

Data stream mining: methods and challenges for handling concept drift.

WARES, S., ISAACS, J. and ELYAN, E.

2019



Review Paper

Data stream mining: methods and challenges for handling concept drift



Scott Wares¹  · John Isaacs¹ · Eyad Elyan¹

© The Author(s) 2019 **OPEN**

Abstract

Mining and analysing streaming data is crucial for many applications, and this area of research has gained extensive attention over the past decade. However, there are several inherent problems that continue to challenge the hardware and the state-of-the-art algorithmic solutions. Examples of such problems include the unbound size, varying speed and unknown data characteristics of arriving instances from a data stream. The aim of this research is to portray key challenges faced by algorithmic solutions for stream mining, particularly focusing on the prevalent issue of concept drift. A comprehensive discussion of concept drift and its inherent data challenges in the context of stream mining is presented, as is a critical, in-depth review of relevant literature. Current issues with the evaluative procedure for concept drift detectors is also explored, highlighting problems such as a lack of established base datasets and the impact of temporal dependence on concept drift detection. By exposing gaps in the current literature, this study suggests recommendations for future research which should aid in the progression of stream mining and concept drift detection algorithms.

Keywords Stream · Data · Mining · Concept drift

1 Introduction

The global datasphere is a notional environment in which data produced worldwide is contained. According to Reinsel et al. [62], the global datasphere contained over 20 ZB of data (20 billion TB) in 2017. Its growth is expected to continue rapidly, and by 2025 it is hypothesised to contain 160 ZB of data. Such exponential growth is due, in part, to hardware developments and an increase in user availability and accessibility.

Data is now generated at a near constant and limitless rate, resulting from an array of devices, networks and everyday tasks such as credit card transactions and mobile phones [1]. Data arriving online and in a sequential, continuous fashion is known as a data stream. These data streams provide a potential source of valuable quantitative and qualitative data, providing it can be extracted in a timely manner.

Adapted machine learning techniques can be used to harvest interesting information from the stream in a process known as stream mining. However, the volume, velocity and temporal nature of the arriving data can cause complex challenges for stream mining. Unlike stationary data in traditional batch learning scenarios, streaming data can be looked at only once and is unbound in size. This is an inherent problem for streaming data and poses a challenge for real time processing.

Concept drift is another inherent problem, caused by streaming data changing, or evolving, over time. Concept drift also occurs at varying rates of severity. A shift in the underlying distribution can result in the feature vectors of arriving instances no longer reflecting the class label. This causes a negative impact upon the reliability and accuracy of classifiers that make predictions using the current distribution.

✉ Scott Wares, s.wares1@rgu.ac.uk; John Isaacs, j.p.isaacs@rgu.ac.uk; Eyad Elyan, e.elyan@rgu.ac.uk | ¹The Sir Ian Wood Building, Robert Gordon University, Garthdee Road, Aberdeen, Scotland, UK.



SN Applied Sciences

(2019) 1:1412

| <https://doi.org/10.1007/s42452-019-1433-0>

Received: 10 July 2019 / Accepted: 9 October 2019

Published online: 15 October 2019

SN Applied Sciences
A **SPRINGER NATURE** journal

This paper provides a critical review of stream mining with a focus on stream mining challenges, concept drift detection algorithms and problems with their evaluative process such as a lack of established datasets. This research is primarily concerned with supervised data stream mining and drift detection methods. Literature and techniques involving unsupervised approaches, such as Sethi and Kantardzic [64] and de Mello et al. [23], are not reviewed. Unlike existing reviews such as Gama et al. [36], this paper reviews modern, recent approaches to handling concept drift, as well as established methods covered in earlier reviews. Ditzler et al. [26] provides a summary of the challenges and approaches for learning in both static and evolving data streams. However, their review lacks detail and depth in their explanation of techniques and instead. Similarly, the recent review by Krawczyk et al. [48] primarily focuses on ensemble methods, and only briefly highlights the most popular non-ensemble based techniques. The timeliness of this review is of particular interest. Successful mining of data streams can potentially provide rich quantitative and qualitative information. Such information could have a tremendous impact on business practices across various industries, such as oil and gas where streaming data is abundant. However, because of the challenges posed by stream mining it is relatively unharnessed. This research suggests that the reason for stream mining not be fully harnessed is that the domain of concept drift detection is not progressing at a quick enough pace. For example, the domain of imbalanced data, in the context of stream mining, has produced various adaptations for algorithms to handle class imbalance, such as DDM-OCI [69], LFR [67], Learn++.NIE [24] and Learn++.CDS [25]. The field of class imbalance in stream mining has also published recent reviews that provide timely suggestions for future research [47], whereas in contrast the reviews for concept drift are several years old [36] and do not encapsulate recent advancements.

The main contributions of this paper are outlined as follows:

1. Critically and extensively review existing literature, discussing both well established and recent concept drift detection techniques.
2. Categorise existing methods for concept drift detection based on their underlying model. Existing methods are categorised as either statistical based, window based or ensemble based.
3. Identify and discuss clear gaps in the literature areas for improvement, particularly relating to current evaluative measures and the availability of public datasets for benchmarking.
4. Provide possible avenues for future research that address current research gaps in the existing literature.

This paper is structured as follows. Section 2 provides a general overview of stream mining, its key differences from traditional batch learning and design requirements for stream mining algorithms. Section 3 focuses on concept drift, providing definitions and discussing algorithmic requirements. Section 4 categorises and critically reviews existing concept drift detection techniques. Section 5 discusses evaluation methods, as well as highlighting the concept of temporal dependence. Section 6 concludes the paper and offers suggestion for future research direction.

2 Stream mining overview

Data streams possess specific and unique characteristics that differentiate them from other forms of data. In traditional machine learning contexts, the data is often referred to as “batch” data. That is, all of the data is immediately available in its entirety and is stored in memory. This is of stark contrast to stream mining, where data streams produce elements in a sequential, continuous fashion, and may also be impermanent, or transient, in nature [2, 33]. This means stream data may only be available for a short time. The difference between traditional methods and data streams is described by Babcock et al. [2] in the following ways:

1. Data elements in the stream arrive online.
2. The system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams.
3. Data streams are potentially unbound in size.
4. Once an element from a data stream has been processed, it is discarded or archived. It cannot be retrieved easily unless it is explicitly stored in memory, which is small relative to the size of data streams.

The unique characteristics of a data stream contribute to the challenges in processing its arriving elements. Since batch data is persistent, it can be queried once in its entirety and individual data elements can be accessed at random. Data streams however, since transient, must be queried continuously by the algorithm. The data elements in the stream cannot be accessed at random; they can only be accessed in the sequence in which they arrive from the stream. The key differences between processing traditional batch data and stream data are shown Table 1.

Data streams are either *static* (sometimes referred to as stationary) or *evolving*, and are classified depending on the condition of their core distribution.

Static data has an underlying distribution that does not shift over time. That is, the features that define the target label for learning remain constant and consistent. Static

Table 1 Batch data versus streaming data

Batch data	Stream data
Offline	Real time
Slow data generation	Rapid data generation
Persistent data	Transient data
Process entire data	Process samples of data
Constant availability	Limited availability
Complex techniques used if required	Linear techniques widely used
Fixed size	Unbound in size
Random access	Sequential access
Known data characteristics	Unpredictable data characteristics

datasets are frequent in traditional machine learning contexts where features defining ground truth labels do not change [18].

Unlike static data, the distribution for an evolving data stream may change over time. Feature vectors may change over a time period t such that they no longer reflect the class label. Aggarwal [1] describes how data streams possess an inherent temporal component that causes them to be time dependent by nature. Consider customer purchasing behaviour as an example. A model trained to predict weekly sales for a clothing store might use attributes such as advertising costs, promotions and customer footfall. However, this model may become less accurate over time, perhaps due to seasonality. The shift in distribution over time is commonly known as *concept drift*.

In the domain of machine learning, traditional applications implement batch learning techniques on static datasets. Batch learning approaches involve having the entirety of the training data available at any given time. The data can be processed once or multiple times before an algorithm produces an output decision.

Data streams by nature are incompatible to batch learning for a number of reasons. Most obviously where traditional applications have all of the data available immediately, data streams must be mined in a distributed fashion since examples arrive continuously and in a sequential manner. In contrast to batch and multi-pass learning, stream mining algorithms must be designed to work with one pass of the data only [1].

A solution to the ineffectiveness of batch learning may at first seem obvious; translate batch learning algorithms into one-pass variants. However, the innate temporal nature of data streams may render this solution redundant as a one-pass conversion approach may not consider the evolution of the underlying distribution. Concept drift is something stream mining algorithms simply must take into consideration.

Since data streams are unbound in size, the volume and velocity can result in the imposition of hardware limitations. The most obvious of which is memory; it is not feasible to continuously explicitly store stream elements since their maximum size is almost always unknown. A second hardware limitation is processing resource. The speed at which elements arrive from a stream, as well as the stream size, can quickly consume available resources. As such, stream mining algorithms should be both computationally fast and lightweight [18].

3 Concept drift

Traditional machine learning algorithms operate with the assumption that the data distribution is static. For data streams, the distribution of arriving examples may change over time due to the stream's innate temporal nature. This renders traditional batch learning algorithms unsuitable for applications that learn from data streams.

In reality, data streams produce copious amounts of data at a near constant rate. Gama [33] provides examples of such streams, including surveillance systems, sensor networks and telecommunication systems. However, with recent modern hardware developments these streams are produced by new devices such as smart household appliances and car navigation systems. Algorithms that seek to learn from data streams must be able to accurately model the underlying distribution. The ability to detect and adapt to changes in the distribution of examples is paramount for data stream mining algorithms.

The shift in the underlying distribution of examples arriving from a data stream is referred to as concept drift. Concept drift occurs over time and the rate at which the drifts occurs varies. It can be responsible for various symptoms, including previous examples to become irrelevant; their distribution no longer accurately reflects their corresponding class label. This is reflected in the clothing store sales prediction example previously mentioned. If seasonality causes clothing sales to be higher in the summer months, then examples from winter months may not accurately reflect class labels. It may be the case that the model predicts low sales, but due to seasonality there are less people shopping and the sales are in fact high for a winter season. As such, models must be capable of forgetting previous examples once concept drift has occurred.

