One or Two Things We know about Concept Drift
A Survey on Monitoring Evolving Environments*

Fabian Hinder †‡ Valerie Vaquet † Barbara Hammer

Bielefeld University - Cognitive Interaction Technology (CITEC)

Inspiration 1, 33619 Bielefeld - Germany

October 25, 2023

Abstract

The world surrounding us is subject to constant change. These changes, frequently described as concept drift, influence many industrial and technical processes. As they can lead to malfunctions and other anomalous behavior, which may be safety-critical in many scenarios, detecting and analyzing concept drift is crucial. In this paper, we provide a literature review focusing on concept drift in unsupervised data streams. While many surveys focus on supervised data streams, so far, there is no work reviewing the unsupervised setting. However, this setting is of particular relevance for monitoring and anomaly detection which are directly applicable to many tasks and challenges in engineering.

This survey provides a taxonomy of existing work on drift detection. Besides, it covers the current state of research on drift localization in a systematic way. In addition to providing a systematic literature review, this work provides precise mathematical definitions of the considered problems and contains standardized experiments on parametric artificial datasets allowing for a direct comparison of different strategies for detection and localization. Thereby, the suitability of different schemes can be analyzed systematically and guidelines for their usage in real-world scenarios can be provided. Finally, there is a section on the emerging topic of explaining concept drift.

Keywords: concept drift, drift detection, drift localization, drift explanation, monitoring, explainability, survey

1 Introduction

The environment surrounding us is perpetually changing. While humans are trained to navigate an ever-changing environment, these changes pose challenges to many automated systems [1]. Considering monitoring and control tasks in critical infrastructure [2], manufacturing [3], and quality control [4], in order to work reliably, automatized processes and supervision algorithms need to be able to detect, react, and adapt to changes [5].

Formally, these changes can be described as *concept drift* (or drift for short) – a change in the data generating distribution [6]. They can be caused by changes in the observed process, the environment, or the sensors acquiring the data. While when monitoring a system, for instance in manufacturing or quality control, it is crucial to detect changes in the observed process as this might indicate faulty productions or general malfunctions, in automated processes, it is important to detect changes in the sensors and the environment to take appropriate actions, e.g. replacing a faulty sensor or adapting the system processing the collected data to a changed scenario [7, 8, 6].

Most often, we consider drift in *stream setups* [1, 9], where the underlying data distribution changes, requiring models to adapt or to inform a human operator to take appropriate action. This is closely related to concept

*Funding in the frame of the ERC Synergy Grant "Water-Futures" No. 951424 is gratefully acknowledged.

†Authors contributed equally

‡Corresponding Author

evolution in *continual learning* [10], commonly discussed in deep learning, where concepts might appear or vanish. Besides, data streams might suffer from temporarily extreme class imbalances or the availability of features might change over time. This can trigger the problem of so-called catastrophic forgetting where a model cannot properly process samples of a class after updating anymore. Drift is not limited to data streams but can also occur in *time-series* [11] data where the single observations are highly interdependent [12]. Here, drift mainly occurs in the form of trends. Commonly its absence is referred to as stationarity.

Besides the settings where the data samples are collected over time, considering manufacturing and quality control, frequently data is collected at different locations and processed in the scheme of *federated learning* [13]. Here, instead of gathering all the data at a global server processing is done locally and the results are then combined to get an overarching model at the server. Similar to stream learning, in this setting it is necessary to account for differences or drift in the data collected at different locations to obtain a robust global model [14]. Finally, drift must be considered when performing *transfer learning*, a strategy in deep learning [15]. The basic idea is to deal with limited data by pre-training a model on a similar task on a more extensive similar dataset and later on fine-tuning to the task and limited dataset at hand. In this work, we will focus on data streams only. However, many of the discussed strategies can be directly applied to the previously named tasks.

Considering processing drifting data streams there are two main groups of tasks. While one might be interested in keeping a valid model performing some predictive task on the data, e.g. classifying a product into different categories or estimating a property of interest (*online or stream learning*), another goal is to *monitor a system for anomalous behavior* in order to react appropriately. In this work, we will not consider online learning, as there are many surveys providing a good overview of this task [1, 9, 16] and toolboxes [17, 18]. Instead, we will focus on the monitoring scenario, which is very important in many different settings where drift is expected due to the use of sensor devices or sensitivity to changes in the environment. We focus on unsupervised drift detection as this strategy is the most relevant for monitoring settings. Besides, it is highly relevant as a prerequisite for understanding drift phenomena. Notice that this setup is also different from drift detection for online learning with limited or no label information which is also sometimes referred to as unsupervised drift detection [19]. The difference to supervised drift detection will be discussed in detail in Section 2.2. Furthermore, we also provide a formal mathematical definition of the main problems, concepts, and notions and a survey in how far these are addressed by current technologies. Moreover, we also have a look at the in depth analysis techniques like drift localization in data space and the problem of drift explanation.

The task of monitoring is to observe a system and to provide all the information necessary to enable a human operator or automatized downstream tasks to take actions that ensure that the system runs properly. Which information is required depends on the specific task [20, 21]. However, generally, it can be summarized by addressing the following questions about the drift [9]:

The first question in every setting concerns *whether (and when)* drift occurs. The task of determining whether or not there is drift during a given time period is called *drift detection* [6]. We will consider drift detection in Section 3 in detail. In case a drift is detected, additional questions need to be raised to appropriately react to the change in the data distribution.

A second question of interest might concern the severity of the drift, as this might influence which kind of action needs to be taken. Usually *drift quantification* [9] can be realized as a precursor of drift detection: As described later, many methods for drift detection estimate the rate of change by some kind of metric and trigger an alarm if those changes exceed a threshold. Thus, we will not focus on this question in great detail.

In order to take appropriate action, it is important to pinpoint the drift more precisely. While drift detection and quantification deal with the when by assigning drift-related information to the time component, i.e., finding change points, determining the rate of change, *drift localization and segmentation* [9] focus on the *where* and assign drift-related information to the data space. Consider for example quality control. There might already be an algorithm in place screening the data for known anomalies. However, in case new anomalies in the product occur it is required to detect those to analyze whether some action is required, e.g. discarding the item. In this case, it is crucial to identify the anomalous items, i.e. the drifting data samples, for further analysis. We will focus on drift localization in Section 4.

In some settings answering the discussed questions is not sufficient. There are systems in which a malfunction, i.e. the drift, causes a change in a number of features in all data points collected after the drift event. For example, this might be the case if a sensor is degrading and thus yielding changed measurements. In this case, only using drift localization does not provide much information about what actually happened. Instead, we need more detailed information of *what* exactly happened and *how* it can be described. Providing these detailed, complete, and human-understandable descriptions of ongoing drift is referred to as *drift explanation* [22]. Such methods are designed to support human operators by providing relevant information on monitoring and adaptation processes. This is relevant as the complexity of drift can easily surpass the level of information which is provided by change points or the estimate of the rate of change. Indeed, drift can manifest in a change in the correlation of several features alone, making it nearly impossible for humans to observe without machine aid. In a sense, drift explanations can be seen as the explainable AI (XAI) counterpart for drift detection: while usual XAI explains why a model makes a decision [23, 24], drift explanations provide an explanation of why a drift detector alerts for drift. This commonly makes use of various techniques, including classical XAI. We provide an overview of the most advanced drift explanation schemes in Section 5.

This paper is structured as follows. First, we provide a formalization of concept drift (Section 2), and position this paper both in the body of related work in the intersection of the stream setup and supervised and unsupervised approaches (Section 2.2). Afterwards, we focus on drift detection (Section 3) and drift localization (Section 4): We first formalize these tasks and provide a general scheme most approaches realize. We then propose a categorization of the methods and finally analyze the strategies with respect to drift and stream-specific criteria. Before concluding this survey (Section 6), we discuss drift explanation by highlighting some of the most advanced and interesting contributions (Section 5).

2 Concept Drift – defining the setup

Before we look at drift detection, drift localization, and drift explanations in detail, in this section, we formally define drift and discuss different setups for working with it.

2.1 A Formal Model of Concept Drift

In the classical setup of machine learning, one assumes that the distribution at training, testing, and application time is always the same, i.e., we assume that the data generating distribution $\mathcal{D}$ is time-invariant. In this case, a sample of size n is a collection of i.i.d. random variables $X_1, \ldots, X_n \sim \mathcal{D}$.

As discussed before, the assumption of time-invariant distributions is violated in many real-world applications, in particular, when learning on data streams. To resolve this issue from a purely formal point of view, we incorporate time into our considerations by allowing every point to follow a potentially different distribution $X_i \sim \mathcal{D}_{t_i}$ that depends on the time point t_i of observation. As it is unlikely to observe two samples at the same time, i.e., $t_i \neq t_j$ for all $i \neq j$, it is common to simply write $\mathcal{D}_i$ instead of $\mathcal{D}_{t_i}$ [6].

This relates to the classical setup if all X_i actually follow the same distribution, i.e., $\mathcal{D}_i = \mathcal{D}_j$ holds for all i, j. One speaks of *concept drift* if this assumption is violated, i.e., there exist i, j such that $\mathcal{D}_i \neq \mathcal{D}_j$ [6].

However, as pointed out by [25] this definition of concept drift depends on the used sample and not on the underlying process. In particular, it might happen that if we take two samples from the same data source over the same period of time using different sampling frequencies, one sample will have concept drift and the other will not. This makes understanding concept drift a hard problem. In order to deal with the issue, it was suggested in [25] to additionally take the statistical properties of time into account. To do so we consider an model of time $\mathcal{T}$ rather than a mere index set. We assume that there is a distribution P_T on $\mathcal{T}$ that describes the likelihood of observing a sample at time t, and a collection of distributions $\mathcal{D}_t$ for all $t \in \mathcal{T}$ albeit, in practice, only a finite number of time point is observed. Together P_T and $\mathcal{D}_t$ form what is referred to as a *drift process*

Definition 1. Let $\mathcal{T} = [0, 1]$ and $\mathcal{X} = \mathbb{R}^{d}$[1]. A *drift process* $(P_T, \mathcal{D}_t)$ from the *time domain* $\mathcal{T}$ to the *data space*

[1] All considerations below also work in a very similar way for arbitrary measure spaces. However, as some formal issues may arise

$\mathcal{X}$ is a probability measure P_T on $\mathcal{T}$ together with a Markov kernel $\mathcal{D}_t$ from $\mathcal{T}$ to $\mathcal{X}$, i.e., for all $t \in \mathcal{T}$ $\mathcal{D}_t$ is a probability measure on $\mathcal{X}$ and for all measurable $A \subset \mathcal{X}$ the map $t \mapsto \mathcal{D}_t(A)$ is measurable. We will just write $\mathcal{D}_t$ instead of $(P_T, \mathcal{D}_t)$ if this does not lead to confusion.

We can derive two particularly important types of distributions: By adding a time-stamp to every sample from the moment of its arrival the data follows what we will refer to as the holistic distribution $\mathcal{D}$. By collecting all samples observed during a certain time window $W \subset \mathcal{T}$ the data follows the windowed distribution $\mathcal{D}_W$. Formally the distributions are given by the following:

Definition 2. Let $(\mathcal{D}_t, P_T)$ be a drift process from $\mathcal{T}$ to $\mathcal{X}$. We refer to the distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{T}$ which is uniquely determined by the property $\mathcal{D}(A \times W) = \int_W \mathcal{D}_t(A) \mathrm{d}P_T(t)$ for all $A \subset \mathcal{X}$, $W \subset \mathcal{T}$ as the *holistic distribution* of $\mathcal{D}_t$[2]. Furthermore, we call a P_T non-null set $W \subset \mathcal{T}$ a *time-window* and denote by $\mathcal{D}_W(A) = \int_W \mathcal{D}_t(A) \mathrm{d}P_T(t \mid W) = \mathcal{D}(A \times W \mid \mathcal{X} \times W)$ the *mean distribution* during W.

A benefit of a drift process is that it allows us to sample data from it. This is in stark contrast to the sample-based setup, as there is no reasonable way to create a new sample from an old one. There are two ways to draw new data from a drift process. One option is to draw i.i.d. samples from the holistic distribution $\mathcal{D}$. These samples are dated-data points (T, X) that are often obtained by the following procedure in practice: First draw the time of observing X, i.e., $T \sim P_T$, and then draw X according to $\mathcal{D}_t$ assuming $T = t$, i.e., $X \mid [T = t] \sim \mathcal{D}_t$. Another sampling method that is often used in practice is to take i.i.d. samples from $\mathcal{D}_W$ for some time window W. Notice that a collection of observations that are collected during a time window W according to $\mathcal{D}$ are exactly distributed according to $\mathcal{D}_W$. Hence both ways to sample are just formal descriptions of practically relevant procedures to obtain data over time.

We derive a definition for drift based on the definition above. As we are now talking about a property of a data-generating process and not just a sample drawn from it, we must add a slight adaptation corresponding to the property that drift can actually be observed: We say that $\mathcal{D}_t$ has drift if the chance of obtaining a sample for which there is drift in the sense of the definition given above occurs is larger zero, i.e., $X_1, X_2, \ldots$ is a sample, then there are i, j such that

$$\mathbb{P}_{X_i} \overset{\text{def. } X_i}{=} \mathcal{D}_{T_i} \neq \mathcal{D}_{T_j} \overset{\text{def. } X_j}{=} \mathbb{P}_{X_j}$$

with a chance larger than zero. Due to measure theoretical reasons the number of samples actually does not play a role so we can also consider only two samples. Thus, we obtain the following definition.

Definition 3. Let $(P_T, \mathcal{D}_t)$ be a drift process. We say that $\mathcal{D}_t$ has *drift* iff

$$\mathbb{P}_{T,S \sim P_T}[\mathcal{D}_T \neq \mathcal{D}_S] = P_T^2(\{(t, s) \in \mathcal{T}^2 \mid \mathcal{D}_t \neq \mathcal{D}_s\}) > 0.$$

One may be wondering why this is different from the existence of $s, t \in \mathcal{T}$ with $\mathcal{D}_t \neq \mathcal{D}_s$. Formally speaking this has to do with P_T null sets. It might happen that the difference only occurs in such a short amount of time, that we will never see only a single sample drawn from the other distribution and thus we will never be able to observe the drift in the data. It is thus a mere artifact of the formal model, rather than the actual process.

In [25] several other, equivalent formalizations which relate to scenarios which have been considered in the literature of concept drift are given: being not equal to a standard distribution, i.e., $P_T[\mathcal{D}_t \neq P] > 0$ for all distributions P on $\mathcal{X}$; being not equal to the mean distribution, i.e., $P_T[\mathcal{D}_t \neq \mathcal{D}_{\mathcal{T}}] > 0$; different distributions for two time-windows, i.e., $\mathcal{D}_W \neq \mathcal{D}_{W'}$ for some $W, W' \subset \mathcal{T}$. One of the key findings however, which allows the development of new methods, is that drift can equivalently be formulated as data X and time T are dependent, i.e., not statistically independent:

Theorem 1. *Let $(\mathcal{D}_t, P_T)$ be a drift process from $\mathcal{T}$ to $\mathcal{X}$ and let $(T, X) \sim \mathcal{D}$ be distributed according to the*

from that, we will stick with this far simpler special case for the sake of clarity.

