the fifth best memory consumption (not significantly worse than the best detector at $p < 0.05$), but the second worst runtime.

By contrast, the performance of CDDM is poor. Because the instances in these data streams are high dimensional, the learners are uncalibrated, triggering a high rate of false positives. On recall and memory, CDDM is ranked in the middle the detectors. On all other metrics it is in the lower half of detectors. This affirms the statement in Section 4.5 that CDDM must be extended with online calibration for practical usage.

6.4 Triage Simulation

In this section we evaluate our novel concept drift detectors in the motivating example domain of GP referrals triage. Due to a lack of real triage data annotated by concept drift, we instead use a synthetic data stream with concept drift deliberately introduced.

We base our synthetic data stream on the MIMIC-III dataset - a publicly available repository of free-text electronic health records [35]. There are several types of documents within the dataset, so we limit our usage to 'radiology' documents, which have the advantage of having structured headers. We preprocess the 522,279 radiology documents by converting the text into bag-of-words format. To keep the instance dimensionality reasonable, we eliminate tokens which occur in fewer than 40% or more than 60% of documents, resulting in 59,652 dimensional features. We shuffle the order of the documents to make sure the only concept drifts which occurs in the data stream have been deliberately engineered.

To simulate a triage rule, we randomly label 30 referral documents with priority labels 1 to 4. We then train a decision tree on these instance-label pairs. We repeat this several times to obtain a set of "triage concepts". Concept drift is simulated by labelling the instances prior to the drift point using one triage concept, and then labelling the instances after the drift with another, randomly selected concept. In each instantiation of the data stream a single concept drift occurs halfway through the stream.

We run two iterations of the medical triage data stream under each of the variations described in Section 6.3, namely high noise, low noise, gradual drift, abrupt drift, long concepts, short concepts. The results are given in Table 6.3 and visualised in Figure 6.5.

Similar to Section 6.3, we see competitive performance from BWAF. It achieves the second highest precision, recall, and F_1 score, as well as the third best mean delay (none of which is significantly worse than the best detector at $p < 0.05$). The memory consumption is sixth best (not significantly less than the best detector at $p < 0.05$), although the runtime is fifth worst.

As in Section 6.3, the performance of CDDM is poor. CDDM achieved the worst

Table 6.3: Results of synthetic triage data streams.

Detector	Precision	Recall	F1	Mean Delay	Memory (bytes)	Runtime (ms)
ADWIN	0.45 (0.13)	0.52 (0.17)	0.48 (0.14)	184.53 (72.04)	2071.95 (2703.76)	1188.44 (1201.33)
BWAF	0.61 (0.11)	0.64 (0.09)	0.61 (0.09)	64.92 (70.37)	2031.33 (3074.38)	1074.40 (890.51)
CDDM	0.11 (0.19)	0.60 (0.13)	0.11 (0.14)	65.22 (97.95)	1202.42 (1182.93)	326.97 (790.04)
CUSUM	0.50 (0.14)	0.44 (0.16)	0.47 (0.14)	216.06 (50.18)	2180.51 (3042.57)	1273.14 (1099.90)
DDM	0.57 (0.13)	0.55 (0.16)	0.56 (0.15)	176.42 (64.35)	2212.87 (3177.23)	1182.96 (1092.59)
EDDM	0.38 (0.18)	0.48 (0.17)	0.41 (0.16)	191.56 (79.70)	2017.23 (2741.34)	1047.99 (985.13)
EWMA	0.47 (0.14)	0.52 (0.17)	0.48 (0.13)	136.42 (110.62)	1790.92 (2524.41)	1038.35 (982.84)
FHDDM	0.33 (0.14)	0.60 (0.13)	0.41 (0.12)	100.25 (87.19)	1594.83 (1881.50)	489.60 (381.47)
FHDDMS	0.21 (0.11)	0.60 (0.13)	0.30 (0.11)	97.19 (90.45)	1208.86 (976.79)	292.01 (222.99)
FHDDMS_{add}	0.16 (0.10)	**0.67 (0.00)**	0.25 (0.11)	**62.39 (51.61)**	**1122.88 (841.49)**	**156.54 (110.67)**
HDDM_A	0.60 (0.11)	0.56 (0.15)	0.58 (0.13)	122.89 (94.24)	2178.58 (3180.54)	1217.33 (1110.18)
HDDM_W	0.44 (0.11)	0.59 (0.14)	0.50 (0.11)	83.36 (99.85)	1987.08 (2840.06)	922.85 (872.30)
MDDM_A	0.30 (0.14)	0.57 (0.15)	0.38 (0.13)	105.22 (94.68)	1586.03 (1860.45)	490.82 (378.84)
MDDM_E	0.30 (0.13)	0.58 (0.14)	0.38 (0.12)	103.72 (93.50)	1561.36 (1640.40)	462.48 (340.40)
MDDM_G	0.31 (0.12)	0.60 (0.13)	0.39 (0.12)	93.47 (89.14)	1565.05 (1858.74)	473.55 (349.91)
Null	0.50 (0.00)	0.33 (0.00)	0.40 (0.00)	250.00 (0.00)	2156.45 (3465.22)	2181.42 (2075.25)
PH	0.44 (0.10)	0.35 (0.08)	0.39 (0.07)	244.67 (21.99)	2112.62 (3124.96)	1594.42 (1440.53)
RDDM	**0.63 (0.07)**	0.63 (0.10)	**0.63 (0.09)**	116.64 (66.99)	2117.44 (3076.81)	1078.29 (993.25)
SeqDrift2	0.46 (0.11)	0.46 (0.16)	0.45 (0.12)	208.89 (58.49)	1994.71 (2702.04)	1440.93 (1548.25)