3.1 Definition

A learning algorithm observing examples with a stationary distribution would observe training examples in the form (\mathbf{x}_i, y_i) , where \mathbf{x}_i is the feature vector and $y_i \in \{c_1, c_2, \dots, c_n\}$ for the i th example. A class prediction

at a particular time stamp t would be given as \hat{y}_t based on the feature vector \mathbf{x}_t .

In contrast, a data stream may produce examples with a non-stationary distribution. The stream may initially consist of examples $e_i = (\mathbf{x}_i, y_i)$, similar to that of a static distribution. However, if the distribution should shift at some point in time then this may no longer hold true. According to Gama et al. [36], concept drift between time t_0 and t_1 can be defined as:

$$P_{t_0}(\mathbf{x}_i, y_i) \neq P_{t_1}(\mathbf{x}_i, y_i), \quad (1)$$

where p_t is the joint distribution at time t between the feature vector \mathbf{x}_i and the target class label y_i .

Kelly et al. [45] states that concept drift may occur in three distinct ways. Firstly, the class prior probabilities $p(y)$ may change over time. Secondly, the class distributions $p(\mathbf{x} | y)$ may change over time. And thirdly the class posterior distribution $p(y | \mathbf{x})$ may change. From the point of view of classification, only $p(\mathbf{x} | y)$ changes would affect the prediction and therefore require an algorithmic response.

The rate at which concept drift occurs can be categorised as one of three distinct forms; sudden drifts, gradual drifts and recurring drifts. Brzezinski and Stefanowski [15] concisely describes these three types of concept drift; sudden drifts occur when the source distribution is suddenly replaced by another distribution entirely, gradual drift occur at a much slower rate, and recurring drifts occur when older concepts reappear after some time period. Drifts can also be described as incremental where the drift consists of many intermediate changes, for example a network sensor deteriorates and becomes less accurate [36].

Similarly to categorising the rate of change, concept drift itself can be defined as either *real* or *virtual* [36, 71, 74]. Real concept drift refers to changes in $p(y | \mathbf{x})$, a change in the probability of a class label y given feature vector \mathbf{x} . This can result in classifier decision boundaries becoming affected. Alternatively, virtual drift refers to changes in $p(\mathbf{x})$ but not in $p(y | \mathbf{x})$. The result in this case is that the distribution has changed but the decision boundaries of the classifier are unaffected. Figure 1 illustrates the differences between real and virtual drift.

As is described by Zliobaite [74], it is not important if the drift is real or virtual since $p(y | \mathbf{x})$ is dependant on $p(\mathbf{x} | y)$. Explicit characterisation of various types of concept drift is effectively illustrated by Webb et al. [70]. This paper will not make any further differentiation between real or virtual drifts.

Where stream mining algorithms bring with them their own set of challenging algorithmic requirements, algorithms

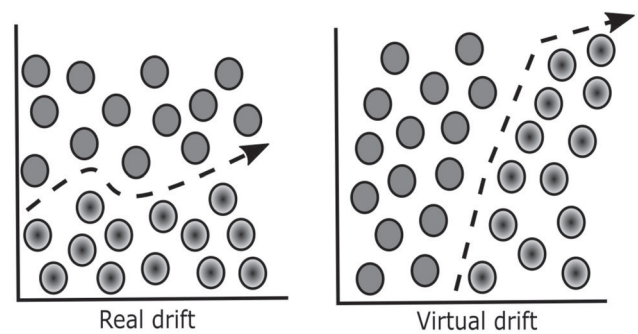


Fig. 1 Types of drifts. Classes represented circles

for concept drift detection must also meet certain demands. The following points are considered to be critical challenges that concept drift detection algorithms should overcome [35, 36]:

1. Detect as soon as possible the point at which the distribution has changed.
2. The crossover period during a shift in concept can produce noise. For example as distribution D_0 shifts to D_1 , examples produced by D_0 will act as noise for D_1 .
3. Algorithms should be computationally faster than the arrival time of examples from the stream. They should also be lightweight enough to not consume more than some fixed amount of memory for storage.

Any classifier operating with stream data must contain mechanisms to meet these requirements, otherwise their predictive performance will diminish over time. The predictive model will likely have to be capable of updating with new data as it arrives, or even replacing itself entirely.

4 Concept drift detection

This section critically reviews existing concept drift detection techniques by categorising drift detection methods as either statistical based, window based or ensemble based. Table 2 provides a full illustration of the categorisation and techniques covered by this paper.

4.1 Statistical methods

The Sequential Probability Ratio Test (SPRT) [66] is the backbone to a number of algorithms for concept drift detection. Given two distributions P_0 and P_1 for time period w , should the underlying distribution shift from P_0 to P_1 then the probability of observing elements from P_1 should be higher than

Table 2 Summary of concept drift detection techniques

Type	Algorithm	Acronym	Reference
Statistical based	Cumulative sum	CUSUM	Page [57]
	Page-Hinckley test	PH	Page [57]
	Drift detection method	DDM	Gama et al. [34]
	Early drift detection method	EDDM	Baena-Garcia et al. [3]
	Reactive drift detection method	RDDM	Barros et al. [4]
	Linear four rates	LFR	Wang and Abraham [67]
	Hierarchical linear four rates	HLFR	Yu and Abraham [73]
	Statistical test of equal proportions	STEPD	Nishida and Yamauchi [56]
	Fisher proportions drift detector	FPDD	de Lima Cabral and de Barros [22]
	Fisher-based statistical drift detector	FSDD	de Lima Cabral and de Barros [22]
	Fisher test drift detector	FTDD	de Lima Cabral and de Barros [22]
	McDiarmid drift detection methods	MDDMs	Pesaranghader et al. [58]
		MDDM-A	Pesaranghader et al. [58]
		MDDM-G	Pesaranghader et al. [58]
	MDDM-E	Pesaranghader et al. [58]	
Window-based	Concept-adapting very fast decision tree	CVFDT	Domingos and Hulten [27]
	Efficient concept-adapting very fast decision tree	E-CVFDT	Liu et al. [51]
	Adaptive sliding window	ADWIN	Bifet and Gavalda [8]
Block-based ensembles	Streaming ensemble algorithm	SEA	Street and Kim [65]
	Accuracy weighted ensemble	AWE	Wang et al. [68]
	Accuracy updated ensemble	AUE	Brzeziński and Stefanowski [14]
	Semi-supervised adaptive novel class detection and classification over data stream	SAND	Haque et al. [38]
	Efficient concept drift and concept evolution handling over stream data	ECHO	Haque et al. [39]
Incremental-based ensembles	Dynamic weighted majority	DWM	Kolter and Maloof [46]
	Learn++		Polikar et al. [59]
	Learn++.MT		Muhlbaier et al. [53]
	Learn++.NC		Muhlbaier et al. [55]
	Learn++.NSE		Muhlbaier and Polikar [54]
	Learn++.NIE		Ditzler and Polikar [24]
	Learn++.CDS		Ditzler and Polikar [25]

the probability of observing elements from P_0 . The statistical test is given as:

$$T_w^n = \log \frac{P(x_w \dots x_n | P_1)}{P(x_w \dots x_n | P_0)} = \sum_{i=w}^n \log \frac{P_1[x_i]}{P_0[x_i]} \quad (2)$$

$$= T_w^{n-1} + \log \frac{P_1[x_n]}{P_0[x_n]}$$

Introduced by Page [57], the Cumulative Sum (CUSUM) is a statistical technique based on the SPRT and is commonly adopted for concept drift detection. It receives as an input the residual of any filter, for example a Kalman filter, and outputs an alarm when the mean of the input data differs greatly from zero. Bifet [7] gives CUSUM as

$$g_0 = 0$$

$$g_t = \max(0, g_{t-1} + \epsilon_t - \nu) \quad (3)$$

if $g_t > h$ then alarm and $g_t = 0$,

where ϵ_t is the current observed value, ν is the allowed magnitude of change, t is the current time and h is a parameter defined threshold. This expressions functions for detecting changes that occur in a positive direction. If changes in a negative direction are required for detection then the $\min()$ function should be used in place of $\max()$.

CUSUM is memoryless in the sense that the probability of a drift being detected is not related to a drift having already been detected. It is also worth noting that the accuracy of CUSUM is dependant on the parameters ν and

h . Low values of v enable faster detection rates but at the cost of an increased rate in false positives.

The Page-Hinckley (PH) test, also proposed by Page [57], is a variant of the previously mentioned CUSUM test. The PH test can detect changes in the average behaviour of a process. The PH test for an increasing signal can be given as [7]:

$$\begin{aligned} g_0 &= 0 \\ g_t &= g_{t-1} + (\epsilon_t - v) \\ G_t &= \min(g_t, G_{t-1}) \\ \text{if } g_t - G_t &> h \text{ then alarm and } g_t = 0, \end{aligned} \quad (4)$$

where ϵ_t is the current observed value, v is the allowed magnitude of change, t is the current time and h is a parameter defined threshold. If the signal is decreasing then $G_t = \max(g_t, G_{t-1})$ and $G_t - g_t > h$ should be utilised as the stopping rule instead. Similarly to CUSUM, the PH test is memoryless but its accuracy is again parameter dependent on the values of v and h . Larger values of h will result in a lower false alarm rate, but some changes may also be missed altogether.

While the two algorithms are similar, they do offer solutions for different data streaming scenarios. Since CUSUM uses the residual from any predictor as an input is well suited for various applications of stream mining, and has been recently used for anomaly detection in video streams by Yang et al. [72]. PH is instead ideally suited for detecting abrupt changes in signal processing environments, but has also been adopted recently for the development of a rule learning algorithm for regression [29].

The performance of drift detection algorithms based on SPRT is often reliant on their false alarm and missed detection rates, as noted by Gama et al. [36]. However, during evaluative procedures such measures are usually overlooked as metrics to evaluate drift detector performance. This is explained fully in Sect. 5 of this research. The primary drawback and impact on performance is both algorithms' reliance on the parameters v and h . Both CUSUM and Page-Hinckley are considered state-of-the-art drift detectors in this category.