[2]Both existence and uniqueness of $\mathcal{D}$ are assured by the Fubini-Tonelli theorem.

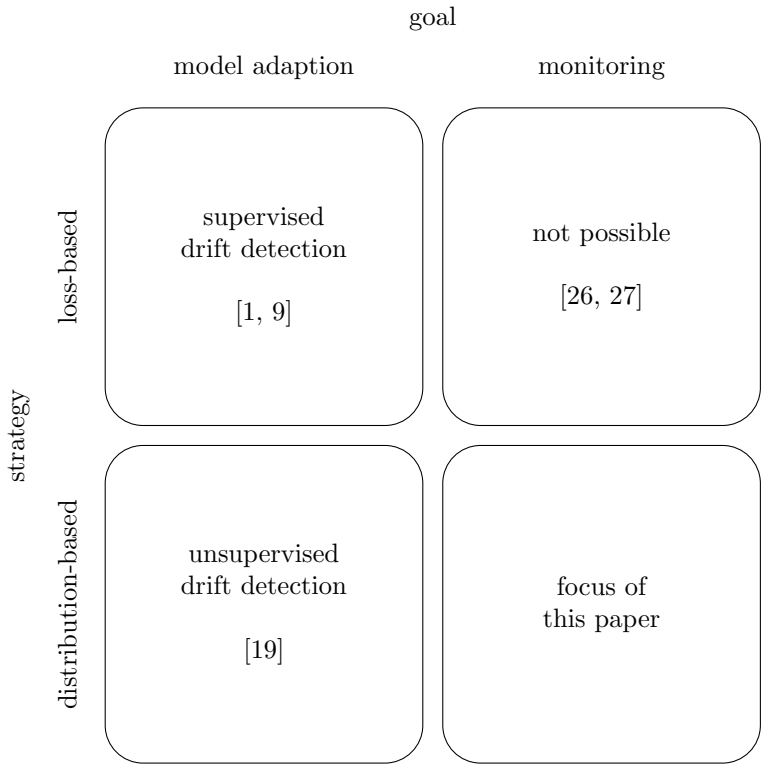


Figure 1: Display of the drift analysis categorization according to the goal and the applied strategy

holistic distribution. Then $\mathcal{D}_t$ has no drift if and only if $T \perp\!\!\!\perp X$ are statistically independent, i.e., there exist $W \subset \mathcal{T}$ and $A \subset \mathcal{X}$ such that $\mathbb{P}[T \in W, X \in A] \neq \mathbb{P}[T \in W]\mathbb{P}[X \in A]$.

2.2 Concept Drift in Supervised and Unsupervised Setups

In the last section, we provided a definition of drift based on the data-generating process. Drift is usually further categorized. A first general distinction is usually drawn based on the way drift manifests itself in time. While the distribution might completely change at time t (*abrupt drift*), there are also slower changes occurring over an interval of time. In *incremental drift*, the distribution changes smoothly over time. In contrast, in *gradual drift*, during the period of change, the samples are drawn from both distributions with different probabilities. Finally, in many real-world applications, we expect old distributions to reoccur, for instance, due to seasonalities. This phenomenon is referred to as *reoccurring drift*. Notice that some authors refer to abrupt drift as concept shift and only call a continuous change as in gradual or incremental drift concept drift. If not further specified we consider all the aforementioned kinds of drift at once.

Besides, one categorizes drift according to how the distribution changes in the data and label space. Assuming that the data stream consists of labeled-data-pairs $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, where Y is the label, in addition to the changes in the joint distribution of X and Y, the marginal and conditional distributions are of interest. Usually, a change of the posterior $\mathcal{D}_t(Y \mid X)$ is referred to as *real drift*, and a change of the marginal $\mathcal{D}_t(X)$ is referred to as *virtual drift*. Sometimes virtual drift is also called data drift, while real drift is referred to as concept drift.

As pointed out by [27], from a statistical point of view, drift in the marginal distribution $\mathcal{D}_t(X)$ and the joint distribution $\mathcal{D}_t(X, Y)$ can be modeled in a common mathematical framework, although the interpretations are of course different. In particular, in both cases drift is equivalent to the statistical dependency of time T and (labeled-)data X and (X, Y), respectively, if the time-enriched representation is (holistic distribution) considered. Real drift $\mathcal{D}_t(Y \mid X)$ on the other hand is equivalent to conditional statistical independence of label Y and time T given data X, i.e., $Y \perp\!\!\!\perp T \mid X$.

Analogous to general machine learning tasks, we can consider *supervised* settings, i.e., those that are concerned with conditional distributions, and *unsupervised* tasks, i.e., those that are concerned with the joint or marginal

distributions. While in supervised settings both real and virtual drift might be present, in unsupervised settings only virtual drift has to be considered.

As briefly discussed before, there are two main goals when facing drifting data streams. While the goal might be to *keep an accurate learning model* even if the data stream is drifting, it might be of interest to accurately detect and describe the drift in the data distribution (*monitoring*). These two goals intersect with two overarching approaches concerning the discussed settings. While in a supervised setting, one is usually relying on analyzing *model-losses*, i.e., how well a specific model can reconstruct the data or perform a prediction or forecasting task, as a proxy, in unsupervised settings one is immediately considering the *data distribution*. Structuring approaches according to these dimensions, we obtain a categorization shown in Fig. 1.

Considering this structure considerable work has been conducted on model adaption by using loss-based strategies in the supervised setting. As discussed before, here the main goal is to keep an accurate model. Thus, relying on the model loss (such as the so-called interleaved test-train loss) as an indicator of when to update the model is a reasonable choice. There are many surveys on this particular task [1, 9, 16], so we will not consider it in detail in this work.

In contrast, there are fundamental mathematical obstacles preventing that technologies which are observing the model loss can monitor general drift phenomena other than specific ones which effect the loss. The connection between model loss, model adaption, and real drift is rather vague and heavily depends on the used model class and the precise setup [27, 26]. Thus, using loss-based approaches for drift detection for monitoring setups is not suitable in general.

There exist unsupervised distribution-based approaches for both model adaption and monitoring. As argued before, we are interested in unsupervised drift detection for monitoring tasks and address this topic in the following. To the best of our knowledge, no structured survey has been conducted on drift analysis for this particular task. There exists unsupervised drift detection for model adaption such as surveyed in the work [19]. The authors focus on methods designed to trigger model adaption while using only few to none labeled data, essentially aiming at minimizing labeling costs and on maintaining high model performance rather than monitoring. Furthermore, the survey by [11] also discusses unsupervised change point detection which is a similar problem but in the context of time-series data which we do not consider here.

3 Drift Detection

As discussed before, the first important question when monitoring a data stream is *whether (and when)* a drift occurs. In this section, we will focus on how this can be accomplished by means of drift detection.

3.1 Problem Setup and Challenges

The task of determining whether or not there is drift during a time period is called *drift detection*. A method designed to perform that task is referred to as *drift detector*.

Most works on drift detection focus on the supervised scenario with abrupt drift. As discussed in [27] this problem is very different from the problem of unsupervised drift detection, i.e., when we are interested in any kind of change and not only in changes in the label distribution given data. We will focus on the unsupervised case. Surprisingly, there does not exist a formal mathematical definition of valid drift detectors in the literature, so we provide a formalization, first.

One can consider drift detectors as a kind of statistical analysis tool that aims to differentiate between the null hypothesis "for all time points t and s we have $\mathcal{D}_t = \mathcal{D}_s$" and the alternative "we may find time points t and s with $\mathcal{D}_t \neq \mathcal{D}_s$". More formally, a drift detector is a map or algorithm that, when provided with a data sample S drawn from the stream, tells us whether or not there is drift.

We can formalize that such a drift detection model is valid or accurate, respectively, in the following way: (a) the algorithm will always make the right decision if we just provide enough data, or (b) the algorithm is a valid statistical test. This leads to the following definitions:

Definition 4. A *drift detector* is a decision algorithm on data-time-pairs of any sample size n, i.e., a (sequence of) measurable maps $A_n : (\mathcal{T} \times \mathcal{X})^n \to \{0, 1\}$.

A drift detector A is *surely drift detecting* iff it raises correct alarms in the asymptotic setting, i.e., for every drift process $\mathcal{D}_t$ and every $\delta > 0$ there exists a number N such that for all $n > N$ we have

$$\mathbb{P}_{S\sim\mathcal{D}^n}\left[A_n(S) = \mathbf{1}[\mathcal{D}_t \text{ has drift}]\right] > 1 - \delta.$$

Notice that the definition is not uniform across multiple streams (or drifts if the method is local in time), i.e., for some streams it suffices to have 100 samples to correctly identify drift, for others 10,000 are not enough. This is not a shortcoming of drift detection but a common scheme for all statistical tests. If we for example want to test whether or not the mean values of two normal distributions are the same or not we have to make estimates of the mean values. The smaller the variance and larger the number of samples, the more precise our estimate, still it is never perfect. Thus, in order to derive a reasonable decision from those estimates, the difference of the estimate needs to be smaller than the difference between the estimate and the true value. Therefore, to be able to see a shift of the means that is far smaller than the variance we need more data. By considering smaller and smaller differences we can push the number of samples required ad infinitum. Yet, the statement that we need an infinite amount of data to make any statement is not very helpful.

To cope with that problem we have to take the two kinds of errors into account: A type I error occurs if there is no drift but we detect one (false alarm), and a type II error occurs if there is drift but we do not detect it. As discussed above, avoiding type II errors is not feasible. Also, as the effect of very mild drifts is usually less severe, missing one might as well be less problematic in practice. Thus, we focus on controlling the type I error.

Controlling the number of false alarms can be stated as follows: Once we provide a certain number of samples, the chance of a false alarm falls below a certain threshold. That number of samples must not depend on the data stream we consider. As this is also fulfilled for the trivial solution that never detects drift, we require the chance that the detector detects drift in case there actually is some to be larger than this threshold provided enough data from the stream is available. Here, the amount of required data is stream-specific as discussed above. If a drift detector fulfills these properties at least for some streams, we say that it is valid. If this holds for all streams, then we call the drift detector universally valid. Formally:

Definition 5. A drift detector A is *valid* on a family of drift processes $\mathfrak{D}$, iff it correctly identifies drift in the majority of cases:

$$\begin{aligned}\limsup_{n\to\infty} \sup_{\mathcal{D}_t\in\mathfrak{D},\mathcal{D}_t \text{ has no drift}} &\mathbb{P}_{S\sim\mathcal{D}^n}[A_n(S) = 1] \\ &< \inf_{\mathcal{D}_t\in\mathfrak{D},\mathcal{D}_t \text{ has drift}} \liminf_{n\to\infty} \mathbb{P}_{S\sim\mathcal{D}^n}[A_n(S) = 1].\end{aligned}$$

We say that A is *universally valid* if it is valid for all possible streams, i.e., $\mathfrak{D}$ is the set of all drift processes.

Notice that validity does not imply that A actually makes the right prediction even if they make use of larger and larger sample sizes. For a concrete case, it makes no statement about the correctness of the output except that it is more likely to predict drift if there actually is drift. This probability however holds across all streams independent of the severity of the drift. Thus, for monitoring, we need a drift detector that is universally valid and surely drift-detecting.

One is frequently additionally interested in the exact time point of the drift. This problem is usually addressed indirectly due to the fact that if there is drift observed in a certain time window the algorithm will raise an alarm which is then considered to be the time point of drift.

3.2 A General Scheme for Drift Detection

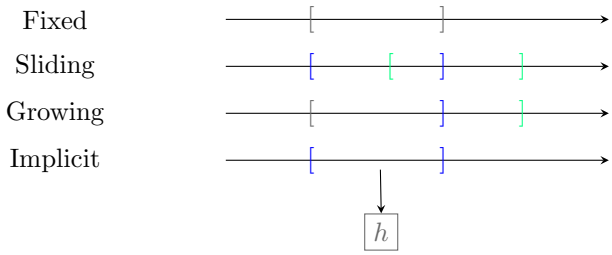


Figure 3: Illustration of Reference Window Types. Area in brackets refers to reference window $W(t), W(s)$ for time point $t < s$. Border of $W(t)$ is marked in dark blue, border of $W(s)$ in light green, overlapping borders in gray.

As discussed before, the goal of drift detection is to investigate whether or not the underlying distribution changes. As visualized in Fig. 2, drift detection is usually applied in a streaming setting where a stream of data points is arriving over time. At time t a sample $S(t)$ containing some data points which are observed during $W(t)$ and thus are generated by $\mathcal{D}_{W(t)}$ becomes available. On an algorithmic level, existing drift detection schemes can be described according to the four-staged fashion visualized in the figure. Before delving into a more detailed discussion of the different existing approaches, we describe this general scheme and briefly summarize the overarching choices for realizing the four stages.

The way these stages are implemented varies depending on the specific algorithm. In this section, we discuss some of the most prominent choices for the relevant stages 1-4 of this drift detection scheme as described in [9].

Stage 1: Acquisition of data

input:	data stream
output:	window(s) of data samples, e.g. one reference window and one containing the most recent samples

As a first step, a strategy for selecting which data points are used for further analysis needs to be selected. Depending on the strategy used (we will discuss those in Section 3.3) either one or two windows of the data are selected. Most approaches rely on sliding window strategies [9]. There are four main categories that are visualized in Fig. 3: While one can rely on a stationary fixed window that keeps the same sample until it is reset, it is also possible to use a dynamic window approach. Options for the latter are so-called growing windows, i.e. the start point stays constant and new samples are added to the window of considered data points, and sliding windows, i.e. the window keeps the same size and is shifted over the stream as new samples are arriving. Finally, the window can be implicitly realized by building a model on the corresponding data.

Apart from these examples, some approaches use preprocessing such as a deep latent space embedding [28]. We do not explain those possibilities in more detail.

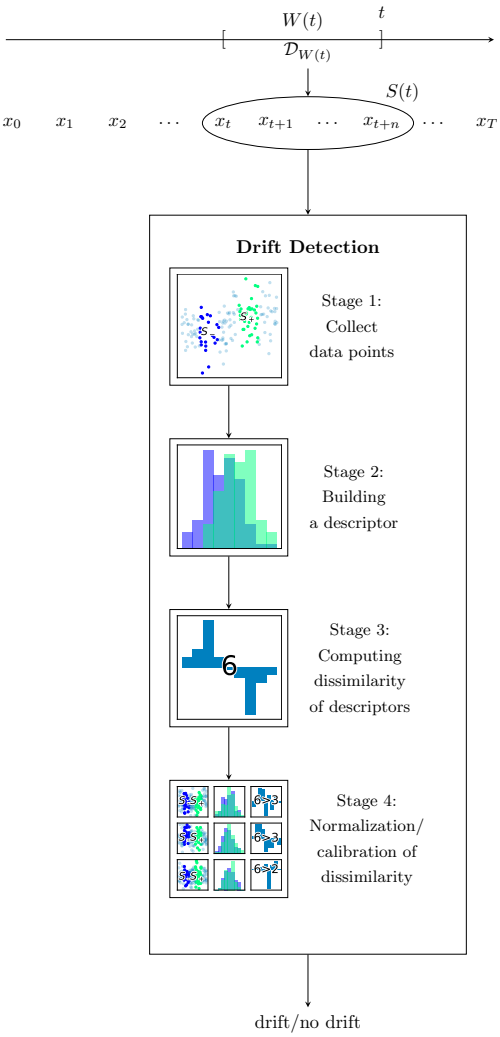


Figure 2: Visualization of drift detection from a data stream. Given a data stream, for each time window $W(t)$ a distribution $\mathcal{D}_{W(t)}$ generates a sample $S(t)$. A drift detection algorithm estimates whether the $S(t)$ contains drift or not by performing a four-stage detection scheme. Illustrated drift detector uses: two-sliding windows (stage 1), histogram descriptor (stage 2), total variance norm (stage 3), and bootstrap normalization (stage 4).