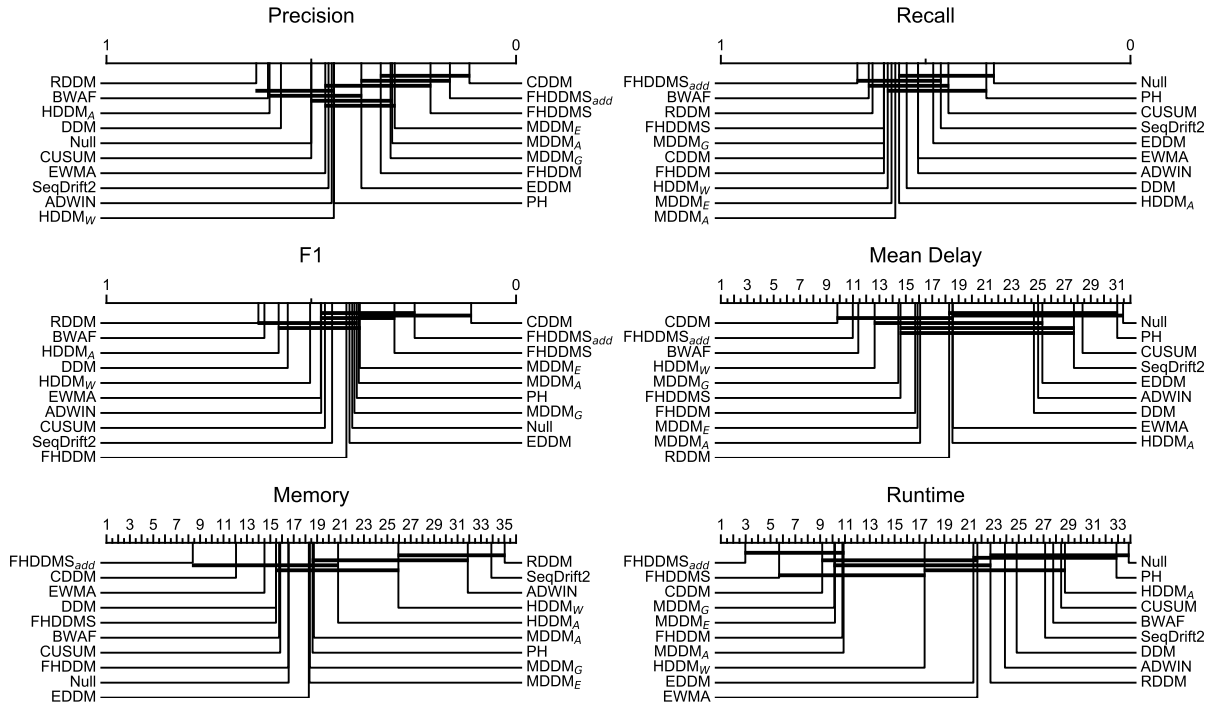


Figure 6.5: CD diagram of synthetic triage data streams.

precision and F_1 of all the detectors, although it achieved the sixth best recall, the best mean delay, the second best memory consumption, and the third lowest runtime. As before, this poor performance can be attributed to the models being miscalibrated, a problem which is especially acute given the high dimensionality of the data.