Another statistical method for detecting concept drift is based around monitoring the class distribution's constancy over time, as described by Brzezinski and Stefanowski [15]. This is undertaken by adopting various statistical techniques that produce "alarms" when the class distribution begins to change as time passes.

Drift Detection Method (DDM) [34] is a method that statistically compares two windows and controls the errors produced by a learning model during prediction. One window contains all of the data, and the second window consists of only data from the start of the stream to the point

at which the error rate of the prediction model increases. The windows are not kept in memory, only statistical information and recent errors are stored.

It is assumed that the error rate will decrease as the number of examples for observation increases, so long as the distribution is stationary. It is therefore suggested that a significant increase in the error rate of some learning model would indicate a change in the class distribution. DDM involves two principle variables in the form of p_t and s_t , where p is the probability of misclassification, s is the standard deviation and t is some timestamp. The standard deviation s_t is given as $s_t = \sqrt{p_t(1-p_t)/t}$. When $p_t + s_t$ reaches its minimum value the following conditions are checked:

- $p_t + s_t \geq p_{\min} + 2 \cdot s_{\min}$ as a warning level. Examples are stored in preparation of contextual drift.
- $p_t + s_t \geq p_{\min} + 3 \cdot s_{\min}$ as drift level. Concept drift is assumed to exist, the learning model is reset and a new model is trained using examples stored since the warning level was triggered. p_{\min} and s_{\min} are also reset.

DDM is almost memoryless, only statistics p_t and s_t are stored alongside the necessity of some available memory to store examples for retraining. A major flaw with DDM is that is only suitable for the detection of abrupt drifts. Gradual drifts can cause examples to be stored in memory for lengthy time periods which has the potential to cause catastrophic memory overflows.

A number of algorithms have built upon DDM and have aimed to improve its performance. The most famous of these is the Early Drift Detection Method (EDDM) [3]. EDDM uses the same approach and heuristics as DDM, however, rather than monitoring error rates, the distances between errors is measured. As predictions improve, the distance between two misclassification errors should increase. The window resizing follows the same procedure as DDM. The fundamental drawback to EDDM is that at least 30 errors are required for calculation which causes issues when applying this to imbalanced datasets. A more modern proposal to improve DDM is that of Barros et al. [4], who suggest the Reactive Drift Detection Method (RDDM) which discards older instances of particularly gradually occurring drifts in order to overcome potential memory overflows.

The DDM-OCI algorithm [69] was developed to solve the problem with using EDDM and imbalanced datasets. DDM-OCI makes the assumption that for imbalanced datasets, drifts only occur when there is a change in the minority class recall during classification. However it is entirely possible that a drift can occur without affecting the minority class recall, for example a drift from data that has an unbalanced class distribution to one which is balanced.

DDM-OCI also suffers from an issue where a number of false positives can be triggered due to a weakness in its test statistic \hat{R}_{TPR}^t .

With DDM-OCI falling short of overcoming the class imbalance problems present in EDDM, the Linear Four Rates framework (LFR), was proposed by Wang and Abraham [67]. It is designed as a direct improvement over the DDM-OCI algorithm. LFR monitors the four values, or rates, given by a typical confusion matrix; precision and recall for both minority and majority classes. Statistical bounds are set as thresholds and should any of the four rates exceed the threshold then a drift is assumed to have occurred.

The current, most recent advancement in these algorithms is the Hierarchical Linear Four Rates method (HLFR) proposed by Yu and Abraham [73], given in Algorithm 1. HLFR operates using a two layer, hierarchical structure wherein the first layer is responsible for detecting potential drifts, and the second layer validates said drift and communicates this information back to the first layer. Layer one monitors the same four rates of the confusion matrix as was introduced by LFR. When a drift is detected, layer two applies a permutation test to confirm if the detected drift is true or a false positive. In the occurrence of a false positive the testing process restarts.

Algorithm 1 HLFR Algorithm

Require: Data $\{X_t, y_t\}_{t=0}^{\infty}$ where $X_t \in R^d, y_t \in \{0,1\}$
 Ensure: Concept drift time points $\{T_{cd}\}$

- 1: **for** each $t = 1$ to ∞ **do**
- 2: Perform Layer-I hypothesis testing
- 3: **if** Layer-I detects potential drift point T_{pot} **then**
- 4: Perform Layer-II hypothesis testing on T_{pot}
- 5: **if** Layer-II confirms the potentiality of T_{pot} **then**
- 6: $\{T_{cd}\} \leftarrow T_{pot}$
- 7: **else**
- 8: Discard T_{pot} ; Reconfigure and restart Layer-I
- 9: **end if**
- 10: **end if**
- 11: **end for**

Experimental results using real datasets indicate that HLFR outperforms DDM, EDDM and LFR in terms of not only accuracy but also in terms of its time to detection (detection delay). The framework presented with HLFR symbolises a move away from the traditional “concept drift detector plus classifier” approach. This current method suffers from various evaluation problems, discussed in depth in Sect. 5. The proposed framework of HLFR is a concrete starting point for future research that aims to address such issues.

DDM and EDDM are considered state-of-the-art statistical based detectors [7], even though they have drawbacks in relation to imbalanced data. Algorithms that have aimed to address this, such as DDM-OCI and LFR have fallen short. The result of which is that two, aged drift detectors that are sub-par in terms of performance are still considered

state-of-the-art. A second reason is due to the datasets used in concept drift experimentation. This is explained fully in Sect. 5, but a lack of benchmark datasets means that many experiments used simulated data in which parameters can be defined to avoid class imbalance, thus artificially avoiding the drawbacks of DDM and EDDM.

STEPD [56], or Statistical Test of Equal Proportions, monitors the predictions of a base classifier for drift detection in a similar manner to that of DDM and EDDM. However, STEPD also uses two parameters as significance levels to distinguish between detected drifts and warnings. These are $\alpha_d = 0.003$ and $\alpha_w = 0.05$ respectively. STEPD uses two windows to compare the result of the classifier. The first window is a “recent” window, of which its size is defined by a parameter with a default value of 30 instances. The second window is the “older” window which contains all instances observed since the last detected drift. STEPD compares the accuracies of these windows through a hypothesis test of equal proportions, given as

$$T(r_o, r_r, n_o, n_r) = \frac{|r_o/n_o - r_r/n_r| - 0.5 \times (1/n_o + 1/n_r)}{\sqrt{\hat{p} \times (1 - \hat{p}) \times (1/n_o + 1/n_r)}} \quad (5)$$

where r_o is the number of correct predictions from examples within the n_o “older window”, r_r is the correct predictions from examples within the n_r “recent” window and $\hat{p} = (r_o + r_r)/(n_o + n_r)$. The result is used to calculate the p -value from the standard normal distribution table and is then compared against to α_d and α_w to determine if a drift or warning alarm must be issued. If p -value $< \alpha_d$ then a drift is detected, similarly if p -value $< \alpha_w$ then a warning is signalled.

The authors recognised that for small sample sizes the statistical test of equal proportions was ineffective, admitting that Fisher’s Exact test [32] should have been used but was ignored due to its high computational cost. Recent work by de Lima Cabral and de Barros [22] proposes three new methods using the Fisher’s Exact test, Fisher Proportions Drift Detector (FPDD), Fisher-based Statistical Drift Detectors (FSDD) and Fisher Test Drift Detector (FTDD). All three methods use the same approach as STEPD in regards to the two windows and the significance thresholds. However, these approaches use different statistical tests and measure the difference in errors rather than correct predictions.

FPDD uses the Fisher’s Exact test when the number of errors or correct predictions in either window is smaller than five, otherwise it operates exactly like STEPD. FSDD extends FPDD such that instead of using the test of equal proportions in situations where the number of errors or correct predictions in either window is smaller than five, the chi-square test for homogeneity of proportions is

applied [17] FTDD explicitly uses the Fisher's Exact test for drift detection. Experimental results showed that all three proposed methods outperformed STEP, with very little difference between themselves.

The McDiarmid Drift Detection Methods (MDDMs) proposed by Pesaranghader et al. [58] uses a weighting scheme to give substance to elements of a sliding window to enable faster concept drift detection. MDDM applied McDiarmid's inequality [52] to detect drifts in evolving streams. The algorithm uses a sliding window of size n which stores prediction results. If the prediction is correct, a 1 is inserted into the window, otherwise a 0 is inserted. Each element in the window is weighted such that $w_i < w_{i+1}$. This means that elements at the head of the window have larger weights than that of the tail. Three methods based on different weighting schemes are presented. MDDM-A which uses the arithmetic weighting scheme, MDDM-G which uses the geometric weighting scheme and MDDM-E which uses the Euler weighting scheme. The arithmetic weighting scheme for MDDM-A is given as

$$w_i = 1 + (i - 1) \times d \quad (6)$$

where $d \geq 0$ is the difference between two consecutive weights. The geometric scheme used by MDDM-G is given as

$$w_i = r^{(i-1)} \quad (7)$$

where $r \geq 1$ is the ratio of two consecutive weights. Finally, the Euler scheme adopted for MDDM-E is given by the authors as

$$r = e^\lambda \quad (8)$$

where $\lambda \geq 0$. Prediction results are processed sequentially and the weighted average of all elements within the window is calculated alongside each arriving prediction result. This is used to update two variables, μ_w^t the current weighted average and μ_w^m the maximum weighted average observed thus far. A drift is detected if there is a significant difference between μ_w^m and μ_w^t . The significance is determined by McDiarmid's inequality. The authors selected popular concept drift datasets for experimental testing, including Elec2, Forest Covertype and Poker Hand. These

are explained in Sect. 5 of this paper. Their results found that MDDMs outperformed existing methods including EDDM, CUSUM and Page-Hinckley in terms of drift detection delay and classification accuracy.