Stage 2: Building a descriptor

input: window(s) of data samples

output: possibly smoothed descriptor of window(s)

The goal of the second stage is to provide a possibly smoothed descriptor of the data distribution in the window obtained in stage 1.

Possible descriptors are grid- or tree-based binnings, neighbor-, model-, and kernel-based approaches: Binning can be considered as one of the simplest strategies. The input space is split into bins and the number of samples per bin is counted. The bins can be obtained as a grid or by using a decision tree. In order to cope with the exponential growth of the number of bins with dimensions, some authors consider one-dimensional projections [29]. Also, non-equally spaced grids have proved useful.Decision trees can be constructed randomly [30] or using a criterion that takes temporal structure into account [31, 32] which can result in better performance.

Instead of using machine learning techniques to find a binning, we can also use a machine learning model's data compression capabilities. To do so, we train the model and this way implicitly store the training data [33, 34, 35]. Later on, the information can be accessed by performing querying. Different options for prominent choices will be discussed in Section 3.3.

Neighborhood-based approaches offer robust, non-parametric methods for a wide variety of applications [36, 37]. In this case, the adjacency matrix of the k-neighborhood graph is used.

Similarly, kernel methods offer another versatile, non-parametric approach [38, 39]. Here the entire information is contained in the so-called kernel matrix which can be considered as a pair-wise similarity.

Stage 3: Computing dissimilarity

input: descriptor of window(s)

output: dissimilarity score

In the last stage, two descriptors have been obtained. The goal of this stage is to compute a dissimilarity score to analyze the windows' similarity. Although several approaches for building descriptors exist, many utilize the same or at least comparable dissimilarity measures. Popular choices are the total variation norm [40], Hellinger distance [29, 41], MMD [39, 38], Jensen-Shannon [42, 43], Kullback-Leibler divergence [30], model loss [39, 37], neighbourhood intersection[37], Wasserstein metric [44, 32, 22], mean and moment differences [45, 32, 22]. Suitable combinations of dissimilarity measures and descriptors are summarized in Table 1 and will be discussed later. Notice, that the comparison does not have to be based on a two-sample approach but can also use more advanced strategies like independence tests into account.

Stage 4: Normalization

input: dissimilarity score

output: normalized dissimilarity

As the obtained dissimilarities typically depend on both, the method, i.e., stages 1-3, and the concrete distribution at hand, it is necessary to normalize the result to obtain a useful scale. One of the most common ways to do this is to relate the dissimilarity to the statistic of a statistical test. In this case, the p-value offers a normalized scale. In the literature, a large variety of approaches are considered. Examples are the kernel two-sample test [38, 39], the Kolmogorov-Smirnov test [46], the neighborhood-statistic [37], or the HSIC-test [25, 47]. Also, other specific metrics like accuracy or the ROC-AUC also offer a normalized scale [48].

Ensemble and hierarchical approaches Some authors [9] suggest to combine multiple drift detectors. They are usually arranged in an ensemble, e.g. by combining multiple p-values after stage 4 into a single one, or hierarchical, e.g. by combining a computationally inexpensive but imprecise detector with a precise but computationally expensive validation. Although those approaches differ on a technical level, they do not from a theoretical perspective, as the suggested framework is sufficiently general.

Strategy	Type	Method	Stage 1 (Reference Window)	Stage 2	Stage 3	Stage 4	DD	DP	DL	DE
Two-Sample	MB	D3 [48]	Sliding-window	Virtual-Classifier	ROC-AUC	–	✓[a]	✗	✗	✗
	ST	Window-KS [46, 49]	Sliding-Window	Feature-wise Empirical CDF	KS-statistic	KS-distribution	✓	✗	✗	✗
	ST	MMD [38, 39]	Sliding-Window	Kernel-Matrix	MMD	Bootstrap test	✓	✗	✗	✗
	ST	HDDDM [29]	Histogram	Feature-wise Histogram	Hellinger Distance	Adaptive Threshold	✓	✗	✗	✗
	ST	PCA-CD [50]	Fixated-Window	KDE and Histograms on PCA-protection	maximum symmetrised Kulback-Leibler Divergence	Page-Hinkley test	✓	✗	✗	✗
	ST	Drift Magnitude [40, 41]	Sliding-Window	Gird Histogram	total variation / Hellinger distance	–	✓	✗	✗	✓
	ST	*kdq*-Tree [30]	Sliding-Window	*kdq*-Tree bins	sym. Kulback-Leibler Divergence	Scanner Statistic	✓	✗	✓	✗
	ST	LDD-DIS [37]	Sliding-window	k-Neighbourhood	Neighbourhood Ratio (LDD)	Bootstrap test	✓	✗	✓	✗
	ST	LSDD [51, 52]	Growing-Window	Density Estimator	L^2-Distance of Densities	Bootstrap test + FP correction	✓	✗	✗	✗
	ST	MB-DL [22]	Sliding-Window	Random Forest	Kulback-Leibler Divergence to Time Independent Model	Bootstrap test	✓	✗	✓	✓[b]
	MB	Neighbour Density Comparison [32, 36]	Sliding-Window	k-Neighbohood	Kulback-Leibler Divergence	–	✓	✗	✗	✗

	MB	Random Proj. Bin. [32, 39]	Sliding-Window	Histogram on Random Projection	Total Variation	–	✓	✗	✗	✗
Meta-Statistic	LB	Model+AdWin [28]	ML Model	-	Model-loss	AdWin-Statistic	(✓)[c]	✓	✗	✗
	ST	ShapeDD [53]	Consecutive Sliding-Windows	Kernel-Matrix	MMD	Shape Match + Bootstrap Test	(✓)[d]	✓	✗	✗
Block Based	ST	DAWIDD[25]	Sliding-Window	Kernel-Matrix	HSIC	Bootstrap test	✓	✗	✗	✗
	CL	KCpD[54, 55, 56]	Sliding-Window	Kernel-Matrix	Kernel-Variance	None / Model Selection Heuristic	✓	✓	✗	✗
	MB	Moment Tree Binning [32]	Sliding-Window	Moment Tree	Total Variation	–	✓	✗	✓	✓[b]
	MB	Drift Segmentation [57, 22]	Sliding-Window	Kolmogorov/Moment Tree		–	✗	✗	✓	✓[b]

Table 1: Overview of unspervised drift analysis methods from the literature. Headers stand for: Drift Detection (DD), Drift Pinpointing in time (DP), Drift Localization (DL), Drift Explanation (DE). Stage 1 current window is sliding window in all cases. Type refers to the type of normalization strategy used: Statistical Test (ST), model Loss Based (LB), virtual classifier / Model Based (MB), and CLustering heuristic based (CL). [a] depends on model but low requirements [32], [b] used by [22] as basis, [c] depends on model, dataset and training [26, 27], [d] for distant abrupt drifts

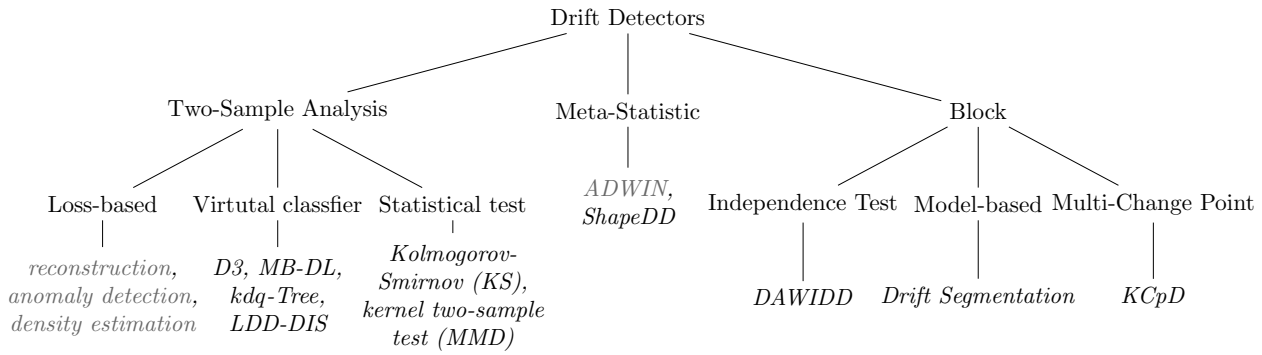


Figure 4: Taxonomy of Drift Detection/Localization approaches discussed in this paper. Methods in marked in gray are supervised.

3.3 Categories of Drift Detectors

So far, we formally defined the properties a drift detection algorithm should fulfill and described on an algorithmic level how different approaches can be implemented. In this section, we focus on concrete approaches. We propose a categorization according to the main strategies of the approaches, relying either on an analysis of two samples, meta-statistics, or a block-based strategy. We present methods organized according to the taxonomy presented in Fig. 4. An overview of the approaches considered in this survey is presented in Table 1. We provide the overall strategy, whether the method follows a model-based, statistical test-based, loss-based, or clustering heuristic-based scheme, and how the stages discussed in the last section are realized.

3.3.1 Two-Sample Analysis Based

Formally drift is defined as the difference between two time points. This can be detected by applying a two-sample test, i.e., a statistical test that checks whether or not two data distributions are the same. Drift detectors based on this idea are the most common ones in the literature. In order to perform such a test we split our sample $S(t)$ into two samples $S_-(t)$ and $S_+(t)$ and then apply the test to those. The construction of the descriptor, distance measure, and normalization (stages 2-4) are then left to the used testing scheme. Besides classical statistical tests, there also exist more modern approaches that make use of advanced machine learning techniques.

As stated above, in order to apply this scheme we need to split the obtained sample into two sub-samples which are then used for the test. This step is crucial as an unsuited split can have a profound impact on the result. Choosing an inappropriate step can diminish the test's performance. In severe cases, it can make the drift vanish and thus undetectable as we consider time averages of the windows. However, there exists theoretical work that suggests that the averaging out does not pose a problem practice [53].

From a more algorithmic perspective, there are essentially three ways the testing procedure is approached. Loss-based and virtual classifier-based approaches rely on machine learning techniques while statistical test-based approaches rely on statistical tools. We will discuss those in the following.

Loss-based approaches A large family of loss-based approaches uses machine learning models to evaluate the similarity of newly arriving samples to already received ones. In this case, the reference window (stage 1) is implicitly stored in a machine learning model which is also used as a data descriptor (stage 2). The dissimilarity is usually given by the model loss. It is further analyzed using drift detectors which are commonly used in the supervised setup [8, 58, 59, 60, 61] and serve as a normalization (stages 3 & 4).

There are several candidates implementing this strategy. One of the most common model choices are auto-encoders which compress and *reconstruct* the data [39]. Here, the idea is that the models are successfully compressing and reconstructing data generated by the distribution at training time but have difficulties generalizing to data generated by another distribution, i.e. if concept drift occurred in between changing the data distribution. Thus, an increase in the mean reconstruction error indicates the presence of drift.

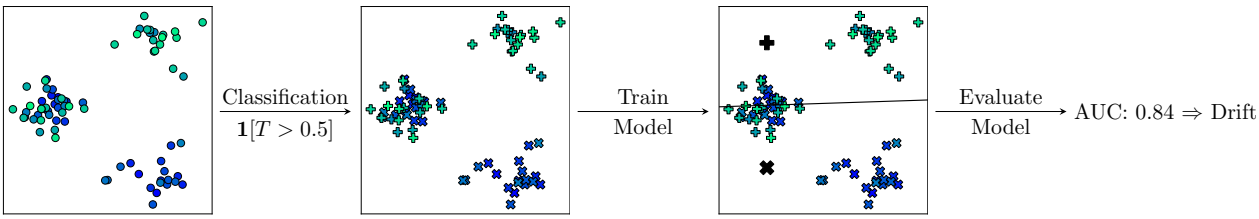


Figure 5: Visualisation of Virtual Classifier based Drift Detection. 1. Collect data (moment of arrival is color coded: dark blue to green), 2. Mark all samples arrived before a certain time as class -1 (cross) and after as class +1 (plus), 3. Train model to distinguish class -1 and +1, 4. Evaluate model, if performance better than random chance then there is drift.

Other popular model choices are models like 1-class SVMs or Isolation Forests. Originating from *anomaly detection*, they provide an anomaly score that estimates how anomalous a data point is. The rationale for using this approach for drift detection is that data points from a new concept look like anomalous data points from the point of view of the old concept. An increase in the mean anomaly score indicates drift.

Finally, *density estimators*, which are designed to estimate the likelihood of observing a sample, can be applied to detect drift. Here the idea is that a sample from a new concept is assumed to be unlikely to be observed in the old concept, resulting in a low occurrence probability. Thus, a decrease in the mean observation probability indicates drift [62, 63].

Although these methods are quite popular as they are closely connected to supervised drift detection, they also face the same issues. In particular, it has been shown on a theoretical [26] and practical [27] level that such approaches do not work well in discovery tasks monitoring for drift as the drift might be irrelevant to the decision boundary that can be learned by the model. In particular, the resulting methods are neither surely drift detecting nor universally valid as the connection of model loss and drift is rather loose. Drift might not influence the classification as attempted by the model and thus not be detectable by means of model-loss-based detection schemes. The suitability strongly depends on the characteristics of the drift and the chosen model class. We will thus not focus on them for the rest of this paper.

Virtutal classifier based approaches A different approach using machine learning models is based on the following idea of virtual classifiers[64, 65]: If a classifier performs better than random guessing, then it must exploit some properties in the data. In particular, the distribution of the single classes cannot be the same. Otherwise, it would not be able to tell the samples apart.

This idea can be employed for drift detection as follows (see Fig. 5 for an illustration): All data points of a reference sample $S_-(t)$ and a current state sample $S_+(t)$ are stored (stage 1). All data points in the reference sample $S_-(t)$ are labeled as class -1 and all data points in the current state sample $S_+(t)$ as $+1$. They are used to train a model describing the data stream (stage 2). It can either be trained anew or updated every time new samples are arriving. The model's performance, e.g., its accuracy, serves as a similarity measure for our drift detection scheme (stage 3). Depending on the precise setup, i.e., the relative size of samples, used performance measure, etc., common classification metrics can serve as normalized scores indicating drift if the model performs significantly better than random chance (stage 4).