6.5 Conclusion

In this chapter we experimentally validated our novel drift detectors, and investigated the performance of drift detectors on a synthetic medical triage dataset. We have seen that CDDM can achieve leading precision F_1 and precision scores on Bernoulli streams with feature drift and uneven noise. We have also seen that BWAF is a competitive drift detector on a battery of benchmark data streams, and a synthetic GP referrals triage data stream. It consistently achieved one of the best precision, recall, F_1, detection delays scores. BWAF's memory usage reasonable, consistently within the top quartile, although its runtime was generally not competitive. CDDM did not perform competitively in these experiments. We attribute this to the models being uncalibrated, thus affirming the claim made in Section 4.5 that CDDM should be combined with online calibration for practical usage.

7
Conclusion

In this chapter we summarise our work and promising future work. Section 7.1 highlights major achievements of this thesis. Section 7.2 identifies the main limitations of our work. Section 7.3 discusses the most promising future work.

7.1 Achievements

The following list highlights the major achievements of our research:
Chapter 3

- We introduced multiple drift detector (MDD), a framework for monitoring for several types of concept drift, specifically real drift, label drift, and feature drift.
- This allows users to receive early warnings that a model is no longer fit for purpose due to changes in the instance distribution, so that the model can be recalled if necessary.
- MDD is flexible and can be implemented using any existing drift detection method.
- We also introduced a graphical interface for visualising the evolution of the data stream.

Chapter 4

- We introduced calibrated drift detection method (CDDM), a drift detector which only detects reducible, and not irreducible, performance degradation.
- CDDM is thus able to avoid false positives due to feature drift which do not change the decision boundary. In these cases, retraining the model is at best pointless and at worst detrimental due to reducing the size of the training data.
- We demonstrated the value of CDDM by achieving the highest F_1 score on a Bernoulli data stream with feature drift and variation in prediction difficulty.

Chapter 5

- We introduced Bayesian drift detection method (BDDM), a method for exactly calculating posterior probabilities of drift timing and performance degradation.
- This allows user to make more rational decisions about model retraining, based on expected utility.
- BDDM is able to accommodate uneven time series, by factoring the spaces between instances into prior values.
- We also introduce beta with adaptive forgetfulness (BWAF), an efficient heuristic approximation of BDDM. BWAF requires only four registers of memory, and has $O(1)$ computational complexity. It also does not require parameter selection.
- BWAF was shown to be competitive on standard benchmarks, achieving the second highest F_1 score. It was also competitive on the synthetic triage data stream, again achieving the second highest F_1 score.

7.2 Limitations

We identify the following major limitations of this work.

7.2.1 Multiple Drift Detector

The purpose of this system was to provide early warnings of feature drift when label values are delayed compared to feature values. However, there are limitations to the types of feature drifts which can be detected.

By representing free-text features as bags of words, all information about the arrangement of tokens is lost, and so only changes at the token-frequency level are detectable. MDD also assumes independence between features. Thus changes in the correlation between features are undetectable by MDD.

7.2.2 Calibrated Drift Detection Method

In Chapter 4, we identified two limitations of CDDM. The first was that it cannot detect "small" deviations from miscalibration. The minimum detectable deviation depends on the window size and the detection threshold, as specified in Equation 4.18.

The second limitation is that CDDM requires its model to be calibrated. Most machine learning methods require post-processing of probabilistic predictions to become calibrated [45], and these can require large amounts of training data to achieve [38] and are not necessarily robust against dataset drift [43]. CDDM will therefore not be practical until progress has been made on the problem of online calibration.

7.2.3 Bayesian Concept Drift Detection

The main limitation of BDDM is that it scales poorly, being $O(t^2)$ in time and $O(t)$ in space. It is thus impractical for high-volume data streams.

The main limitation of BWAF is that the heuristics it uses to approximate BDDM are not well motivated theoretically. It is not known if they introduce biases or inefficiencies.

7.3 Future Work

We suggest the follow as promising directions for future work.

7.3.1 User Testing

In Chapter 1, we motivated our work as bridging a gap between academic research on concept drift and real data science applications. We used GP referrals triage as a motivating example for this discussion. However, we have not yet shown that our algorithms actually help users in real applications.