4.2 Window-based detectors

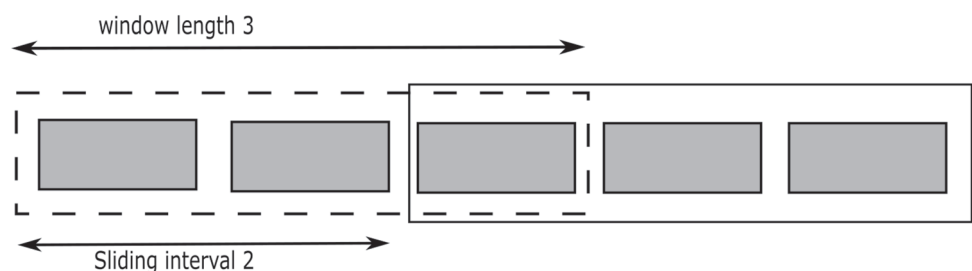
Concept drift detection algorithms that are window-based operate in a unique way. Rather than monitoring arriving instances from a stream individually, sliding windows of varying sizes, widths, are instead statistically monitored. Larger windows correlate with higher performance accuracy, however they may also contain concept drift within themselves that could escape unnoticed. Smaller windows tend to facilitate better concept drift detection. The sizing of a window is a fundamental problem when designing any stream mining algorithm that utilises sliding windows. Figure 2 provides an illustration of how a sliding window operates.

The Hoeffding Tree is a mathematically justified algorithm used to construct decision trees. The Very Fast Decision Tree (VFDT) is a heuristic algorithm proposed by Domingos and Hulten [27] that is based on the Hoeffding Tree. The VFDT is an algorithm which incrementally constructs a decision tree of incoming examples from a data stream, without the need to store examples in memory. A critical difference between the Hoeffding tree and traditional classification trees lies in the selection of which node is used for splitting. Where traditional trees adopt techniques such as Information Gain [60] to determine the best node for splitting, VFDTs use the Hoeffding bound to determine how many examples are necessary to identify the best splitting node based on a user defined confidence threshold. The Hoeffding bound states that, for a random variable r in range R and where \bar{r} is the mean of n observations, with probability $1 - \delta$ the true mean is at least $\bar{r} - \epsilon$ where ϵ is given as

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \quad (9)$$

It is also worth noting that the Hoeffding bound has been suggested to be statistically inappropriate for constructing

Fig. 2 Sliding window. Borders identify two different windows



decision trees for data stream mining [21, 44, 63]. VFDTs are, however, only suitable for static streams and include no method for forgetting or restarting learning in the presence of concept drift. In order to enable VFDTs to account for concept drift, Hulten et al. [41] introduced Concept-adapting Very Fast Decision Tree (CVFDT). CVFDT monitors a sliding window of examples. As new examples arrive, CVFDT updates its node statistics by incrementing counts of the new, arriving, examples. Counts that relate to the oldest example in the window, which must be forgotten, are decremented. If necessary, older examples are removed from the window. Hulten et al. [41] states that if the concept is changing then nodes that previously passed the Hoeffding test may no longer pass. In this case, CVFDT grows a second sub-tree with the new best attribute, according to the Hoeffding bound, at its root. If the new sub-tree's accuracy outperforms that of the older tree then it replaces it completely.

Criticising CVFDT for not offering mechanisms which handle specific types of drift, such as gradual or abrupt, Liu et al. [51] proposed the E-CVFDT algorithm which utilises a caching mechanism. Their results show that E-CVFDT yields a higher classification accuracy for gradual concept drifts, but does not make any improvements during sudden drifts.

The Adaptive Sliding Window (ADWIN) algorithm [8] is another popular, window-based detector for coping with concept drift. Assuming a stream of examples x_1, x_2, \dots, x_n produced by some distribution at time t , these serve as inputs to ADWIN to produce sliding window W . Let $\hat{\mu}_W$ denote the average of examples contained within W and μ_W represent the unknown average of μ_t such that $t \in W$. Let n be the length of W and n_0 and n_1 be the lengths of W_0 and W_1 respectively, such that $n = n_0 + n_1$. Algorithm 2 provides the algorithm for ADWIN.

Algorithm 2 ADWIN Algorithm

```

1: Initialise window  $W$ 
2: for each  $t > 0$  do
3:    $W \leftarrow \cup W\{x_t\}$  (add  $x_t$  to head of  $W$ )
4:   repeat
5:     Drop elements from the tail of  $W$ 
6:   until  $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| > \epsilon_{cut}$  holds for every split of  $W$  into  $W = W_0 \cdot W_1$ 
7:   output  $\mu_W$ 
8: end for

```

Whenever two “large enough” subwindows of W display “distinct enough” averages, a drift is assumed to have occurred, and the older of the two subwindows is dropped. The terms “large enough” and “distinct enough” are defined by the Hoeffding bound statistic. The average

of the two subwindows are tested to determine if they are larger than ϵ_{cut} , given as $|\mu_{W_0} - \mu_{W_1}| > 2\epsilon_{cut}$ where

$$\begin{aligned}
 m &= \frac{1}{1/n_0 + 1/n_1} \\
 \delta' &= \frac{\delta}{n} \\
 \epsilon_{cut} &= \sqrt{\frac{1}{2m} \cdot \ln \frac{4}{\delta'}}
 \end{aligned}
 \tag{10}$$

This was considered computationally expensive since all “large enough” subwindows are checked for potential cuts. The window contents are also explicitly stored and memory requirements scale linearly with the window size. This has the obvious drawback of potentially large memory and processing requirements.

A solution to the resource demands of ADWIN is proposed in the form of ADWIN2, provided in Algorithm 3.

Algorithm 3 ADWIN2 Algorithm

```

1: for each time  $t$  do
2:   for all partitions of  $W$  into  $W_1 + W_2$  do
3:     if  $W_2$  comprises exactly a number of buckets then
4:       if test  $(W_1, W_2, \delta) \geq \mathcal{E}(|W_1|, |W_2|, \delta)$  then
5:         declare change and drop the older bucket
6:       end if
7:     end if
8:   end for
9: end for

```

In order to reduce the computational time when determining the best cutting point in the window, buckets are adopted as a means for grouping data within the window. Such buckets have two core elements; capacity and content. Each time a new example arrives from the stream, if the element is “1” then a new bucket is created of content 1 and capacity equal to the number of elements arrived since the last observed “1”. The remaining buckets are then compressed.

In order to eliminate the problems of high memory demands caused by the explicit storing of all window contents, a variation of the exponential histogram [20] is used. The performance of ADWIN2 is given by the authors in Big O notation as $O(\log W)$ memory and cutpoints, with the worst case per example equating to $O(\log^2 W)$.

ADWIN2 is usually referred to directly as ADWIN in published work and is considered one of the state-of-the-art concept drift detectors. A common problem with sliding windows is that the width usually needs to be predefined. Varying the width of the window has an impact on performance, thus applying the correct width value is important. Typically this is done by means of some user-defined

parameter. ADWIN, however, is parameterless and the sliding window is sized dynamically by the algorithm itself. It also provides excellent performance due to the use of buckets and the adaptation of the exponential histogram to for compression.

4.3 Block-based ensemble detectors

In machine learning, an ensemble refers to a group or collection of classifiers that work together to achieve greater predictive performance. Block-based ensembles process data in blocks, or chunks, of some specified size. The performance of block-based ensemble methods is based heavily on the chunk size. Similarly to that of sliding windows, larger chunks tend to produce more accurate classifiers but may contain concept drift within themselves. Alternatively, smaller based chunk sizes are typically more effective at drift detection but produce inferior performing classifiers.

The Streaming Ensemble Algorithm (SEA) was first proposed by Street and Kim [65] and is a block-based ensemble learning algorithm. Individual classifiers are constructed from examples read in sequential blocks (chunks), which are then added to a fixed size ensemble. If the ensemble is full, then the worst performing classifier is removed from the ensemble entirely.

In their experiments, C4.5 [61] classifiers are used for building the ensemble. The output prediction is given as the simple majority voting of the entire ensemble. Results for testing with concept drift showed that the algorithm was capable of recovering quickly by discarding classifiers trained on the outdated data.

A notable drawback to SEA is the way in which classifiers are replaced. Merely replacing the worst performing classifier with the most recently trained has the potential to still leave several, outdated and poorly performing classifiers in the ensemble, depending on the predetermined ensemble size.

This is improved upon by Wang et al. [68]'s Accuracy Weighted Ensemble (AWE), a block-based algorithm which trains a new learning model with each new chunk of data in a similar fashion to that of SEA. Where AWE improves upon SEA is in the model replacement. AWE uses a version of the mean square error to select n best classifiers to construct an entirely new ensemble, thus removing all other outdated and poorly performing classifiers. The algorithm for AWE is given in Algorithm 4.

Algorithm 4 AWE Algorithm

Input: S a dataset of $ChunkSize$ from the stream
 K the total number of classifiers
 C a set of previously trained classifiers
Output: C a set of K classifiers with updated weights

- 1: Train classifier C' from S
- 2: Computer error rate of C' via cross validation on S
- 3: Derive weight w' for C'
- 4: **for** each classifier $C_i \in C$ **do**
- 5: apply C_i on S to derive MSE_i
- 6: Compute w_i
- 7: **end for**
- 8: $C \leftarrow K$ of the top weighted classifiers in $C \cup \{C'\}$
- 9: **return** C

Let S denote a data stream in chunks S_1, S_2, \dots, S_n where each chunk is of equal size and C_i represent some classifier for S_i . The weight of classifier C_i is the estimated prediction error using the most recent data S_n . Since S_n is a data stream and will produce examples in the form (\mathbf{x}, c) where \mathbf{x} is the feature vector and c is the class label, the classification error of C_i is $1 - f_c^i(x)$ where $f_c^i(x)$ is the probability that x is an example of class c . As such, the mean square error of C_i is given by

$$MSE_i = \frac{1}{S_n} \sum_{(x,c) \in S_n} (1 - f_c^i(x))^2 \quad (11)$$

Should a classifier predict randomly then the mean square error can be given as:

$$MSE_r = \sum_c p(c)(1 - p(c))^2 \quad (12)$$

A random classifier contains no meaningful knowledge of the data, it makes predictions simply at random. Therefore MSE_r is used as a threshold for weighting and classifiers whose error rate is at least equal to MSE_r are discarded. The weight of a classifier C_i is given as:

$$w_i = MSE_r - MSE_i \quad (13)$$

One drawback to AWE is the issue of chunk-size optimisation, however it should be noted that this is common in all block-based ensemble methods. A second drawback is the weighting function of AWE, in particular the MSE_r threshold. In environments with sudden concept drift it can have a silencing effect on the entire ensemble resulting in no class prediction [14].