For a practical realization of this approach, it makes sense to use k-folds to make optimal usage of the provided data without testing on the train set. Furthermore, using ideas from statistical learning theory, the model loss can be further refined in order to obtain actual p-values[66, 64]. However, this does not work well in practice as the bounds provided by learning theory are too loose. Notice, the choice of the used model class determines which types of drift can be detected in which intensity as it needs to be capable of capturing the change [32]. Furthermore, the authors showed that for many learning models virtual classifiers provide surely drift detecting algorithms assuming an appropriate split point is chosen. It is also suggested that in these cases the resulting algorithms are also universally valid. Also, as pointed out by [53] the chance of choosing an invalid split point is essentially zero.

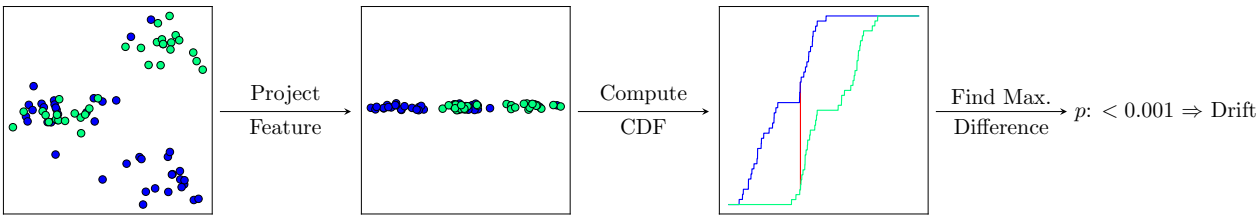


Figure 6: Visualisation of Kolmogorov-Smirnov test for Drift Detection. 1. Collect data (two-windows; $S_-(t)$ blue and $S_+(t)$ green), 2. Feature-wise projection, 3. Compute largest difference (red line) between CDFs ($\hat{F}_{t-}$ of $S_-(t)$ and $\hat{F}_{t+}$ of $S_-(t)$) of feature-wise before and after distribution, 4. Use analytic H_0 distribution to obtain p-value.

We will consider D3 [48] as an example candidate of this class. In the default implementation by the authors, it uses two sliding windows of equal size (stage 1), logistic regression (stage 2), and ROC-AUC which is already a normalized score (stages 3 & 4). There are several other approaches that use similar ideas but different models and/or evaluation metrics [37].

Statistical test based approaches The approaches discussed so far are usually based on intuitive considerations and can be considered as ad-hoc approaches. Another way to deal with the problem is to turn to mathematical and statistical tools that were designed for this purpose, namely two-sample tests. In contrast to the previous approaches those usually come with theoretical justifications but sometimes lack knowledge of practical situations.

Most classical two-sample tests are designed for one-dimensional data only. The most prominent candidate from this category that plays an important role in drift detection is the *Kolmogorov-Smirnov (KS) test*. We have visualized the testing procedure in Fig. 6. The test requires two collections of one-dimensional samples as input (stage 1). Apart from this requirement, the data acquisition strategy can be chosen freely. It then computes the empirical cumulative distribution function (cdf) (stage 2):

$$\hat{F}_{S_\pm(t)}(x) := \sum_{s \in S_\pm(t)} \mathbf{1}[s < x]$$

which counts the number of observations smaller than the input. The similarity is then computed as the maximal distance of the two cdfs (stage 3)

$$\hat{d}(S_-(t), S_+(t)) := \sup_x \left| \hat{F}_{S_+(t)}(x) - \hat{F}_{S_-(t)}(x) \right|.$$

It has been shown in the original paper that in case $S_-(t)$ and $S_+(t)$ follow the same distribution (i.e. $\mathcal{D}_{W_-(t)} = \mathcal{D}_{W_+(t)}$, which is the case if there is no drift), then for large sample sizes $\hat{d}$ follows a distribution that does not depend on the distributions $\mathcal{D}_{W_\pm(t)}$ [67] allowing us to directly compute a p-value which can be used as a normalized scale (stage 4).

If we are dealing with higher than one-dimensional data, we can apply the procedure to every dimension separately (see Algorithm 1). The p-values can then be combined using various techniques. One of the simplest is to take the minimum as it suffices that there is drift in one of the features (Bonferroni correction). A drawback of this approach is that we do not consider drift that only affects correlations as those are not considered. One way to deal with this issue is to use randomly chosen projections [32, 39]. However, this does not perform well in practice [42]. There are incremental versions of the KS-test that allow for a fast computation in a streaming setup [49].

Another important statistical test in drift detection is the *kernel two-sample test* [38, 39] which is based on Maximum Mean Discrepancy (MMD). MMD can be thought of as the optimal performance of a collection of

Algorithm 1 Kolmogorov-Smirnov Windowing

procedure KSWIN((x) data stream d-dimensional, $n_{\max 2}$ current window size, $n_{\max 1}$ reference window size, p_{detect} detection threashold)
  Initialize Windows $W_1 \leftarrow [], W_2 \leftarrow []$
  while Not at end of stream x **do**
    $W_2 \leftarrow W_2 + [x]$ ▷ Add new sample
    if $|W_2| > n_{\max 2}$ **then**
      $x \leftarrow \text{POP}(W_2)$ ▷ Move sample from current to reference window
      $W_1 \leftarrow W_1 + [x]$
    end if
    if $|W_1| > n_{\max 1}$ **then**
      $\text{POP}(W_1)$ ▷ Drop oldest sample
    end if
    if $|W_1| > n_{\min}$ **then**
      $p \leftarrow 1$
      $W_1' \leftarrow \text{UniformSubsample}(W_1, n_{\max 2})$
      for $i \in \{1, \ldots, d\}$ **do**
        $w_1 \leftarrow \{x_i \mid x \in W_1'\}$, $w_2 \leftarrow \{x_i \mid x \in W_2\}$ ▷ Extract i-th feature from each sample
        $p_d \leftarrow \text{TestKS}(w_1, w_2)$
        $p \leftarrow \min(p_d, p)$
      end for
      if $p < p_{\text{detect}}/d$ **then**
        Alert drift
      end if
    end if
  end while
end procedure

models:

$$\text{MMD}(P, Q) := \max_{\|f\|_{\mathcal{H}} \leq 1} \left| \mathbb{E}_{X \sim P}[f(X)] - \mathbb{E}_{Y \sim Q}[f(Y)] \right|.$$

In order to compute this efficiently one makes use of kernel methods, which allow us to estimate the MMD using

$$\begin{aligned}
&\widehat{\text{MMD}}_b(X_1, \ldots, X_n \ , \ Y_1, \ldots, Y_m) \\
&:= \frac{1}{n^2} \sum_{i,j=1}^{n} k(X_i, X_j) - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(X_i, Y_j) + \frac{1}{m^2} \sum_{i,j=1}^{m} k(Y_i, Y_j) \\
&= \begin{pmatrix} \frac{1}{n} \\ \vdots \\ \frac{1}{n} \\ -\frac{1}{m} \\ \vdots \\ -\frac{1}{m} \end{pmatrix}^{\top} \underbrace{\begin{pmatrix} k(X_1, X_1) & \cdots & k(X_1, X_n) & k(X_1, Y_1) & \cdots & k(X_1, Y_m) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(X_n, X_1) & \cdots & k(X_n, X_n) & k(X_n, Y_1) & \cdots & k(X_n, Y_m) \\ k(Y_1, X_1) & \cdots & k(Y_1, X_n) & k(Y_1, Y_1) & \cdots & k(Y_1, Y_m) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(Y_1, X_1) & \cdots & k(Y_1, X_n) & k(Y_1, Y_1) & \cdots & k(Y_1, Y_m) \end{pmatrix}}_{=K} \begin{pmatrix} \frac{1}{n} \\ \vdots \\ \frac{1}{n} \\ -\frac{1}{m} \\ \vdots \\ -\frac{1}{m} \end{pmatrix}
\end{aligned}$$

where k is a kernel function and the matrix K is the kernel matrix

Similar to the KS test, the MMD test requires two collections of sample points (stage 1). However, in contrast to KS those do not need to be one-dimensional, indeed, the test can even deal with non-vectorial data as long as a kernel can be defined on that data. The descriptor is then given by the kernel matrix K based on the reference and current state sample (stage 2). This allows the computation (an estimate) of the MMD by multiplying it with an appropriate weight vector from both sides (stage 3). As the resulting score is heavily influenced by the choice of the kernel and the dataset a normalization is necessary. To do so the authors [38] suggest either

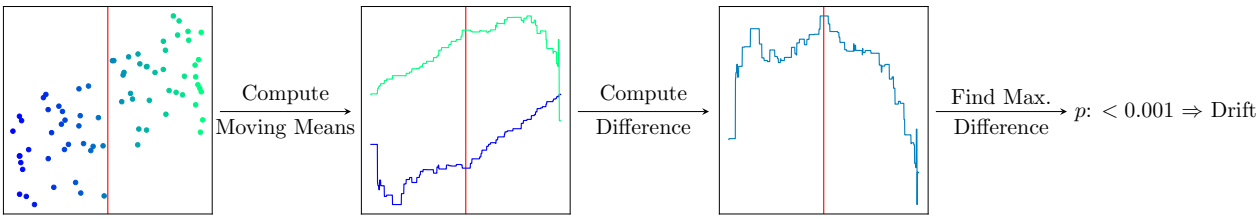


Figure 7: Visualisation of AdWin Drift Detection. 1. Collect data ($S(t)$ values between 0 and 1; red line marks drift), 2. Compute moving means ($\mu_{s-}(t)$ blue $\mu_{s+}(t)$ green), 3. Find largest difference, 4. Use analytic H_0 distribution to obtain p-value.

using a normalization based on the higher moments of the $\widehat{\text{MMD}}_b$ estimator or using a permutation boot-strap approach that compares the MMD of the original split with randomly permutated ones which both provide us with p-values (stage 4). Unfortunately, the moment-based approach tends to be less robust. Thus, we will make use of the permutation approach. There are several more approaches that follow similar lines or arguments based on various descriptors or metrics [55, 68, 69, 51, 52, 70].

A drawback of all the aforementioned approaches is that the selection of the split point is somewhat random but still has a huge influence on the resulting performance. Furthermore, as we perform the testing on sometimes heavily overlapping time windows we face the multiple testing problem, i.e., the fact that statistical analysis and interference tools become less reliable if applied to the same set of observations over and over again. One way to deal with this issue is using meta-statistics which do not consider every time point/time window in isolation but rather compare the resulting values.

As the notion of validity of a drift detector is derived from considering the problem as a statistical test, most test base approaches are valid if test specific criteria are meat, e.g., as we will see in Section 3.4 the feature-wise Kolmogorov-Smirnov tests assumes that the drift is not contained in the correlation only. However, it is not clear how to derive a surely drift detecting algorithm for a statistical test: Although, the p-value tends to 0 as the number of samples increases, assuming there is drift, it does not tend to 1 in the non-drifting cases so that an appropriate threshold has to be chosen which can be complected in practice. Furthermore, as in all two-sample cases, a split point has to be chosen which can have a strong effect on the performance of the test [32, 53] (see Section 3.4). However, the chance of choosing a split point that invalidates the detection is very unlikely [53].

3.3.2 Meta-Statistic Based

So far we have been dealing with two-sample approaches. In a sense, those are the simplest approaches as they consider every time point in the stream separately which leads to issues like the multiple testing problem, sub-optimal sensitivity, and high computational complexity. Meta-statistic approaches try to deal with some of these issues by not considering each estimate separately but rather combining the values of several estimates to get better results. Though there are several algorithms that combine the result of multiple drift detectors (ensemble approaches), those are usually applied at the same time point. To the best of our knowledge, there are only very few algorithms that fall into this category. We will describe two algorithms in detail.

AdWin AdWin [59] stands for ADaptive WINdowing and is one of the most popular algorithms in supervised drift detection. It takes values between 0 and 1 as input, which are interpreted as model losses. However, other inputs such as p-values are also possible. We have illustrated the workings of AdWin in Fig. 7: First, it stores the values in a single growing or sliding window $S(t)$ (stage 1). For every time point $s \in W(t)$, this window is split $S(t) = S_{s+}(t) \cup S_{s-}(t)$ and the mean value $\mu_{s\pm}(t)$ (and variance) are computed for both halves (stage 2). The (variance normalized) difference of mean values then provides the statistic of interest (stage 3):

$$\hat{d}(t) = \sup_{s \in W(t)} |\mu_{s+}(t) - \mu_{s-}(t)| .$$

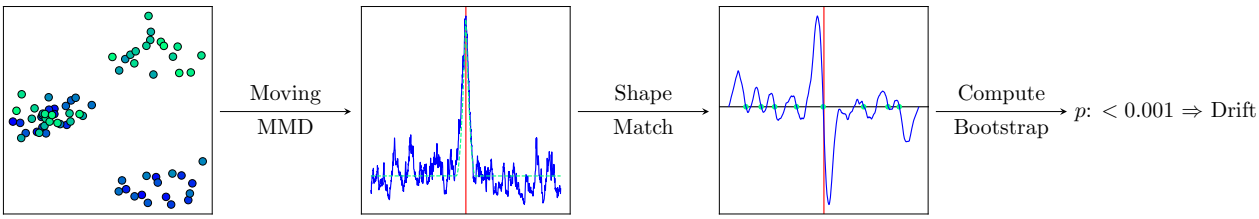


Figure 8: Visualisation of Shape Drift Detector. 1. Compute MMD for all time points ($\hat{\sigma}$, dotted line shows theoretically expected shape σ, red line indicated time point of drift), 2. Match obtained shape with theoretically expected one ($\hat{\sigma} * w$), 3. Candidate points are where the match score changes from positive to negative (black line is 0, dots mark candidates), 4. Compute H_0 distribution for MMD using permutation bootstrap at candidate points to obtain p-value.

Using the assumption that the samples are Bernoulli random variables, i.e., take on the values 0 and 1 only, as is the case for classification errors one can compute p-values for the H_0-hypothesis that the performance of the model only becomes better over time (stage 4). If the H_0-hypothesis is rejected then the model's accuracy has decreased indicating a drift. The time point of the largest discrepancy is considered as the moment the drift took place.

There are algorithmic solutions that allow for an efficient, incremental computation of the AdWin statistic. However, as stated before the connection of model-loss as drift is rather vague [27, 26] thus we will exclude AdWin from further considerations.

ShapeDD The Shape Drift Detector [53] is another meta-statistic-based drift detector. In contrast to AdWin it focuses on the discrepancy of two consecutive time windows, a quantity referred to as drift magnitude [40]:

$$\sigma_{d,l,\mathcal{D}.}(t) = d(\mathcal{D}_{[t-2l,t-l]}, \mathcal{D}_{[t-l,t]}).$$

Here d is allowed to be any (quasi-)metric. This includes several dissimilarity measures used in other drift detectors: total variation norm [40], Hellinger distance [29], MMD [39, 38], Jensen-Shannon-metric (A metric derived from the Kullback-Leibler divergence [30]), feature-wise maximum KS statistic [46], and classification accuracy linear models with non-linear preprocessing [48]. The core finding of [53] is that the drift detector is based on is the fact that (for quasi-norms d) the graph of σ takes on a characteristic shape (see Fig. 8) in case there is drift. This shape does not depend on the dataset or the used distance measure but only on the length of the sliding window. Thus, in contrast to many other drift detectors, ShapeDD directly takes the fact that it makes use of sliding windows into account. In case there are several drift events, the same shape is repeated once for each drift assuming the drifts are sufficiently far apart. Finding those parts of σ that match the characteristic shape allows for a precise pinpointing of the drift event.