It would therefore be valuable to evaluate our algorithms and frameworks with user tests and case studies. Are users able to effectively interpret the early warnings from MDD? How often do the kinds of false positives and false negatives that CDDM prevents come up in real data streams? We have argued that the probability distributions provided by BDDM and BWAF will help experts make expected utility calculations, but are real application domains to well-specified enough for this to be possible?

7.3.2 Online Calibration

CDDM assumes that models are calibrated, which is not true of most learning algorithms. Calibration can be achieved off-line via techniques such as calibration maps [38][45], but these can require large samples of data [38].

It would therefore be beneficial to investigate avenues by which CDDM could be applied to models which are not automatically calibrated. One way to achieve this is via calibrating learners online, an area which has not been well studied.

7.3.3 Bayesian Drift Detection

BWAF has shown promise as a drift detector, achieving the second highest F_1 score of the detectors tested in the benchmark data streams. It would be worthwhile conducting further theoretical and experimental investigations into BWAF's performance.

Specifically, it would be worth investigating whether it is possible to prove theoretical guarantees on the false positive and false negative rate of BWAF. Additionally, experimentally determining under what circumstances BWAF tends to perform well or perform poorly. Can the computational efficiency of BWAF be significantly improved? Are there other heuristic approaches to Bayesian drift detection which achieve even better performance than BWAF?

Bibliography

[1] A Deep Learning Platform for GP Referral Triage. `hhttps://precisiondrivenhealth.com/a-deep-learning-platform-for-gp-referral-triage/`. Accessed: 2020-06-30.

[2] Dash overview. `https://plotly.com/dash/`. Accessed: 2020-07-10.

[3] *Referral Triage Guidelines*, 2018.

[4] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[5] Daniel K Antwi, Herna L Viktor, and Nathalie Japkowicz. The perfsim algorithm for concept drift detection in imbalanced data. In *2012 IEEE 12th International Conference on Data Mining Workshops*, pages 619–628. IEEE, 2012.

[6] Stephen Bach and Mark Maloof. A bayesian approach to concept drift. In *Advances in neural information processing systems*, pages 127–135, 2010.

[7] Stephen H Bach and Marcus A Maloof. Paired learners for concept drift. In *2008 Eighth IEEE International Conference on Data Mining*, pages 23–32. IEEE, 2008.

[8] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86, 2006.

[9] Roberto SM Barros, Danilo RL Cabral, Paulo M Gonçalves Jr, and Silas GTC Santos. Rddm: Reactive drift detection method. *Expert Systems with Applications*, 90:344–355, 2017.

[10] Roberto Souto Maior Barros and Silas Garrido T Carvalho Santos. A large-scale comparison of concept drift detectors. *Information Sciences*, 451:348–370, 2018.

[11] Daniel Barry and John A Hartigan. Product partition models for change point problems. *The Annals of Statistics*, pages 260–279, 1992.

[12] et al. Bernstein, Ian. *Musculoskeletal Services; Supporting Self-management Pharmacological Management Triage Specification Pathway Referral Criteria*, 2015.

[13] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007.

[14] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 299–310. Springer, 2010.

[15] Roberto Souto Maior de Barros, Juan Isidro González Hidalgo, and Danilo Rafael de Lima Cabral. Wilcoxon rank sum test drift detector. *Neurocomputing*, 275:1954–1963, 2018.

[16] Danilo Rafael de Lima Cabral and Roberto Souto Maior de Barros. Concept drift detection based on fishers exact test. *Information Sciences*, 442:220–234, 2018.

[17] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[18] Shushank Dogra and Narinder Sharma. Comparison of different techniques to design of filter. *International Journal of Computer Applications*, 97(1), 2014.

[19] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[20] Dhivya Eswaran and Christos Faloutsos. Sedanspot: Detecting anomalies in edge streams. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 953–958. IEEE, 2018.

[21] Paul Fearnhead. Exact and efficient bayesian inference for multiple changepoint problems. *Statistics and computing*, 16(2):203–213, 2006.

[22] Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé Caballero-Mota. Online and non-parametric drift detection methods based on hoeffdings bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2014.

[23] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295. Springer, 2004.