Brzeziński and Stefanowski [14] proposed the Accuracy Updated Ensemble (AUE) algorithm as an improvement to that of AWE. The algorithm for AUE is given in Algorithm 5.

Algorithm 5 AUE Algorithm

Input: S a data stream, k the number of ensemble classifiers
Output: \mathcal{E} an ensemble of k online classifiers with updated weights

- 1: $C \leftarrow 0$
- 2: **for** (all data chunks $x_i \in S$) **do**
- 3: train classifier C' on x_i
- 4: compute error rate MSE of C' on x_i
- 5: derive weight w' for C' using (3)
- 6: **for** (all classifiers $C_i \in C$) **do**
- 7: apply C_i on x_i to derive MSE_i
- 8: compute weight w_i based on (3)
- 9: **end for**
- 10: $\mathcal{E} \leftarrow k$ of the top weighted classifiers in $C \cup \{C'\}$
- 11: $C \leftarrow C \cup \{C'\}$
- 12: **for** (all classifiers $C_e \in \mathcal{E}$) **do**
- 13: **if** $w_e > \frac{1}{MSE_e}$ and $C_e \neq C'$ **then**
- 14: update C_e with x_i
- 15: **end if**
- 16: **end for**
- 17: **end for**

AUE implements online classifiers enabling the individual learning models to be updated directly rather than only adjusting weights as per AWE. If no concept drift were to occur between a series of chunks then the classifiers would improve as if they were trained on one large chunk. This means the block size can be reduced without risking the performance accuracy of the ensemble classifiers. The weighting function in AUE is a simplified version of that used in AWE, and is given as

$$w_i = \frac{1}{MSE_i + \epsilon}, \quad (14)$$

where MSE_i is calculated identically as it is in AWE and ϵ is a small constant value to allow weighting calculations when MSE_i is equal to 0.

Experimental results showed that AUE performed more accurately than AWE on all similar datasets apart from one where the performance accuracy was equal. A second implementation of this algorithm, AUE2, was suggested by Brzezinski and Stefanowski [15] which improves on the memory usage and accuracy of AUE by implementing a new weighting function and pruning base learners.

AUE overcomes the problems present in AWE. The weighting function is redesign to cope with sudden concept drifts, and the use of online classifiers allows smaller chunk sizes to be used without a severe reduction in classifier accuracy. However when no concept drift is occurring, all classifiers are updated with arriving chunks. Should this continue over multiple chunks then the outcome is that ensemble classifiers lose their uniqueness.

Recent advances in concept drift detection using block-based ensembles have introduced new algorithms entirely. The Semi-supervised Adaptive Novel Class Detection and Classification over Data Stream (SAND) framework is proposed by Haque et al. [38]. SAND consists of four

independent modules; Classification, Novel Class Detection, Change Detection and Update.

The framework maintains an ensemble of classifiers based on k-nearest neighbour, using algorithms such as k-means. The ensemble is initially trained on some training data. When an instance arrives from some stream it is classified using majority voting. It also produces a confidence value which indicates the ensemble's confidence in the prediction. These confidence values are stored in a sliding window.

The Change Detection module monitors the distribution of confidence values within the sliding window. Any significant change in the distribution is assumed to be caused by the existence of concept drift. Once change has been detected, a new chunk of data is used to update the ensemble and the chunk boundaries are determined dynamically. Updating is undertaken by requesting only class labels for instances in the current chunk where the confidence values were weak. The ensemble is then updated with the new model and the sliding window is reset.

Experimental results showed that SAND was capable of achieving good prediction accuracy using limited labelled data, however its execution time was inefficient due to the high resource cost of the Change Detection module being executed after the calculation of individual confidence scores.

In order to attempt to remedy the poor execution time of SAND, the Efficient Concept Drift and Concept Evolution Handling over Stream Data (ECHO) was proposed by Haque et al. [39]. ECHO operates in the same manner as SAND, however, the execution of the Change Detection module is selective rather than at each calculation of the confidence threshold. Two methods of selectively invoking the Change Detection module. The first is to use a fixed

threshold γ , such as the classifier confidence threshold. If the confidence of a test instance C^x is less than γ , the Change Detection module is invoked. The second proposed approach is to calculate the probability of invocation based on the C^x . A high confidence value would result in a low probability of invocation.

ECHO performs competitively and is suitable for use in environments which generate a low level of labelled data. However, in stream mining there is an innate assumption that class labels are always available with arriving instances. Whilst this may not be the case in real world scenarios, since concept drift detection is still in its infancy it follows that this assumption can continue to be made. Classifiers are only updated by labels for which there was a low confidence value. While this does aid in situations where labels are missing by lowering the demand for labels, there is no existing mechanism for actually determining if a label is available or not. The result is that classifiers may not be updated with information when there is an available class label, which will negatively impact the potential performance of the ensemble.

4.4 Incremental ensemble detectors

Incremental, or online, ensembles are another method of ensemble based learning. In contrast to block-based ensembles, incremental ensembles process elements sequentially rather than in chunks.

Dynamic Weighted Majority (DWM) was first proposed by Kolter and Maloof [46]. DWM is an ensemble of classifiers referred to as 'experts', where each is given an associated weight. For some test example, the experts each provide a prediction. This is then used in combination with their weights to output the overall prediction in the form of the class which has the largest accumulated weight total.

Should an individual expert provide an incorrect prediction, then its corresponding weight is reduced. If the output prediction of DWM is incorrect then a new expert is created and is assigned a weighting of 1. Experts are normalised by uniform scaling such that the highest weighting possible is 1. Experts with a weight lower than a user-defined threshold value are removed. Through the use of uniform weights and incremental learning, the authors state that the DWM algorithm is capable of handling concept drift.

DWM is considered one of the state-of-the-art concept drift detection methods, and has been as a benchmark algorithm in recent studies reviewing concept drift [7, 36, 76]. However, one particular problem with DWM is the way in which experts are added. Rather than adding

a new expert when the ensemble prediction is incorrect, the age of experts and historical prediction accuracy could be taken into consideration. The base learner also explicitly maintains examples in memory which has the potential to consume large amounts of resources, depending on the stream size. Other algorithms such as CVFDT and ADWIN have already solved this issue so it follows that similar implementations could be made to DWM.

4.4.1 Learn++ algorithms

The Learn++ algorithm family is a set of algorithms consisting of an ensemble of incrementally trained classifiers using batches of data and weighted majority voting. According to Liao et al. [50] existing algorithms in the Learn++ family include Learn++, Learn++.NC, Learn++.MT, Learn++.NSE, Learn++.NIE and Learn++.CDS. Elwell and Polikar [31] also mention the Learn++.MF algorithm as part of the family.

The original Learn++ algorithm, suggested by Polikar et al. [59], constructs k classifiers for a single batch of incoming data. Examples from this batch are used to train a single, first classifier. Prediction errors are used to produce a weighted distribution of all examples, with misclassified examples possessing a higher probability of being sampled. Training the second classifier through to the k classifier for the ensemble, training examples are selected based on the weighted distribution of all examples. Classification errors are then used to update the weighted distribution.

One problem with Learn++ is that all base classifiers are persisted over time T , resulting in old data never being forgotten by the ensemble. This can cause a problem known as 'outvoting'. Older classifiers in the ensemble may produce incorrect predictions due to the aged examples they are trained on. If the outdated classifiers make up the majority of the ensemble, even if their weights are small, they can 'outvote' the classifiers trained on newer data. Thus through the majority weighted voting procedure, produce incorrect predictions. Another problem is that without forgetting old information, concept drift cannot be accounted for.

Older algorithms in the Learn++ family sought to provide solutions to the concept drift problem. Learn++.MT [53] solves the outvoting problem by using a dynamic weighted voting technique. Learn++.NC (New Class), proposed by Muhlbaier et al. [55], furthers this concept and introduces a Dynamically Weighted Consult and Vote (DW-CAV) mechanism which enables incremental learning of new classes. Learn++.NC enabled base classifiers within the ensemble to consult among themselves

when classifying a given example, and weights the decision of each base classifier. Classifiers check their predictions against classes which they are trained and, based on the decision of other classifiers, check if their prediction is in-line with others. If a classifier's decision is an outlier compared to the majority, the classifier may either reduce its voting weighting or withdraw from predicting altogether. These algorithms, however, are only suitable for static, stationary, environments where the distribution does not change.

Learn++.NSE, proposed by Muhlbaier and Polikar [54], aims to account for various forms of concept drift. An ensemble of classifiers is trained on the current data distribution D at time t . Change is monitored by examining the performance of the ensemble over time. Learn++.NSE will generate a new classifier and combine it with the ensemble when the prediction error of the current ensemble falls below some threshold. Classifiers are weighted according to the time t they were instantiated, such that newer classifiers have a larger weighting than older classifiers during prediction.

Whilst Learn++.NSE aimed to address the issue of evolving data, none of the existing Learn++ algorithms accounted for class imbalance at this stage. Learn++.NIE [24] extends Learn++.NSE, but incorporates evaluation measures for data class imbalance, such as f-measure. Learn++.NIE also implements sub-ensembles in place of individual classifiers in order to reduce stochastic errors. An alternative approach was proposed by Ditzler and Polikar [25], whose Learn++.CDS algorithm instead uses preprocessing with SMOTE, an oversampling method, rather than changing the evaluation metric to account for class imbalance. SMOTE adds instances to the minority class in order to create a more balanced dataset.

The comparative performance of the Learn++ algorithms was reviewed by Liao et al. [50]. The performance of each algorithm is dependent heavily on the base classifier used. This was especially apparent in environments with imbalanced data and incremental learning. The current state of the Learn++ algorithm family requires considerable work to produce solutions that can cope with concept drift and imbalanced data, although discussion of the latter is out of scope for this research. Learn++.NSE provides a starting point for using the Learn++ family for concept drift detection. However it is outdated and its approach weighting favours newly created classifiers during prediction which is a flawed approach; it is entirely feasible that older classifiers in an ensemble may be better equipped to make predictions than newer classifiers.