More algorithmically, in the default setup as described by the authors, ShapeDD uses the MMD. We thus use two consecutive sliding windows to compute the MMD using the kernel matrix $\hat{\sigma}(t)$ (stages 1-3). This can be done in an online fashion in linear time with respect to the window length. This has two advantages: First, the MMD is known to work quite well for a large number of different datasets. Second, the computationally expensive part of the MMD-based drift detector is the computation of the normalization rather than the simple statistic.

To match the shape we then compute the convolution $(\hat{\sigma} * w)(t)$ with $w(t) = -1/l$ for $-2l \leq t < -l$, $= 1/l$ for $-l \leq t < 0$ and 0 otherwise, which can be done iteratively in constant time and check for consecutive values where the sign changes from positive to negative, indicating a perfect match of the shape. The obtained points are then used to compute the MMD in a two-window fashion using the found position as the split point (stage 4). As a consequence, most potential split points are not considered in the first place, reducing the average computational complexity of the method. Furthermore, it is less likely to find several candidate points in close proximity, reducing the change of a multiple testing problem and thus increasing the reliability of the method.

Furthermore, in some cases, the precise time point of the drift event can be of interest, which is not provided by the two-window approaches. Additionally, it is unlikely that the same drift is found twice which is also not true for the other approaches. However, the characteristic shape is, in fact, an artifact that results from the way the sampling procedure interacts with a single drift event, and thus no longer holds true if we consider a different windowing scheme (stage 1), several drift events in close succession, or gradual drift.

One way to solve the latter issue is to make use of even more advanced meta-statistics that analyze the entire data block at once.

Besides from that the authors discuss in the original paper that ShapeDD resolves the problem of choosing the right split point, which together with the validity of the MMD two-sample test shows that the method is valid for all drift processes with abrupt drifts that are sufficiently far apart. However, as ShapeDD essentially reduces the drift detection to the application of MMD, it suffers the same issues regarding being surly drift detecting.

3.3.3 Block Based

In contrast to all other drift detectors considered so far, block-based methods do not assume a split of the data into two windows at any point. Instead, they take an entire data segment into account and analyze it at once. To the best of our knowledge, so far there is only one block-based drift-detection algorithm in existence.

Independence Test based DAWIDD [25] is derived from the formulation of concept drift as a statistical dependence of data X and time T and thus resolves drift detection as a test for statistical independence. Here we will make use of the HSIC-test [47] which similarly to MMD makes use of kernel methods. However, instead of searching for a map that discriminates the two datasets it searches for a pair of maps that increases the correlation, i.e., $\sup_{f:\mathcal{T}\to\mathbb{R},g:\mathcal{X}\to\mathbb{R}} \operatorname{cov}(f(T), g(X))$ where similar to MMD f and g are found using kernel-methods. The test requires a single collection of data points and thus a sliding window (stage 1). If available, the real observation time points can be used, otherwise [25] suggested to simply making use of the sample id, i.e., sample X_i was observed at time $T_i = i$. Using HSIC we compute the kernel matrix of data K_X and time K_T as a descriptor (stage 2). The HSIC statistic is then a measure for the dependence of data X and time T and is estimated by $\operatorname{trace}(K_X H K_T H)$, where $H = I - n^{-1}\mathbf{1}\mathbf{1}^\top$ is the kernel-centering matrix (stage 3). Similar to MMD the HSIC can be normalized using higher moments or a permutation bootstrap approach (stage 4). Due to better performance, we make use of the latter. Notice that if the actual observation time is not available we can use the same time kernel matrix K_T and thus precompute HK_TH as well as the permutated versions resulting in a drastic reduction in computation time.

Compared to the other approaches, DAWIDD makes the fewest assumptions on the data or the drift. Not only can it deal with gradual, and reoccurring drift but can even be applied in cases where there is no linear, temporal order as in the case of federated learning where every computational node can experience drift at different points in time. However, this generality comes at the cost that we usually need more data to obtain the same level of certainty. We can consider this as the usual complexity-convergence trade-off found in all areas of machine learning.

As DAWIDD is again a statistical test it is also universally valid but suffers the same issues regarding surely drift detectability. However, in contrast to the two-sample test base methods, we do not have to choose a valid split point which can be problematic in practice.

Clustering based Another way to perform block-based drift detection is provided by clustering-based methods. In contrast to independence test-based approaches, these assume that there are only finitely many, abrupt drifts in the considered time interval. The methods then proceed by clustering the data points with the additional restriction that all data points that belong to one cluster have been observed at consecutive time points. This is a reasonable approach as the variance of several data points coming from the same concept is small, while the variance of data points coming from different concepts is large. Furthermore, since the time points of all

observations belonging to one concept have to be a time interval and every time point has to belong to one interval it suffices to find the boundary points of the intervals which then correspond to change points of the drift process. Using a distributional variance measure V, i.e., a generalization of variance, such algorithms solve an optimization problem of the following form for a predefined number n:

$$\arg\min_{t_0<\cdots<t_n} \sum_{i=0}^{n-1} w(t_{i+1}-t_i)V(\mathcal{D}_{(t_i,t_{i+1}]}),$$

where $\mathcal{T} = (t_0, t_n]$ and w is a weighting function.

An instantiation of this approach was proposed by [54, 71] making use of a kernalized version of variance, i.e., $V(P) = \sup_{\|f\|\leq 1} \operatorname{var}_{X\sim P}(f(X))$, that is closely connected to the MMD and constant weighting $w(\Delta) = \Delta/n$ [56]. Thus, for $X_1, \ldots, X_m \sim P$ we have the estimate $\hat{V}(X_1, \ldots, X_m) = \frac{1}{m}\sum_{i=1}^{m} k(X_i, X_i) - \frac{1}{m^2}\sum_{i,j=1}^{m} k(X_i, X_j)$. Due to the linear nature of the kernel-variance, i.e., it can efficiently be computed from the integral kernel matrix, the problem can efficiently be solved using a dynamic programming approach that assures to always find the optimal solution in polynomial time. The resulting algorithm is commonly called Kernel Change-point Detection (KCpD). Later on, [56] introduced a heuristic to estimate the number of change points by making use of ideas that originate in model selection. More precisely, one considers the same objective as before for different numbers of split points n. If there are more than n change points then we can find a good split and end up with a much smaller value for $n+1$. On the other hand, if there are no more than n change points, then the obtained decrease of the objective is only due to statistical insufficiencies and therefore comparably small.

From a more algorithmic point of view, KCpD essentially searches for blocks along the main diagonal of the kernel matrix so that the mean value of the entries inside the blocks is maximized. The number of blocks is then chosen such that more blocks no longer increase that value significantly.

Notice that there is a close connection between clustering-based algorithms and independence test-based algorithms: In the case of HSIC, by restricting the class of functions $f : \mathcal{T} \to \mathbb{R}$ to indicator functions on intervals, i.e., $f(t) = 1$ for $a < t \leq b$ and 0 otherwise, we force the test to search for functions g such that $g(X)$ is large whenever $T \in (a, b]$ and small everywhere else. Since the norm of the functions is bounded, this implies that the variance of $g(X) \mid T \in (a, b]$ is small. Thus, KCpD can be considered as an instantiation of an independence-based detection scheme.

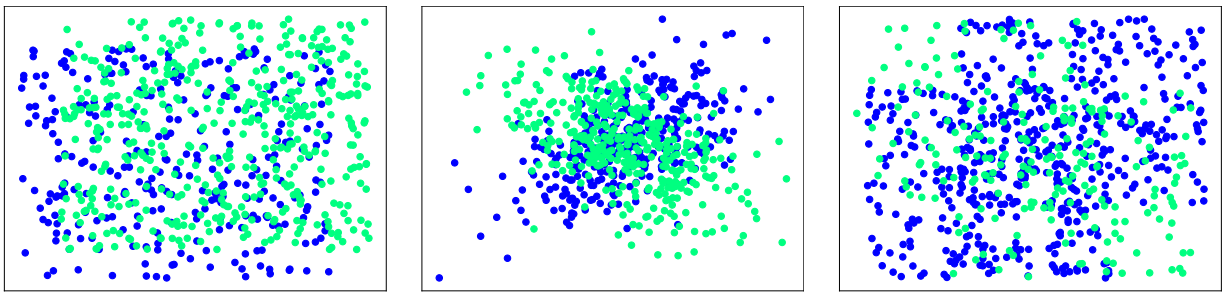
Since KCpD is a mainly heuristic methods it is hard to make any statement about its limiting behaviour. However, in the statistic of the 1-split point case is very similar to the once considered by [32] furthermore it is well known that in many cases kernel-estimates have uniform convergence rates. It is thus reasonable that one can derive universally valid surely drift detesting methods that make use of the same ideas.

There are more algorithms that use similar ideas [72].

Model based Besides the classical kernels which are predefined and not dataset specific, we can also construct new kernels using machine learning models. In [32], random forests with a modified loss function that is designed for conditional density estimation, so-called moment trees [31], are used to construct such kernels. To do so the model is trained to predict the time of observation T from the observation X. The resulting kernels show drastic improvements in drift detection tasks [32]. We can also apply this procedure directly to obtain model-based block-based approaches that can be thought of as an extension of the classifier-based two-window approaches to continuous time by removing the time discretization. The relation between the resulting approaches to DAWIDD is then very similar to the relation of MMD to the model-based two-window approaches.

3.4 Analysis of Strategies

So far, we categorized different drift detection schemes and described them according to the four stages discussed in Section 3.2. In this section, we will consider the different strategies on a more practical level and investigate experimentally in which scenarios which drift detection method is most suitable. For this purpose, we identified four main criteria that describe the data stream and the drift we aim to detect: we investigate the role of the



uniform Gauss two overlap

Figure 9: Illustration of used datasets (default parameters, original size). Concepts are color coded (Before drift: blue, after drift: green).

drift strength, the influence of drift in *correlating features*, the *data dimensionality*, and the *number of drift events*. To cover the strategies described in Section 3.3, we select one representative technique per category. As these approaches are structurally similar, from a theoretical viewpoint they carry the same advantages and shortcomings. We will present and discuss our findings in the remainder of this section.

Experimental Setup *Datasets:* For our experiments, we consider three 2-dimensional, synthetic datasets with differently structured abrupt drift (see Fig. 9). We use modifications of these datasets to evaluate the properties of the discussed drift detection methods.

For the first dataset, we draw samples from a *uniform* distribution on a square. The drift is induced by performing a uniform shift along the diagonal, i.e., in x- and y-direction. Here, the drift intensity describes the length of that shift. Potential extra dimensions contain uniform noise. In this dataset, the drift is thus described by means of a shift, which is not uncommon in practice, but a rather simple setup.

The second dataset is generated by drawing samples from a *Gauss* (normal) distribution. The drift intensity is characterized by the absolute value of the covariance. The drift changes the sign of the covariance, i.e., the drift corresponds to a 90^o rotation of the distribution. Potential additional dimensions are normally distributed with covariance 0. In this dataset, the drift is thus encoded in the correlation only, however, normal distributions are a relatively easy problem to deal with for many algorithms.

For the third dataset, samples are drawn from *two overlapping* uniform distributions on squares. The intensity describes the offset of the two squares. The drift corresponds to a 90^o rotation of the distribution. The noise dimensions are also uniformly distributed. In this dataset, drift is again encoded in correlation only. However, the distribution is not as simple as a normal distribution.

Based on these basic datasets we generate data streams consisting of 750 samples with drift times randomly picked between $t = 100$ and $t = 650$ by varying the following parameters:

- We vary the drift *intensity* of the previously described datasets, the default is set to 0.125.
- We modify the *number* of drift events in the given data streams, the default value is 1.
- Finally, we consider adding noisy dimensions. As described above, each stream contains two drifting dimensions; additional dimensions are noise according to the dataset description. We control the total number of *dimensions*, the default is set to 5, i.e., 3 non-drifting noise dimensions.

Methods: We make use of the following algorithms that cover every major type and sub-type of drift detector discussed in section 3: D3[3], KS, MMD, ShapeDD, KCpD[4], and DAWIDD. For D3 we used the default logistic regression as well as random forests as model. For MMD, ShapeDD, and DAWIDD we used the RBF-kernel and 2,500 bootstrap permutations.

[3]We use the implementation provided by the authors `https://github.com/ogozuacik/d3-discriminative-drift-detector-concept-drift`

[4]We use the implementation provided by [73, 74].

Except for KCpD all methods provide a drift score which we make use of for the evaluation. For KCpD use the extension presented in [56] which estimates the number of drifts (aka change points) given a predefined threshold parameter α that plays a similar role to a p-value threshold. We thus use the smallest α value as drift score.

In order to apply the two-sample (D3, KS, MMD) and block-based (DAWIDD, KCpD) algorithms, we split the stream into chunks of 150 and 250 samples with an overlap of 100 samples of consecutive chucks. We apply the drift detectors separately to each chunk. For the two-window-based approaches, we always choose the midpoint as the split point. This way, we obtain a drift score, p-value, ROC-AUC in the case of D3, for each chunk. For the meta-statistic approaches (ShapeDD) we apply the algorithms directly to the stream obtaining a drift score for each sample. To obtain the chunk-wise drift score, we take the minimum over all scores associated with samples that belong to that chunk.

Furthermore, as KCpD also allows for a precise pinpointing of the drift we also apply it to the entire stream and then make use of the same strategy as in the case of ShapeDD. Notice that this setup is not comparable to the others as it makes use of far more data and is not online. However, the pinpointing capabilities can be important in further downstream tasks.

Evaluation: For each setup we perform 500 runs on independently sampled streams. To evaluate the methods we make use of the ROC-AUC that is defined as the area under the receiver operating characteristic (ROC) curve. Given a number of class scores for binary labeled samples, the ROC-curve plots the true positive rate against the false positive rate for various decision thresholds on the class scores. This way, we get an impression on how many false positives we have to tolerate if we want a certain number of true positives. By taking the area under the curve (AUC) this information is condensed into a single number taking on values between 0 and 1 with 1 a perfect separation, 0.5 random chance. The ROC-AUC has the benefit that we do not have to specify a decision threshold, rather it provides an upper bound on the performance for every threshold. Notice that it is problematic to choose a threshold: on the one hand, it is not realistic in practice to optimize the threshold, because we usually do not have ground truth, but on the other hand, it is necessary as an inappropriate choice can render the method useless. The ROC-AUC measures how well the scores for the drifting and non-drifting cases can be separated and thus provides an upper bound on the expected performance. Furthermore, the ROC-AUC is not affected by class imbalance and thus a particularly good choice as the number of chunks with and without drift is not the same for most setups.