[24] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[26] Spencer Greenberg. Calibration scoring rules for practical prediction training. *arXiv preprint arXiv:1808.07501*, 2018.

[27] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

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

[29] Juan Isidro González Hidalgo, Laura Maria Palomino Mariño, and Roberto Souto Maior de Barros. Cosine similarity drift detector. In *International Conference on Artificial Neural Networks*, pages 669–685. Springer, 2019.

[30] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[31] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[32] David Tse Jung Huang, Yun Sing Koh, Gillian Dobbie, and Russel Pears. Detecting volatility shift in data streams. In *2014 IEEE International Conference on Data Mining*, pages 863–868. IEEE, 2014.

[33] Hamish Huggard, Yun Sing Koh, Patricia Riddle, and Gustavo Olivares. Predicting air quality from low-cost sensor measurements. In *Australasian Conference on Data Mining*, pages 94–106. Springer, 2018.

[34] Ashish K Jha. Meaningful use of electronic health records: the road ahead. *Jama*, 304(15):1709–1710, 2010.

[35] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[36] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *ICML*, pages 487–494, 2000.

[37] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. *Learning for text categorization*, pages 33–40, 1998.

[38] Meelis Kull, Telmo M Silva Filho, Peter Flach, et al. Beyond sigmoids: How to obtain well-calibrated probabilities from binary classifiers with beta calibration. *Electronic Journal of Statistics*, 11(2):5052–5080, 2017.

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

[40] Anjin Liu, Yiliao Song, Guangquan Zhang, and Jie Lu. Regional concept drift detection and density synchronized drift adaptation. In *IJCAI International Joint Conference on Artificial Intelligence*, 2017.

[41] Alexandra Lheureux, Katarina Grolinger, Hany F Elyamany, and Miriam AM Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 5:7776–7797, 2017.

[42] Thomas M Mitchell et al. Machine learning, 1997.

[43] Jose G Moreno-Torres, Troy Raeder, RocíO Alaiz-RodríGuez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530, 2012.

[44] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[45] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 625–632. ACM, 2005.

[46] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *International conference on discovery science*, pages 264–269. Springer, 2007.

[47] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[48] Russel Pears, Sripirakas Sakthithasan, and Yun Sing Koh. Detecting concept change in dynamic data streams. *Machine Learning*, 97(3):259–293, 2014.

[49] Ali Pesaranghader. *A Reservoir of Adaptive Algorithms for Online Learning from Evolving Data Streams*. PhD thesis, University of Ottawa, 2018.

[50] Ali Pesaranghader and Herna L Viktor. Fast hoeffding drift detection method for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 96–111. Springer, 2016.

[51] Ali Pesaranghader, Herna L Viktor, and Eric Paquet. Mcdiarmid drift detection methods for evolving data streams. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2018.

[52] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[53] SW Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, 1959.

[54] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters*, 33(2):191–198, 2012.

[55] Elsom S. Sands, N. and R. Colgate. *UK Mental Health Triage Scale Guidelines*, 2015.

[56] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[57] Parinaz Sobhani and Hamid Beigy. New drift detection method for data streams. In *International conference on adaptive and intelligent systems*, pages 88–97. Springer, 2011.

[58] Philip E Tetlock and Dan Gardner. *Superforecasting: The art and science of prediction*. Random House, 2016.

[59] Abraham Wald. *Sequential analysis*. Courier Corporation, 2004.

[60] Heng Wang and Zubin Abraham. Concept drift detection for streaming data. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2015.

[61] Shuo Wang, Leandro L Minku, Davide Ghezzi, Daniele Caltabiano, Peter Tino, and Xin Yao. Concept drift detection for online class imbalance learning. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2013.

[62] Shuo Wang, Leandro L Minku, and Xin Yao. A learning framework for online class imbalance learning. In *2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*, pages 36–45. IEEE, 2013.

[63] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[64] Gerhard Widmer and Miroslav Kubat. Learning flexible concepts from streams of examples: Flora2. 1992.

[65] Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *European Conference on Machine Learning*, pages 227–243. Springer, 1993.

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

[67] Pranjul Yadav, Michael Steinbach, Vipin Kumar, and Gyorgy Simon. Mining electronic health records (ehrs) a survey. *ACM Computing Surveys (CSUR)*, 50(6):1–40, 2018.

[68] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Icml*, volume 1, pages 609–616. Citeseer, 2001.

[69] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699, 2002.