5 Evaluation

In traditional machine learning scenarios, the typical evaluation procedure is to train a model, cross-validate and then test using metrics such as prediction accuracy or f-score. For stream mining this approach is ineffective. Since stream data arrives online in a continuous and sequential fashion, it is not possible to first train the model and then test. Instead one of two methods can be used; prequential evaluation, sometimes referred to as interleaved-test-then-train, or holdout evaluation.

Prequential evaluation is implemented using the following procedure. For each arriving element from a stream the model is first tested by predicting the class label, after which the same element is used to train the model. Prequential evaluation can be used in conjunction with sliding windows and decaying factors to improve classification results in evolving data streams. A full comparative assessment is given by Gama et al. [35].

The holdout evaluation procedure offers an alternative approach. This involves with-holding a subset of data examples from the classifier to be used as a training set at specific time intervals, for example every one hundred thousand instances. Algorithm 6, adapted from Bifet and Kirkby [9], shows an example algorithm for holdout evaluation. The most obvious problem for holdout is the acquisition of examples for use as training data. A solution to this is to systematically store arriving samples from the stream at varying intervals. Similarly, ascertaining the adequate number of examples to provide accurate evaluation measurements also poses a challenge. It is suggested by Bifet and Kirkby [9] that a test set in the region of tens of thousands of examples is sufficient, however, this is an enormous potential range and doesn't provide a concise estimate.

Algorithm 6 Holdout Evaluation Framework

```

 $m_{bound}$ : the maximum memory allocation for the model
 $n_{test}$ : Holdout examples as a test set
 $n_{train}$ : Training examples arriving from the stream
while Evaluation is required do
  for  $i = 1$  to  $n_{train}$  do
    Get next example  $e_{train}$  from the stream
    Train and update the model, ensuring  $m_{bound}$  is valid
  end for
  for  $i = 1$  to  $n_{test}$  do
    Get next example  $e_{test}$  from the test set
    Test the model using  $e_{test}$  and update model accuracy
  end for
end while

```

Krempel et al. [49] states that a problem with evaluating stream mining classifiers in general is a lack of benchmark datasets for cross comparison. Instead, datasets are often synthesised used tools such as MOA [10]. This is also a problem for evaluating concept drift detection algorithms.

Table 3 Drift detector evaluation metrics

Metric	Explanation	Formula
MTFA	Mean time to false alarms. Frequency of false alarm triggers	$E_{\theta_0}(t_a)$
FAR	False alarm rate	1/MTFA
MTD	Mean time to detection. How quickly occurring drift is identified	$E(t_a - t_0 + 1 t_a \geq t_0)$
ARL	Average run length. Time to alarm after change of size θ	$E(t_a - k \text{change of size } \theta \text{ at time } k)$

Few benchmark datasets exist for testing concept drift detectors. The most popular dataset used for evaluating concept drift detection algorithms is the Electricity Dataset [40]. This dataset is used in various concept drift related research publications [7, 34, 46, 75]. This dataset was taken from the Australian New South Wales electricity market, in which electricity prices are not statically set. Instead, the price fluctuates according to demand. The dataset is constructed of 45,312 electricity prices which were taken at 30 min intervals. Examples are labelled as either "UP" or "DOWN" which reflect their current price in comparison to the last 24 h. Another popular dataset is the Forest Covertype [13] dataset. This consists of 581,012 instances of 54 attributes that describe various types of forest cover of the Roosevelt National Forest in northern Colorado. The KDD'99 dataset is well known and subject to somewhat extreme temporal dependence. This dataset contains information pertaining to simulated intrusions in a military network environment. It contains 23 class labels representing either normal traffic or some form of intrusion. The dataset contains over 494,000 records of 41 features each. This dataset is of considerable age, but due to the high level of temporal dependence has been used in recent studies such as Zliobaite et al. [76]. The Poker Hand dataset from UCI Machine Learning Repository [28] contains one million records of 11 attributes that represent a poker hand of five cards from a standard 52 card deck. Each card has two corresponding attributes, the suit and the rank, and there is one additional attribute that describes the hand, e.g. royal flush or full house. A dataset similar to Poker Hand that contained more class labels was used in the work of Cattral et al. [16]. The Airlines dataset from Data Expo 2009 [42] contains 120 million records of 13 attributes relating to flight departure and arrival information from internal commercial flights in the USA between October 1987 and April 2008. The target class label is the arrival delay given in seconds, and the classification goal is to determine the flight delay time given the arrival and departure information. This dataset has been used in the work of Ikonomovska et al. [43].

Another fundamental problem in evaluating drift detectors stems from the use of classifier accuracy as an

evaluation metric. This has been criticised in recent literature [7, 12, 76] where it has been suggested that classifier accuracy doesn't reflect the performance of the concept drift detector. Bifet [7] explains that drift detectors should be evaluated in terms of their ability to handle false alarms, their true detection rate and the time taken to correctly identify an occurring drift. This is reflected in evaluation criteria proposed by Basseville et al. [5] and Gustafsson and Gustafsson [37], which are given in Table 3. These are existing, historic metrics which capture properties of concept drift detectors, however, at the time of writing there appears only the work of Bifet et al. [11] utilises these metrics for evaluation.

These metrics have existed for over a decade, yet are not used in published work. One possible reason for this is a lack of frameworks which support these metrics. Each must be independently calculated when implementing models, which is time consuming and can increase complexity. Since prediction accuracy is readily available in virtually all machine learning frameworks, it's of no surprise that this metric is used to evaluate the impact of change detectors. This is only bolstered by the idea that in most instances the performance of the change detector itself may be viewed as unimportant; only the performance of the classifier truly matters.

The lack of an existing evaluation framework was an issue that was addressed by Bifet et al. [11] where the authors propose CD-MOA, a GUI extension to MOA (Massive Online Analysis). CD-MOA offers an interface for evaluating change detectors. However, the evaluation measures provided are again different. CD-MOA provides information on time and memory resources, as well as a metric called RAM-Hours which merges both time and memory together. The most recent version of CD-MOA also provides a measure based on Cohen's Kappa statistic [19], which compares observed accuracy with an expected accuracy.

A further issue with the metrics of Basseville et al. [5] and Gustafsson and Gustafsson [37], as given in Table 3, is their numerousness. Having five independent statistical measures to evaluate a drift detector obfuscates a true representation of performance. In order to tackle this,

Bifet [7] proposed a new single metric, the Mean Time Ratio (MTR), which encapsulates the ratio between MTFA and MTD metrics. The motivation of this was that the MTFA and MTD metrics are arguably the two most important, thus providing a single metric representing these eliminates the confusion of having five metrics. The MTR metric is given as follows.

$$MTR(\theta) = \frac{MTFA}{MTD} \times (1 - MDR) = \frac{ARL(0)}{ARL()} \times (1 - MDR) \quad (15)$$

Ultimately, the problem with current evaluation metrics is a lack of agreement and absence of gold-standard techniques. The metrics presented in Table 3 represent performance characteristics of a change detector, but numerosness and a lack of frameworks has left them virtually unused. The MTR metric aims to eliminate the problem of numerosness by providing a single metric which combines both the mean time to false alarms and the mean time to detection. CD-MOA provides a framework for evaluation, but provides a different set of metrics within its GUI for evaluating drift detectors. The result is that drift detectors are evaluated based on their impact to classifier accuracy, however this doesn't truly evaluate the concept drift detector itself. Further research should aim to establish a standardised set of statistical evaluation metrics that encapsulate the various performance characteristics of a drift detector, as well as its impact on classifier accuracy.

Existing statistical measures focus on the evaluation of baseline classifiers. The Kappa statistic proposed by Cohen [19] is a statistical measure for evaluating classification results when using imbalanced data, both in the context of streaming data and in traditional batch learning. The Kappa statistic is defined as

$$k = \frac{P - P_{ran}}{1 - P_{ran}}, \quad (16)$$

where P is the accuracy of some base classifier and P_{ran} is the accuracy of a classifier that predicts labels at random.

The Kappa statistic, however, is not a suitable metric when temporal dependence exists within the data, as shown by Zliobaite et al. [76]. Instead, the authors propose a new statistic called Kappa Temporal, defined as

$$k_{per} = \frac{P - P_{per}}{1 - P_{per}}, \quad (17)$$

where P_{per} is the probability of a Persistent classifier, a classifier that simply predicts that the next class label will be the same as the immediately previously known class label. Using this measure, trained classifiers performing correctly will achieve a k_{per} score of 1, or if performing

worse than the Persistent classifier P_{per} , a score of 0. The substantial drawback to the Kappa Temporal measure is the direct inverse to the Cohen's Kappa statistic described above. Kappa Temporal is ineffective for imbalanced datasets since a Majority class classifier, a classifier that simply predicts the class with the largest prior probabilities, will outperform that of a Persistent classifier.

Zliobaite et al. [76] offer a solution to this problem by combining both Cohen's Kappa statistic and the Kappa Temporal statistic together, forming the Combined Measure [76]. This is given as

$$K^+ = \sqrt{\max(0, k) \max(0, k_{per})} \quad (18)$$

This Combined Measure will provide a statistical evaluation score of 0 if either the Kappa or the Kappa Temporal metrics fail. This provides a single evaluation metric that encapsulates a classifier's ability to cope with both temporal dependence and imbalanced data. However, it is only a statistical metric and does not offer any mechanism for base classifiers to handle temporal dependence during the classification process.

Further evidence of the need for additional evaluation criteria outwith of classifier performance is shown by Bifet [7]. A "No Change" detector is compared to state-of-the-art drift detectors using a Naive Bayes classifier with both the Electricity and Forest Covertype datasets. The No Change detector is a drift detector that performs no statistical monitoring of the stream data but instead outputs a false positive change every 60 instances. The results of this show that the No Change detector outperforms state-of-the-art detectors in terms of accuracy on both datasets. This reinforces the concept that the use of accuracy as a metric for the performance of concept drift detectors is insufficient.