Drift strengths Depending on the application it might be desirable to employ a drift detector that is capable of robustly detecting very small drifts. To evaluate how well the discussed strategies capture small drifts we run experiments with increasing drifts on all described datasets. From a theoretical perspective, it is reasonable from most setups that weaker drifts are harder to detect. The sensitivity of the method often severely depends on the choice of meta-parameters. For example: If we use D3 with different models, the performance of the model has an impact on how well the derived drift detection works. Thus, if we make use of a comparably simple model like logistic regression we expect to be less sensitive to drift compared to models like k-nearest neighbor or random forests. Similar effects can be found in all drift detectors regarding the respective descriptors, i.e., the choice of the kernel has an impact on the performance of MMD, ShapeDD, and DAWIDD, or the way we choose the projection for KS, i.e., random, sparse, axis-aligned [32, 39, 42].

Our results are visualized in Fig 10. As expected, all methods improve their detection capabilities with increasing drift strengths. However, for ShapeDD we observe a stronger increase, reaching a high detection accuracy already for small drift strengths. As ShapeDD makes use of the same test as MMD this implies that the optimized split point choice often results in large improvements. We will study this effect more closely later on. Furthermore, as predicted the performance of D3 heavily depends on the combination of model and dataset. While logistic regression works better than random forest on the uniform dataset, its performance on the two other datasets is essentially random guessing. This is to be expected as linear models cannot learn the necessary non-linear decision boundaries. Furthermore, for 150 samples D3 with random forest even outperforms ShapeDD for all small intensities on the two overlapping squares dataset. This is to be expected as the drift there is

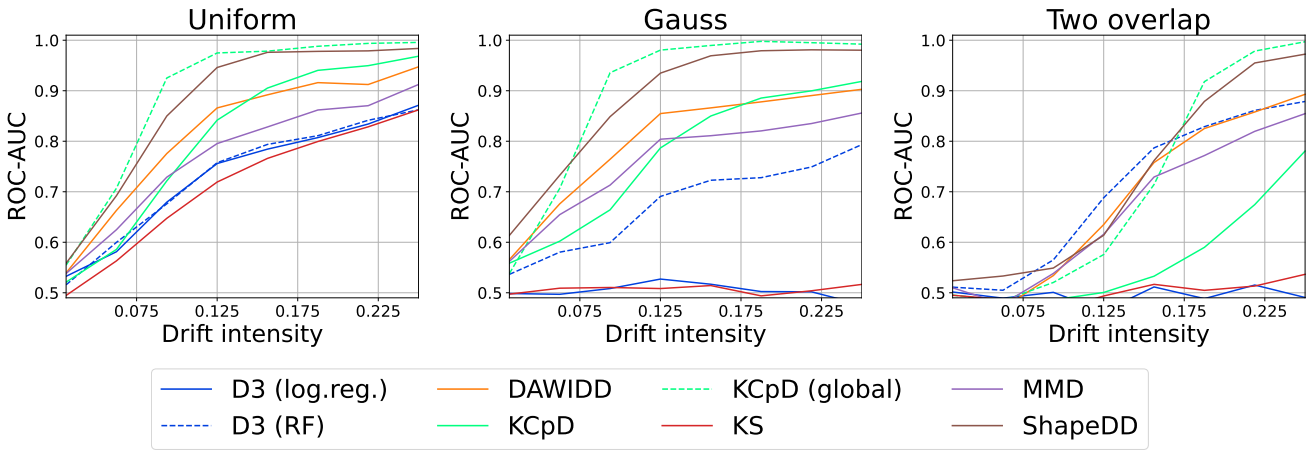


Figure 10: Drift Detection performance for various intensities.

contained in the missing corners which is hard to find using the radial basis functions of the RBF kernels, but easy for the axis-aligned decision trees. This is also an example that shows that meta-statistics can decrease the performance when the signal is too weak to be picked up by the preprocessing. Furthermore, we want to point out that DAWIDD is in second place for all datasets and the smaller parameters, and at least third place. KCpD in the batch variant requires larger intensity to outperform DAWIDD. The global variant of KCpD outperforms all online algorithms which is not surprising considering the amount of used data, however, it is usually closely matched by ShapeDD.

Thus, we suggest incorporating as much domain knowledge into the choice or construction of the descriptor as possible. Furthermore, we recommend the usage of meta-statistic or block-based methods.

Drift in correlating features In real-world applications, drift might only be detected if the correlation of the features is analyzed. To evaluate how well different detectors perform in such settings we consider feature correlations in this experiment. As the correlation is realized on the datasets and not their parameterizations, i.e. in the uniform dataset, the drift manifests itself in the individual features while for the other two, it can only be detected in their correlation, we compare the model performance across different experiments.

As can be seen in Fig. 10 (Gauss, Two overlap) KS shows a performance close to random chance. This is to be expected as in both cases the drift is entirely encoded in the feature correlation and thus lost to a feature-wise analysis as performed by KS. The axis-aligned decision trees used by D3 with random forests show a similar issue in the case of Fig. 11 (Gauss) whereas the kernel-based methods show far more comparable results across all datasets.

We thus advise only using methods that make heavy use of feature-wise analysis if drift in the correlations only is either less relevant or very unlikely. If this is not an option, ensemble-based drift detectors may provide an appropriate solution.

High dimensional data streams In many practical applications, one has to cope with high dimensional data streams containing measurements from many different sensors. In these settings, the drift might be reflected in a few features only. We will investigate, whether a high dimensionality disturbs some of the drift detection strategies. For this purpose, we again run experiments increasing the number of non-drifting features.

Our results are visualized in Fig. 11. We observe a decline in the performance of all methods with an increasing number of non-drifting features. However, the KS-based approach quickly declines to a non-acceptable level. Interestingly, D3 also suffers a comparably fast decline in performance although tree-based methods usually perform intrinsic feature selection. However, the decline is usually less steep compared to the other methods. Additionally, in case of the Gauss dataset (in Fig. 11), where the features are strongly correlated and hard to find the random forests we see a particularly steep decline. We also observe that the additional information provided to the global KCpD does not seem to help the method, which can be explained by the choice of the RBF kernel.

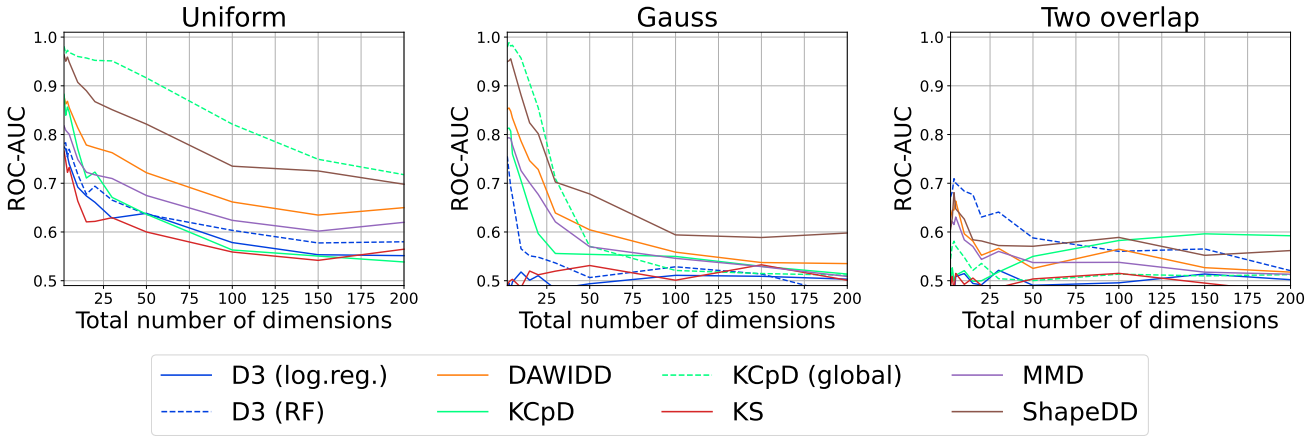


Figure 11: Drift Detection performance for various number of total dimensions.

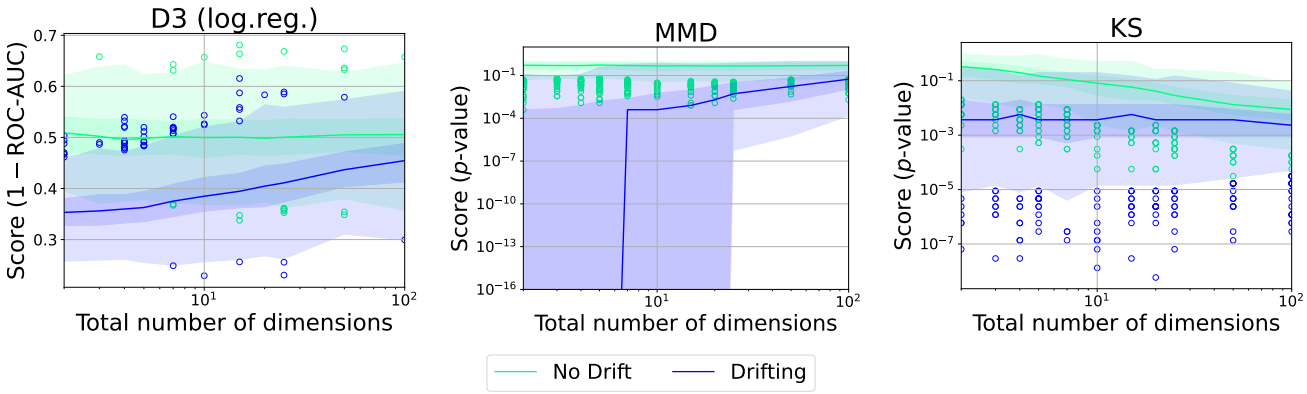


Figure 12: Effect of total number of dimensions on drift score for various drift detectors. Graphic shows median (line), 25% − 75%-quantile (inner area), min − max-quantile (outer area), and outliers (circles).

Thus, we advise to choose appropriate preprocessing techniques or descriptors in order to select or construct suited features.

We further analyzed the behavior of the different methods in a test bed setting. We used the uniform dataset only and drift right in the middle, which we also used as split point. We then compared the obtained drift scores (p-values) for drifting and non-drifting samples, each containing 250 data points. We repeated the procedure 500 times for various numbers of dimensions. The results are shown in Fig. 12.

As can be seen, for D3 and MMD detecting the drift in the drifting sample becomes harder for more dimensions, the score assigned to the non-drifting stream however stays the same. This is not too surprising as the signal-to-noise ratio becomes worse for more dimensions.

For KS it is the other way around: while the score for the drifting sample stayed the same, the score for the non-drifting sample declined, indicating that it becomes more like a drifting sample. This effect is due to the fact that just by random chance, some dimensions look like there could be drift although there is none. This effect is essentially the multi-testing problem posing a huge problem for dimension-wise drift detectors.

We thus advise that in case of high dimensions with high cost in case of false alarms, one should refuse drift detectors that operate dimension-wise.

Number of drift events Another crucial parameter is the number of drift events in a single window. Most drift detectors assume that there is only one drift event per window. While that might not be such a big issue for two-window approaches [53], meta-statistic-based algorithms need to take the resulting interference patterns into account and accommodate for them adequately. The ShapeDD for example makes the explicit assumption that there is at most one drift event in the scanner window. However, in particular, streams that alternate between two states may pose a problem to two-window-based drift detectors. For example, if we observe concepts ABA

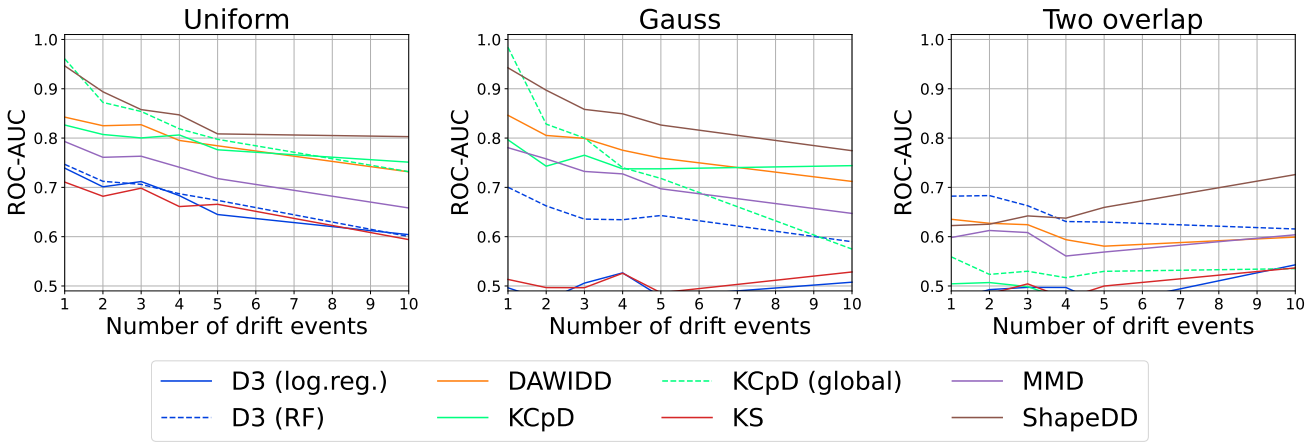


Figure 13: Drift Detection performance for various numbers of drift events.

and split right in the middle of B then the mean distribution in both windows is the same and the drift detector will fail to detect the drift. Although overlapping windows reduce the chance of such scenarios, they can still cause some problems.

As before, we use the same experimental setup, this time increasing the total number of drift events. The results are presented in Fig. 13. As can be seen, all drift detectors struggle when confronted with multiple drifts. Interestingly, in particular, the global KCpD is strongly affected by the number of drifts.

We thus advise making use of block-based drift detectors if several drift events are to be expected. In particular, we suggest not to make use of meta-statistic-based methods unless they can explicitly deal with the setup.

D3 Model choices As already pointed out in the experiment on intensity, different specific instantiations of the algorithm may have a significant impact on the detection performance. We showcase this fact in the case of D3 by considering different learning models. We made use of the same experimental setup as above and used the following models: logistic regression, random forests (RF), extra tree forests (ET) (a variant of random forests), and k-nearest neighbor classifier (k-NN). The results are presented in Fig. 14.

As can be seen the model has a major influence on the method's performance. While the k-NN-based model is outperformed by all other models on the uniform dataset, it clearly outperforms all other models on the Gauss dataset. On the other hand, similar models result in similar performance as can be seen in cases of RF and ET. What surprised us was that we were not able to observe their capability to select the most important features that is theoretically available to all models except k-NN, which still outperformed the other models in 2 out of 3 cases with high dimensionality.

To conclude, we suggest using as much prior knowledge as possible to build a suitable descriptor for the data at hand. This result matches the observations of [32] where the authors argued that the descriptor is actually more important than the metric that is derived from it.

Effect of split point Last but not least we further analyzed the effect of the used split point for the two-window approaches. We used the uniform dataset only to create two types of samples each containing 250 samples. In the first type, the drift occurs right in the middle, which is also used as the split point for the methods. For the second, the drift happens at a randomly chosen point within the sample. We compare the obtained scores (p-values) for various drift intensities, the results are shown in Fig. 15.

As can be seen, all three methods perform significantly better when the drift point is known. Also, they show drastically smaller variance. The latter is to be expected as depending on where the drift point is relative to the used split point and the window size the problem can become simpler or harder.