A possible direction for future research is to design and produce new statistical evaluation criteria. Krawczyk et al. [48] suggests that metrics such as memory consumption, update time and decision time of drift detectors should be taken into account for evaluation. New metrics should not focus solely on the predictive accuracy of the base classifier but incorporate performance factors of the drift detector. A potential start would be to find a suitable statistical combination to incorporate classifier accuracy with the metrics given in Table 3. This would provide a new, harmonious statistical measure that represents both the drift detector and classifier performance.

The work of Bifet [7] not only suggests that accuracy is a poor metric for evaluating drift detectors, but also that the existence of temporal dependence within datasets that contain concept drift is the cause the false alarm phenomenon found with the superior performance of the No Change detector.

Temporal dependence is defined by Zliobaite et al. [76] as “observations that are not independent from each other with respect to time of arrival” [76, p. 459]. In other words, an arriving stream element is not independent from the preceding element in regards to its time of arrival. Temporal dependence itself is not a new issue, and is a known problem in the field of time-series analysis [6]. However, its effects upon stream classification and concept drift are relatively unexplored. At the time of writing there exists very little work in the context of handling temporal dependence during stream mining and concept drift detection. The emergence of temporal dependence in streaming data has started to spawn new research, such as using temporal dependence in streaming data to assist in change detection using a Candidate Change Point model [30].

In a typical streaming scenario, arriving elements are assumed to be independent such that the class labels y_t are dependent on the features vectors x_t . When temporal dependence exists, the class labels are not independent and therefore are likely to be dependent on the previously seen labels. Temporal dependence is given mathematically by Zliobaite et al. [76] as:

$$P(y_t, y_{t-1}) \neq P(y_t)P(y_{t-1}), \quad (19)$$

where t is some timestamp and y are class labels. The authors note that this is known as *first order* temporal dependence, since only the immediately previous label is used for observation. Temporal dependence of the l th order observes the previous l labels. Temporal dependence for a class label is positive if

$$P(y_t, y_{t-1}) < P(y_t)P(y_{t-1}) \quad (20)$$

or negative if the inverse is true.

Zliobaite et al. [76] propose two approaches to account for temporal dependence in the context of stream classification. The first approach, labelled as the Temporal Correction classifier, assumes a model of temporal dependence which is used to formulate an expression for estimating the posterior probabilities. The authors consider only first order dependence in this proposal, and give the expression for estimating the maximum posterior probability as:

$$\frac{P(y_t = i | y_{t-1})}{P(y_t = i)} P(y_t = i | X_t) \quad (21)$$

Table 4 Persistent classifier performance

Dataset	Persistent classifier accuracy (%)
Electricity	85.33
Forest covertype	95.06

While this approach is simplistic, it only accounts for first order temporal dependence. While the assumption that the previous label will be known is commonly made in stream classification, any delay or error in the arrival of labels will negatively impact the performance of this method.

The second method proposed by the authors is described as Temporally Augmented classifier. In contrast to the Temporal Correction classifier, this method relies solely on preprocessing techniques. The approach involves augmenting the observation feature vector X with previously seen labels. A classification model is then trained using these augmented vectors. The prediction \hat{y}_t is then given as:

$$\hat{y}_t = h_t(X_t, y_{t-1}, \dots, y_{t-l}), \quad (22)$$

where h_t is a classification model that estimates the posterior probabilities and l is the length of temporal dependence orders. This approach is not limited to the assumption of first order dependence, as with the first proposed model. However there still exists the assumption that the previous labels will always be known.

As noted by Zliobaite et al. [76], their approach to handling temporal dependence with the Temporally Augmented classifier is simplistic, and that it is often outperformed by a Persistent classifier; a classifier that predicts that the next arriving class label is the same as the previously seen class label. This finding was also stated by Bifet [7].

Table 4 shows the results of a trained Persistent classifier on both the Electricity and Forest Covertype datasets. In both cases, particularly in the Forest Covertype dataset, the Persistent classifier outperforms the state-of-the-art classifiers using Temporally Augmented classifier to account for temporal dependence, according to the results of Bifet [7]. Simply predicting the next label will be the same as the last seen label produces higher predictive accuracy than handling temporal dependence using the Temporally Augmented approach.

The proposed Temporally Augmented approach for handling temporal dependence also only aids the baseline classifier. It does nothing to allow the drift detector itself to account for temporal dependence. The false alarm phenomenon occurs when drift detectors are subject to temporal dependence within the data, as described by Bifet [7]. As such, it should be accounted for at the drift detection level. The current state-of-the-art technique for handling temporal dependence provides no mechanisms for coping at drift detector level, it only offers a basic wrapper for baseline classifiers. Further research is required to investigate the development of new drift detection techniques, or augmentations to

existing drift detection solutions, that can account for temporal dependence in streaming data.

6 Conclusions and future research

Stream mining is a challenging problem but has valuable potential yields, especially in industry and commercial applications. Data streams offer an untapped source of qualitative and quantitative information that could be used in a multitude of different ways to boost businesses in terms of profit and efficiency. However the unbound size, unknown speed and varying characteristics of data streams make applying machine learning techniques a complex task. Whilst online classifiers capable of processing streaming data have been proposed, the task is further obfuscated by concept drift. Evolving data streams with concept drift have a distribution that shifts over time, and at varying rates of severity. Classifiers must be capable of handling concept drift by forgetting outdated information when a shift in distribution occurs.

The in-depth literature review provided in this research shows that multiple approaches for handling concept drift are available. Statistical methods monitor the underlying distribution over time, signalling alarms when a drift has been detected. Window based methods use sliding windows to detect occurring drifts rather than monitoring the whole distribution. Ensemble based detectors handle concept drift by replacing outdated classifiers within ensembles through some form of a weighted voting mechanism.

Through an in-depth, critical review of existing literature, this paper has identified the following shortcomings:

1. Outdated state-of-the-art drift detection methods.
2. A lack of benchmark datasets for evaluation.
3. Flawed evaluation metrics, including an over-reliance on classifier accuracy.
4. The inability of drift detectors to cope with additional data anomalies, such as temporal dependence.

In order to address these shortcomings, the following points address each of the identified shortfalls above in turn, providing direction for future research within the domain of stream mining and concept drift detection.

The current state-of-the-art consists of algorithms that are now somewhat dated, such as ADWIN [8], and have known flaws in them. While there have been a number of attempts to further the state-of-the-art, many methods still suffer from substantial drawbacks. Statistical based methods such as DDM and EDDM have known issues in terms of their ability to handle varying types of

drifts and in producing high rates of false alarms. Recent approaches such as DDM-OCI and LFR have aimed to solve these problems, but have fallen short. Window-based approaches like E-CVFDT still falter under sudden concept drifts. Block-based ensemble methods such as AWE and AUE are subject to dependency on the chunk size and weighting mechanisms. Incremental-based algorithms such as DWM could be improved by not explicitly storing instances and changing the statistical requirements for adding new classifiers to also consider classifier age and performance history.

A lack of benchmark datasets for evaluating is another crucial shortfall. As a result of a lack of benchmark datasets, it is common for datasets to be simulated using generators to account for the lack of benchmark datasets. While simulating data does work, the number of available generators is substantial and each often relies on various user specified parameters. The selection of the most suitable generator and corresponding parameters for a particular problem is open to interpretation. Future research should aim to produce gold-standard datasets, which would provide a collection of agreeable, accepted datasets to be used for experimentation.

Existing literature has exposed drawbacks in the metrics commonly used for evaluation drift detectors [7, 12, 76]. Some existing proposed metrics are given in Table 3, but these are particularly historic and are virtually unused in published work. This study suggests the reason for this is an issue of numerosness coupled with a lack of existing frameworks which incorporate these metrics. This research proposes that future research should aim to develop new statistical measures that capture performance properties of the drift detector and also potentially combine these with performance attributes of the baseline classifier to provide harmonious, statistically relevant metrics. An example of this is the Mean Time Ratio metric proposed by Bifet et al. [11], however this only represents the trade off between the average time to false alarms and true change detection.

The final suggestion for future research is concerned with enabling drift detection algorithms to cope with other data anomalies such as temporal dependence. This paper has discussed and portrayed the problem of temporal dependence and its impact of drift detectors. The current-state-of-the-art for handling this is merely a wrapper for the baseline classifier that augments the feature vector of arriving instances. However, temporal dependence should be handled at the drift detector level. The role of the classifier is well established in machine learning contexts; it is not classifier's responsibility to handle anomalies in the data stream. Concept drift is an anomaly that occurs in real-time data streams, and as such concept drift detection algorithms have been developed to work in

conjunction with classifiers. It follows that since temporal dependence is also a data anomaly, this should be handled by specifically designed algorithmic solutions that can cope with both temporal dependence and concept drift. The structural framework proposed by Yu and Abraham [73] in HLFRR is of particular interest in this context and forms a good starting point for future research in this field.