Considering our results, it is reasonable that approaches that preselect a candidate split point rather than simply choosing one ignorant of the underlying data provide a valid strategy to optimize performance. This

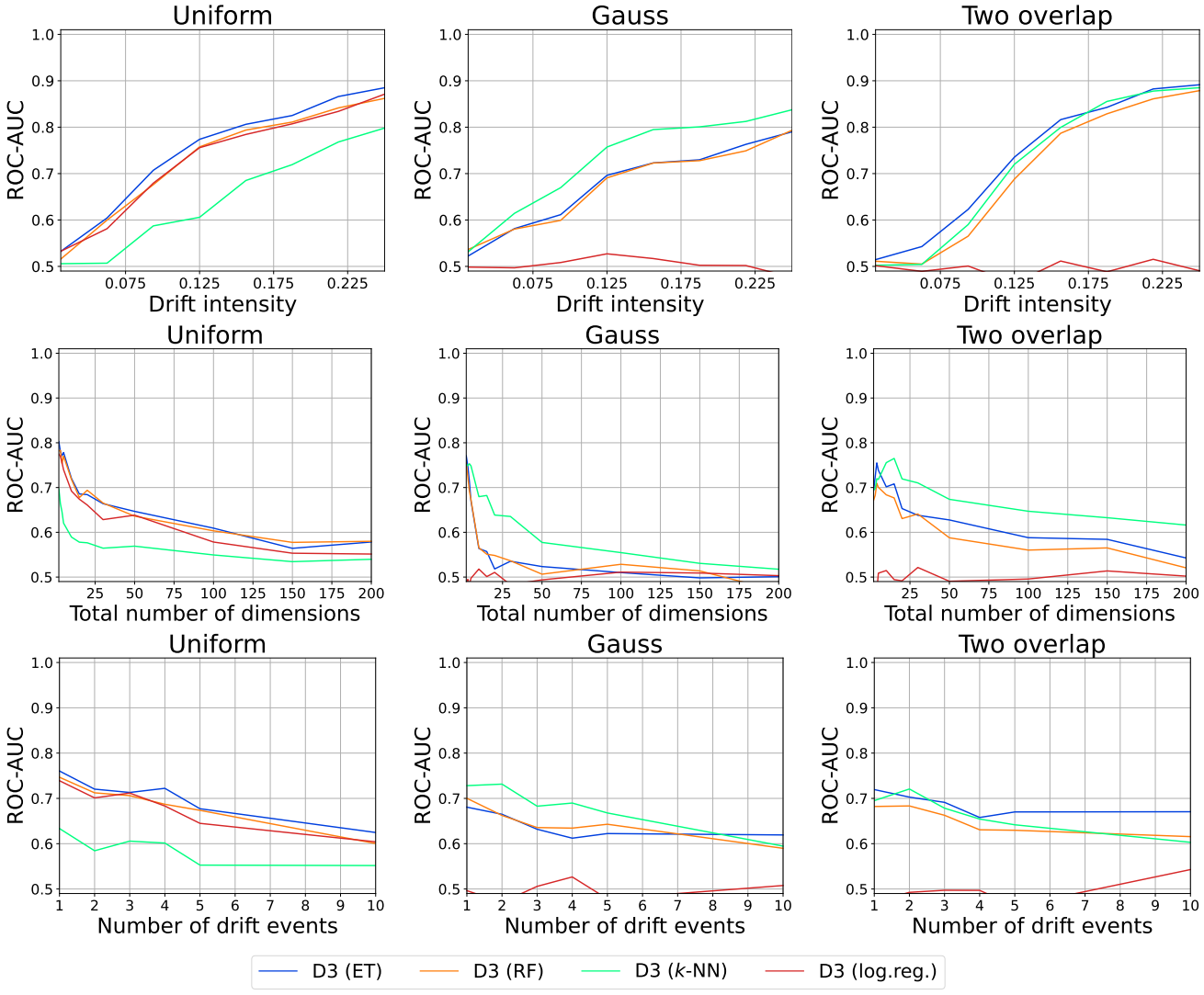


Figure 14: Drift Detection performance for various models used by D3.

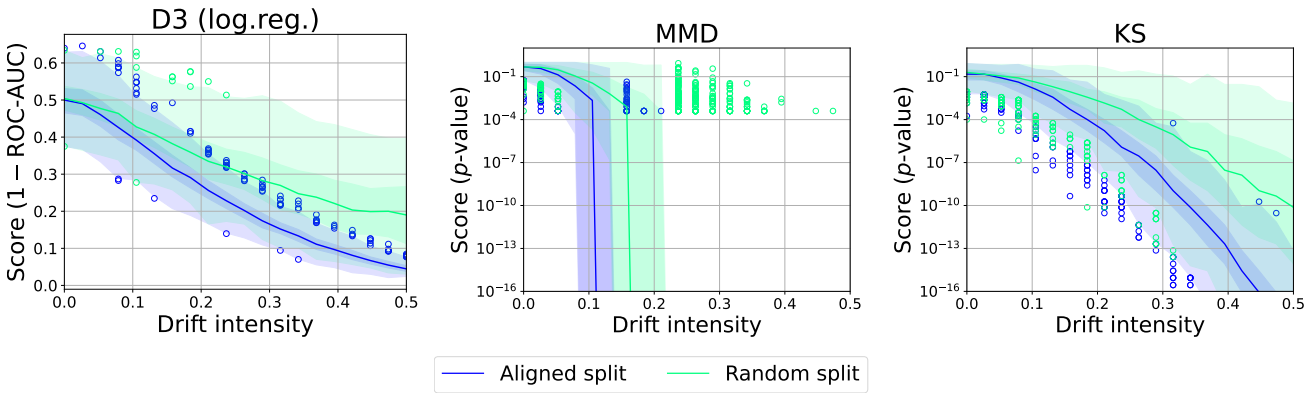


Figure 15: Effect of split point on detection performance of various drift intensities. Graphic shows median (line), 25% − 75%-quantile (inner area), min − max-quantile (outer area), and outliers (circles).

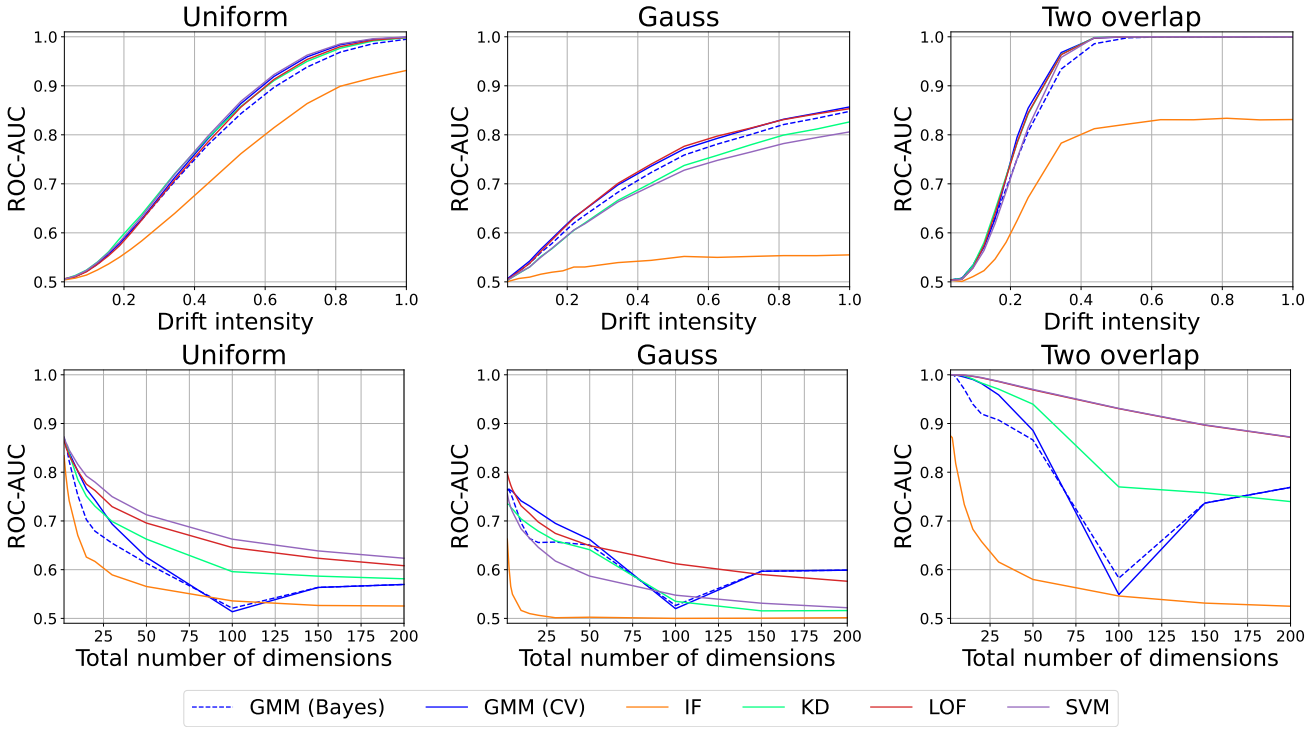


Figure 16: Drift Detection performance for various model/loss-based approaches. Experiments with varying intensity use a different scale than previous experiments.

reflects the discrepancy of performance when comparing MMD with ShapeDD, with the latter being essentially a strategy to find good candidate split points. We thus advise the user to investigate options to preselect a good candidate split point, either through prior knowledge or by choosing an appropriate algorithm.

Loss-Based approaches We also considered loss- and outlier-based approaches, namely one-class-SVM (SVM; RBF kernel), local outlier factor (LOF; $k = 10$), isolation forests (IF), and density-based approaches kernel-density (KD; RBF kernel) estimate, and Gaussian mixture model (GMM; ≤ 10 components selected via cross-validation (CV) or Dirichlet prior (Bayes)). We made use of the scikit-learn [75] standard implementation and default parameters if not specified above. In case of the outlier-based approaches we used the outlier score, in case of the density-based approaches we used sample probability. We considering the same datasets and parameters as before but used a simplified setup: We first trained the models on the first 100 samples (no drift) and then applied them to the remaining stream. As a consequence the number of drift becomes irrelevant to the detection task. The results are shown in Fig. 16.

As the stream contained data from the initial training distribution and the drifted distribution, we expect the

samples from the other distribution to be marked as outliers or assigned a small probability. However, even for an intensity of 0.25 the ROC-AUC falls below 0.7 for all methods but and two out of three datasets. Notice that the experiments using D3 clearly indicate that a significantly better performance is possible. We thus increased the default value for intensity to 0.5.

As expected we found that an increase of intensity leads to better performance. Still most methods require a comparably strong drift to achieve good performance. In particular, isolation forests seem to perform comparably poor. The number of dimensions also has a negative impact on the performance, resulting in lower scores as compared to the approaches studied in the previous sections.

We thus found additional empirical evidence for the results of [27] which challenge the suitability of loss-based approaches for drift detection from a formal mathematical perspective and therefore suggest the reader not to make use of loss-based approaches.

3.5 Conclusion and Guidelines for Drift Detection

After providing a formal definition of a drift detector, we presented and categorized many different approaches for drift detection. Table 1 and Fig. 4 provide a condensed summary of the proposed taxonomy and summarize how different methods are implemented according the the common staged scheme as visualized in Fig. 2. To wrap up our analysis on drift detection, we will summarize the main findings of our experiments with representative implementations of the discussed strategies.

A main finding is that as much domain knowledge as possible should be incorporated when designing drift detection schemes. This concerns selecting appropriate preprocessing techniques, constructing and engineering suitable features, and choosing fitting descriptors in stage 1&2 of the process. Over all experiments, we found that it is advisable to use meta or block-based methods. Next to the finding that choosing good split points is crucial for obtaining good detection capabilities, we found some very situation-specific insights: A feature-wise analysis should only be performed if it is expected that the drift does not inflict itself in correlations. Otherwise, relying on ensemble-based techniques seems to be the better solution. When working with high dimensional data, one should avoid using dimension-wise methodologies, especially if false alarms are costly in the considered application. Finally, if multiple drifts are expected, applying block-based detectors is particularly suitable. Finally, but maybe most importantly loss-based strategies should be avoided when the target of the drift detection is monitoring for anomalous behavior.

4 Drift Localization and Segmentation

Solely detecting and determining the time point of the drift is not sufficient in many monitoring settings. In order to take appropriate action, more questions concerning the drift have to be answered. In this section, we focus on the *where* – our goal is to identify the drifting components in space.

4.1 Problem Setup and Challenges

The task of determining where in data space the detected drift takes place or manifests is referred to as *drift localization*. Informally, the problem of drift localization can be expressed as "finding those regions in the data space that are affected by the drift" [9, 30]. We have illustrated this in Fig. 17, the dotted area is the area of interest. A slightly different angle on this question would be investigating "whether or not a given sample is affected by the drift" [37, 76]. Both questions are relevant in practical applications. If we know which parts of the data space are affected by the drift, we know which analysis has to be redone. On the other hand, if we know which data points are affected by the drift we can update our dataset more efficiently, i.e., we do not need to discard all old data points but only the affected ones.

Both questions can be raised interchangeably: if we know which parts of the data space are affected by drift, we can mark all data points therein as drifting. If, on the other hand, we know for every data point whether

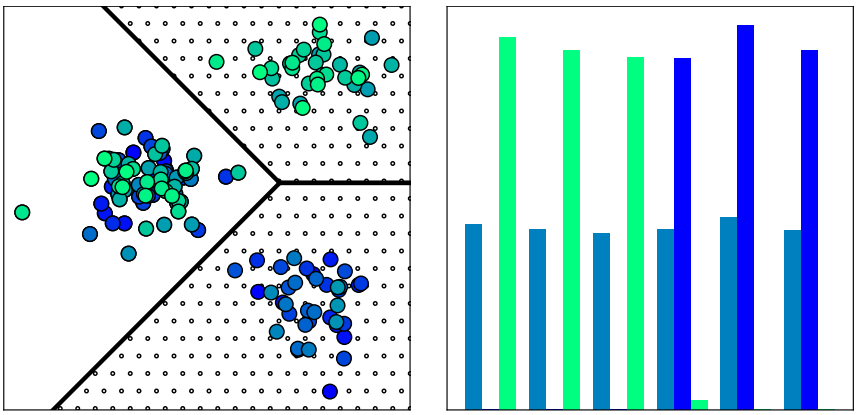


Figure 17: Visualization of distribution consisting of three segments (optimal drift segmentation; borders given by black lines). The two segments on the right form the minimal drift locus (dotted area). Left: Sample drawn from distribution color indicates time point of observation, evolving from dark blue to light green, right: Time point distribution per segment.

or not it is drifting, we can mark the corresponding parts of the data space as drifting. However, in practical applications, identifying the drifting samples is usually more feasible. In particular, we can consider it as yet another statistical test with the H_0 hypothesis "The data point x is not affected by drift".