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interest regarding the publication of this research work.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Aggarwal CC (2007) Data streams: models and algorithms, vol 31. Springer, Berlin
- Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM, pp 1–16
- Baena-Garcia M, del Campo-Ávila J, Fidalgo R, Bifet A, Gavalda R, Morales-Bueno R (2006) Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams, vol 6. pp 77–86
- Barros RS, Cabral DR, Gonçalves PM Jr, Santos SG (2017) RDDM: Reactive drift detection method. *Expert Syst Appl* 90:344–355
- Basseville M, Nikiforov IV et al (1993) Detection of abrupt changes: theory and application, vol 104. Prentice Hall, Englewood Cliffs
- Beck N, Katz JN, Tucker R (1998) Taking time seriously: time-series-cross-section analysis with a binary dependent variable. *Am J Polit Sci* 42(4):1260–1288
- Bifet A (2017) Classifier concept drift detection and the illusion of progress. In: International conference on artificial intelligence and soft computing. Springer, pp 715–725
- Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM international conference on data mining. SIAM, pp 443–448
- Bifet A, Kirkby R (2009) Data stream mining a practical approach, University of WAIKATO, Technical report
- Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: Massive online analysis. *J Mach Learn Res* 11:1601–1604
- Bifet A, Read J, Pfahringer B, Holmes G, Zliobaite I (2013a) CD-MOA: change detection framework for massive online analysis. In: International symposium on intelligent data analysis. Springer, pp 92–103
- Bifet A, Read J, Zliobaite I, Pfahringer B, Holmes G (2013b) Pitfalls in benchmarking data stream classification and how to avoid them. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 465–479
- Blackard JA, Dean DJ (1998) Comparative accuracies of neural networks and discriminant analysis in predicting forest cover types from cartographic variables. In: Proceedings of the second southern forestry GIS conference. pp 189–199
- Brzeziński D, Stefanowski J (2011) Accuracy updated ensemble for data streams with concept drift. In: International conference on hybrid artificial intelligence systems. Springer, pp 155–163
- Brzeziński D, Stefanowski J (2014) Reacting to different types of concept drift: the accuracy updated ensemble algorithm. *IEEE Trans Neural Netw Learn Syst* 25(1):81–94
- Cattal R, Oppacher F, Deugo D (2002) Evolutionary data mining with automatic rule generalization. *Recent Adv Comput Commun* 1(1):296–300
- Chernoff H, Lehmann EL (1954) The use of maximum likelihood estimates in χ^2 tests for goodness of fit. *Ann Math Stat* 25(3):579–586. <https://doi.org/10.1214/aoms/1177728726>
- Chu F, Zaniolo C (2004) Fast and light boosting for adaptive mining of data streams. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, pp 282–292
- Cohen J (1960) A coefficient of agreement for nominal scales. *Educ Psychol Meas* 20(1):37–46. <https://doi.org/10.1177/001316446002000104>
- Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. *SIAM J Comput* 31(6):1794–1813
- De Rosa R, Cesa-Bianchi N (2015) Splitting with confidence in decision trees with application to stream mining. In: 2015 International joint conference on neural networks (IJCNN). IEEE, pp 1–8
- de Lima Cabral DR, de Barros RSM (2018) Concept drift detection based on Fisher's exact test. *Inf Sci* 442:220–234
- de Mello RF, Vaz Y, Grossi CH, Bifet A (2019) On learning guarantees to unsupervised concept drift detection on data streams. *Expert Syst Appl* 117:90–102
- Ditzler G, Polikar R (2010) An ensemble based incremental learning framework for concept drift and class imbalance. In: The 2010 international joint conference on neural networks (IJCNN). IEEE, pp 1–8
- Ditzler G, Polikar R (2013) Incremental learning of concept drift from streaming imbalanced data. *IEEE Trans Knowl Data Eng* 25(10):2283–2301
- Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. *IEEE Comput Intell Mag* 10(4):12–25
- Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 71–80
- Dua D, Graff C (2017) UCI machine learning repository. <http://archive.ics.uci.edu/ml>. Accessed 14 Oct 2019
- Duarte J, Gama J, Bifet A (2016) Adaptive model rules from high-speed data streams. *ACM Trans Knowl Discov Data (TKDD)* 10(3):30
- Duong QH, Ramampiaro H, Nørvgå K (2018) Applying temporal dependence to detect changes in streaming data. *Appl Intell* 48:4805–4823
- Elwell R, Polikar R (2011) Incremental learning of concept drift in nonstationary environments. *IEEE Trans Neural Netw* 22(10):1517–1531
- Fisher RA (1992) Statistical methods for research workers. In: Breakthroughs in statistics. Springer, pp 66–70
- Gama J (2010) Knowledge discovery from data streams. CRC Press, Boca Raton
- Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Brazilian symposium on artificial intelligence. Springer, pp 286–295
- Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. *Mach Learn* 90(3):317–346
- Gama J, Zliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. *ACM Comput Surv (CSUR)* 46(4):44

37. Gustafsson F, Gustafsson F (2000) Adaptive filtering and change detection, vol 1. CiteSeer, Princeton
38. Haque A, Khan L, Baron M (2016a) Sand: semi-supervised adaptive novel class detection and classification over data stream. In: AAAI, pp 1652–1658
39. Haque A, Khan L, Baron M, Thuraisingham B, Aggarwal C (2016b) Efficient handling of concept drift and concept evolution over stream data. In: 2016 IEEE 32nd international conference on data engineering (ICDE). IEEE, pp 481–492
40. Harries M (1999) Splice-2 comparative evaluation: electricity pricing. Technical report, The University of South Wales
41. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 97–106
42. Ikononovska E (2008) Data Expo 2009: airline on time data. <https://doi.org/10.7910/DVN/HG7NV7>
43. Ikononovska E, Gama J, Džeroski S (2011) Learning model trees from evolving data streams. *Data Min Knowl Discov* 23:128–168. <https://doi.org/10.1007/s10618-010-0201-y>
44. Jaworski M, Duda P, Rutkowski L (2017) New splitting criteria for decision trees in stationary data streams. *IEEE Trans Neural Netw Learn Syst* 29(6):2516–2529
45. Kelly MG, Hand DJ, Adams NM (1999) The impact of changing populations on classifier performance. In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 367–371
46. Kolter JZ, Maloof MA (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Third IEEE international conference on data mining, 2003, ICDM. IEEE, pp 123–130
47. Krawczyk B (2016) Learning from imbalanced data: open challenges and future directions. *Prog Artif Intell* 5(4):221–232
48. Krawczyk B, Minku LL, Gama J, Stefanowski J, Woźniak M (2017) Ensemble learning for data stream analysis: a survey. *Inf Fusion* 37:132–156
49. Krempel G, Zliobaite I, Brzeziński D, Hüllermeier E, Last M, Lemaire V, Noack T, Shaker A, Sievi S, Spiliopoulou M et al (2014) Open challenges for data stream mining research. *ACM SIGKDD Explor Newsl* 16(1):1–10
50. Liao J, Zhang J, Ng WW (2016) Effects of different base classifiers to learn++ family algorithms for concept drifting and imbalanced pattern classification problems. In: 2016 International conference on machine learning and cybernetics (ICMLC), vol 1. IEEE, pp 99–104
51. Liu G, Cheng H, Qin Z, Liu Q, Liu C (2013) E-CVFDT: An improving CVFDT method for concept drift data stream. In: 2013 International conference on communications, circuits and systems (ICCCAS), vol 1. IEEE, pp 315–318
52. McDiarmid C (1989) On the method of bounded differences. In: London mathematical society lecture note series. Cambridge University Press, pp 148–188. <https://doi.org/10.1017/CBO9781107359949.008>
53. Muhlbaier M, Topalis A, Polikar R (2004) Learn++.MT: a new approach to incremental learning. In: International workshop on multiple classifier systems. Springer, pp 52–61
54. Muhlbaier MD, Polikar R (2007) Multiple classifiers based incremental learning algorithm for learning in non-stationary environments. In: 2007 International conference on machine learning and cybernetics, vol 6. IEEE, pp 3618–3623
55. Muhlbaier MD, Topalis A, Polikar R (2009) Learn++.NC: Combining ensemble of classifiers with dynamically weighted consult and vote for efficient incremental learning of new classes. *IEEE Trans Neural Netw* 20(1):152–168
56. Nishida K, Yamauchi K (2007) Detecting concept drift using statistical testing. In: International conference on discovery science. Springer, pp 264–269
57. Page ES (1954) Continuous inspection schemes. *Biometrika* 41(1/2):100–115
58. Pesaranghader A, Viktor HL, Paquet E (2018) Mcdiarmid drift detection methods for evolving data streams. In: 2018 International joint conference on neural networks (IJCNN). IEEE, pp 1–9
59. Polikar R, Upda L, Upda SS, Honavar V (2001) Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Trans Syst Man Cybern Part C (Appl Rev)* 31(4):497–508
60. Quinlan JR (1986) Induction of decision trees. *Mach Learn* 1(1):81–106
61. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco
62. Reinsel D, Gantz J, Rydning J (2017) Data age 2025: the evolution of data to life-critical don't focus on big data; focus on the data that's big. Technical reports on IDC. <https://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf>. Accessed 14 Oct 2019
63. Rutkowski L, Pietruczuk L, Duda P, Jaworski M (2012) Decision trees for mining data streams based on the McDiarmid's bound. *IEEE Trans Knowl Data Eng* 25(6):1272–1279
64. Sethi TS, Kantardzic M (2017) On the reliable detection of concept drift from streaming unlabeled data. *Expert Syst Appl* 82:77–99
65. Street WN, Kim Y (2001) A streaming ensemble algorithm (sea) for large-scale classification. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 377–382
66. Wald A (1973) Sequential analysis. Courier Corporation, North Chelmsford
67. Wang H, Abraham Z (2015) Concept drift detection for streaming data. In: 2015 International joint conference on neural networks (IJCNN). IEEE, pp 1–9
68. Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 226–235
69. Wang S, Minku LL, Ghezzi D, Caltabiano D, Tino P, Yao X (2013) Concept drift detection for online class imbalance learning. In: The 2013 international joint conference on neural networks (IJCNN). IEEE, pp 1–10
70. Webb GI, Hyde R, Cao H, Nguyen HL, Petitjean F (2016) Characterizing concept drift. *Data Min Knowl Discov* 30(4):964–994
71. Widmer G, Kubat M (1993) Effective learning in dynamic environments by explicit context tracking. In: European conference on machine learning. Springer, pp 227–243
72. Yang M, Rashidi L, Rajasegarar S, Leckie C, Rao AS, Palaniswami M (2018) Crowd activity change point detection in videos via graph stream mining. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp 215–223
73. Yu S, Abraham Z (2017) Concept drift detection with hierarchical hypothesis testing. In: Proceedings of the 2017 SIAM international conference on data mining. SIAM, pp 768–776
74. Zliobaite I (2010) Learning under concept drift: an overview. ArXiv preprint [arXiv:10104784](https://arxiv.org/abs/10104784)
75. Zliobaite I (2013) How good is the electricity benchmark for evaluating concept drift adaptation. ArXiv preprint [arXiv:13013524](https://arxiv.org/abs/13013524)
76. Zliobaite I, Bifet A, Read J, Pfahringer B, Holmes G (2015) Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Mach Learn* 98(3):455–482

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.