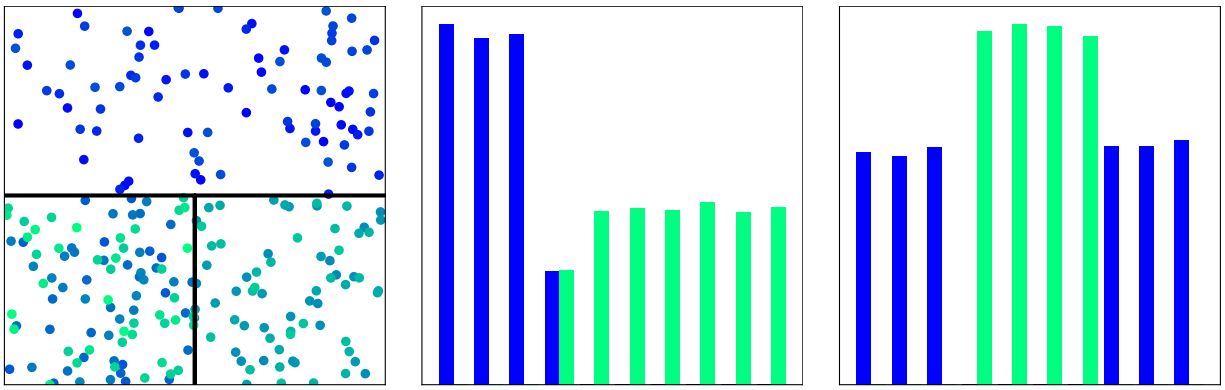
To summarize, we want to separate those parts of the dataset/space where the drift manifests from those irrelevant to the drift. However, this is challenging as the definition of drift is not local in the sense that it makes no statement about the inner workings of the drift process. Rather, it simply states that there is some kind of difference in the distributions over time. Yet, from a mathematical point of view, there is no obvious way to talk about the behavior locally, i.e., at a single point. Thus, before we can work on a solution for the task, we first need to specify what we actually mean. For this purpose, we will make use of the formalization of drift localization presented in [76].

When discussing drift in the unsupervised setup one usually imagines something like a Gaussian moving through space, i.e., $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{D}_t = \mathcal{N}(\mu_t, \sigma)$ where $\mu_\cdot : \mathcal{T} \to \mathbb{R}^d$ is the moving mean of the Gaussian. However, as is well known Gaussians span the entire space and are thus not suited for analyzing the local properties of the drift.

Instead, we suggest to approximate the drift process using a mixture model of uniform distributions on a grid, i.e., denote by $L^{(n)}_{i_1,\dots,i_d} = [i_1/2^n, (i_1+1)/2^n) \times \cdots \times [i_d/2^n, (i_d+1)/2^n)$ the grid cell starting at $(i_1, \dots, i_d)$ with length 2^{-n} and grid based approximation by $\widehat{\mathcal{D}}^{(n)}_t = \sum_{i_1,\dots,i_d \in \mathbb{Z}} \lambda_{i_1,\dots,i_d}(t) \mathcal{U}(\, L^{(n)}_{i_1,\dots,i_d}\,)$ where $\lambda^{(n)}_{i_1,\dots,i_d}(t) = \mathcal{D}_t(\, L^{(n)}_{i_1,\dots,i_d}\,)$ are the time dependent weights assigning with which probability a grid cell will be present in the data at the considered time. Notice that $\widehat{\mathcal{D}}^{(n)}_t$ is a valid approximation in the sense that it converges weakly to $\mathcal{D}_t$ as $n \to \infty$. On the other hand, $\widehat{\mathcal{D}}^{(n)}_t$ has drift if and only if at least one weight function $\lambda_{i_1,\dots,i_d}(t)$ is not constant.

Clearly, if $\mathcal{D}_t$ has no drift, then neither does $\widehat{\mathcal{D}}^{(n)}_t$. On the other hand, if $\mathcal{D}_t$ has drift and we choose n sufficiently large then $\widehat{\mathcal{D}}^{(n)}_t$ has drift, too. However, if we choose n comparably small, then small drifts are lost to us, e.g., consider $\mathcal{X} = \mathbb{R}, \mathcal{T} = [0,1]$ and $\mathcal{D}_t = \delta_{t/2^m}$ then $\widehat{\mathcal{D}}^{(n)}_t$ has drift if $n > m$.

In a sense we decompose the drift into two kinds: 1) the drift that "moves" probability between the cells which becomes apparent in a change of the weights $\lambda^{(n)}_{i_1,\dots,i_d}(t)$, and 2) the drift that is happening inside the cells $L^{(n)}_{i_1,\dots,i_d}$ that is not approximated by $\widehat{\mathcal{D}}^{(n)}_t$ and thus lost in our observation (see Fig. 18 for an example illustration). Notice that this effect is also used drift detection: virtual classifiers [64] try to find a set $L \subset \mathcal{X}$ (corresponds to the class $+1$) such that the samples in the reference window $W_-(t)$ are more likely to fall into L compared to the samples in the current window $W_+(t)$ the drift detection is then based on measuring the drift that moves the probability between L and L^C. In Fig. 18 the classifier may choose L as the upper halve and then observe the discrepancy presented in the middle diagram. An interesting case occurs when we can choose n so large that $\mathcal{D}_t$ has no more drift of the second kind, i.e., no drift inside any of the cells $L^{(n)}_{i_1,\dots,i_d}$. In this case,



Segments | Above vs. below | Below only: left vs. right

Figure 18: Visualization of distribution with three segments, all having drift. Left: optimal drift segmentation, time is color coded, black lines mark segment border, middle/right: Time point distribution of segments. First drift moves distribution from upper halve to left below, second from left below to right below, third from right below to left below. The upper halve is a drift segment (no drift inside), the lower halve is not as the distribution moves from left to right and back.

the entire information on the drift is encoded in the weights $\lambda^{(n)}_{i_1,\dots,i_d}(t)$ and even if approximating $\mathcal{D}_t$ locally by uniform distributions is no good fit the approximation errors are irrelevant for the analysis of the drift. This can be interpreted as the fact that the drifting behaviour inside $L^{(n)}_{i_1,\dots,i_d}$ is homogeneous, i.e., which particular point we consider is irrelevant as the temporal distribution is always the same.

Notice that this idea offers an algorithmic solution for example by employing an quadtree-like scheme using drift detection on each leaf as stopping criterion. However, there is no need to make use of grid cells for the approximation. The important idea is that if we can partition the data space into segments $L_1, \dots, L_n$ such that there is no drift inside each of the segments, the drift is entirely encoded in the way the weight functions $t \mapsto \mathcal{D}_t(L_i)$ deviate from constants. We will refer to sets $L \subset \mathcal{X}$ for which there is no drift inside as *drift segments*. In order to give a formal definition thereof we first need to understand what we mean by "there is no drift inside L".

Intuitively, describing the drift inside a set L we basically mean to perform some kind of spatial restriction of the drift process from $\mathcal{X}$ to L and then apply the usual definition. As pointed out in the beginning, the problem that there is no way to describe $\mathcal{D}_t$ point-wise arises. Thus, we cannot simply define the restriction to be equal to $\mathcal{D}_t$ on L and 0 everywhere else as one would do with a function. We thus need to come up with a formal definition of what restriction means. We would expect it to fulfill at least the following properties:

1. It should again be a drift process on $\mathcal{X}$.
2. It should be concentrated on the set we are restricting on.
3. It should resemble the original drift process on the set we are restriction on.

Let us construct such a process. In order to fulfill the first and third property, let us start with the drift process $(\mathcal{D}_t, P_T)$. This clearly is a drift process on $\mathcal{X}$ that resembles $(\mathcal{D}_t, P_T)$. The second property can be interpreted as: every set $A \subset \mathcal{X} \setminus L$ has probability 0. To realize this we can simply "remove" L^C from every set before taking measure, i.e., we consider $\mathcal{D}_t(A \cap L)$. This however is no longer a measure on $\mathcal{X}$ as $\mathcal{D}_t(\mathcal{X} \cap L) = \mathcal{D}_t(L) < 1$. To cope with that we normalize by $\mathcal{D}_t(L)$, i.e., we consider $\mathcal{D}_t(A \cap L)/\mathcal{D}_t(L) = \mathcal{D}_t(A \mid L)$ which is just the conditional probability given L. This is only defined if $\mathcal{D}_t(L) > 0$ which does not need to be the case for all t. We can interpret this as "at time t we do not make any observations in L" so it does not make any sense to talk about the restriction to L here. We thus also modify P_T to assign the value 0 to all such time points. This can be done in two ways: we can consider the *active time* of L, i.e., the set $\mathcal{T}_L = \{t \mid \mathcal{D}_t(L) > 0\}$, and then consider $P_T(W \mid \mathcal{T}_L)$ using similar arguments as above or we consider the probability that we observe a sample in L during W which leads to $\mathcal{D}(W \times L \mid \mathcal{T} \times L)$. Here the latter choice is more natural form a modeling perspective

as the holistic distribution of the drift process $(\mathcal{D}_t(\cdot \mid L), \mathcal{D}(\cdot \times L \mid \mathcal{T} \times L))$ is given by $\mathcal{D}(\cdot \mid \mathcal{T} \times L)$ which is well defined if $\mathcal{D}_{\mathcal{T}}(L) > 0$ and we can use the other arguments as before. Furthermore, notice that if $\mathcal{D}_t$ has a density f with respect to some measure μ, i.e., $\mathcal{D}_t(A) = \int_A f(x)\mathrm{d}\mu(x)$, then the (scaled and 0 extended) restriction of f onto L, i.e., $f_{|L}(x) = f(x)/\int_L f(x)\mathrm{d}\mu(x)$ if $x \in L$ and 0 otherwise, is exactly the density of $\mathcal{D}_t(\cdot \mid L)$ with respect to μ. In particular, this does not depend on the choice of μ.

We can now formally define the notion of drift segments:

Definition 6. Let $(\mathcal{D}_t, P_T)$ be a drift process from $\mathcal{T}$ to $\mathcal{X}$. Let $L \subset \mathcal{X}$ be a $\mathcal{D}_{\mathcal{T}}$ non-null set, then the *restriction* of $\mathcal{D}_t$ onto L is as the drift process with kernel $A \mapsto \mathcal{D}_t(A \mid L) = \mathcal{D}_t(A \cap L)/\mathcal{D}_t(L)$ and time distribution $W \mapsto \mathcal{D}(W \times L \mid \mathcal{T} \times L)$, with $\mathcal{D}$ the holistic distribution of the original drift process. We refer to $\mathcal{T}_L = \{t \in \mathcal{T} \mid \mathcal{D}_t(L) > 0\}$ is the *active time* of L.

A $\mathcal{D}_{\mathcal{T}}$ non-null set $L \subset \mathcal{X}$ is called a *drift segment* if the restriction $(\mathcal{D}_t(\cdot \mid L), \mathcal{D}(\cdot \times L \mid \mathcal{T} \times L))$ has no drift. A drift segment is called maximal iff it is maximal with respect to set inclusion, i.e., if for every $L \subset L'$ we either have $\mathcal{D}_{\mathcal{T}}(L' \setminus L) = 0$ or the restriction with respect to L' has drift.

A collection of drift segments L_i, $i \in \mathbb{N}$ that cover $\mathcal{X}$, i.e., $\cup_i L_i = \mathcal{X}$, is called a *drift segmentation*. If all segments are maximal then the segmentation is *optimal*.

The notion of a maximal drift segment comes from the observation that if L is a drift segment, then every subset $L'' \subset L$ is a drift segment, too. Thus, maximal enforces that the segments are of reasonable size.

We will now define drift localization. As drift can be arbitrarily complicated, we describe the drifting region as the complement of the non-drifting region, i.e., which part of the distribution has to be "removed" in order to make the drift disappear. As pointed out above, the term of a drift segment only encodes homogeneous drifting behaviour, i.e., there is no drift going on inside of each segment. However, there can still be drift between the segments, e.g., in Fig. 18 there are three segments, but every single point in the data space if affected by drift. If we truly want no drift, then we also have to take the drift between the segments into account. However, using all the consideration done so far this is nothing more than to state that $\mathcal{D}_t(L)$ is constant. As a consequence we have that $\mathcal{D}(W \times L \mid \mathcal{T} \times L) = P_T(W)$. This then leads to the following definition:

Definition 7. A *drift locus* is a measurable set $L \subset \mathcal{X}$ such that $(\mathcal{D}_t(\cdot \mid L^C), P_T)$ has no drift and $t \mapsto \mathcal{D}_t(L)$ is P_T-a.s. A drift locus L is *minimal* if it is contained in every other drift locus L' up to a $\mathcal{D}_{\mathcal{T}}$-null set, i.e. $\mathcal{D}_{\mathcal{T}}(L \setminus L') = 0$. We refer to the process of finding the minimal drift locus as *drift localization*.

The idea of the drift locus is that it is the complement of a segment that does not show any drift, and also no shuffling. The notion of minimality is then analogous to the maximality of the non-drifting segment. Notice that since $\mathcal{D}_t(L)$ is constant, we do not need to consider the active time $\mathcal{T}_L$ as it is either $\mathcal{T}$ or $\emptyset$ ant the latter would imply a $\mathcal{D}_{\mathcal{T}}$-null set.

As discussed in [76] the notion of a minimal drift locus has several nice properties. Among those is the fact that in all practically relevant cases, there is a unique minimal drift locus so the notion of drift localization makes sense from a theoretical point of view. Furthermore, the minimal drift locus is not empty if and only if there is drift. In the following, we will discuss how to obtain a drift localization given data.

4.2 A general scheme for drift localization

To understand the workings of drift localization algorithms, we can essentially apply the same 4-stage scheme we already used for drift detection, see Fig. 19.

Stage 1: Acquisition of data As a first step, we again need a strategy for selecting which data points are to be used for further analysis. Most approaches rely on some instantiation of sliding window strategies. Again we are free to consider sliding, fixed, or growing, as well as implicit reference windows. However, as we usually require a large amount of data for the localization task, to the best of our knowledge there are no methods that make use of an implicit reference window. Similar preprocessing steps as for drift detection, such as a deep latent space embedding, are reasonable tools that have been applied successfully in the literature [22].

Stage 2: Building a descriptor Just as drift detection, drift localization algorithms split the data processing into two steps building a descriptor from data first and then analyzing it. In contrast to drift detection, those usually offer a quite direct connection between locations in data space and the structure of the descriptor. Commonly used are decision trees [30, 76, 57] or k-neighbour based descriptors [37, 76]. However, depending on the analysis algorithm nearly arbitrary machine learning models can be used as descriptors [76].

Stage 3: Computing dissimilarity Based on the descriptor a drift score is computed. In contrast to drift detection, where the score is used to describe the global amount of drift, in drift localization it computes the amount of drift in a region of the data space [30, 76, 57] or a single data point [37, 76]. In particular, for methods that make use of region-wise computations this can be considered as performing a common drift detection that only takes a small region of the data space into account [30]. Indeed, many dissimilarities commonly used in drift detection are essentially based on the idea, that there is drift if we can partition the data space in such a way that the number of samples observed before and after the drift in a single partition differ significantly. In particular, all aforementioned drift detection algorithms make use of this idea in one way or another.

Stage 4: Normalization Similar to drift detection, the obtained dissimilarities are typically very setup-specific and do not allow a direct conclusion regarding whether or not a certain sample or region is drifting. A common strategy is to make use of some kind of bootstrap statistical test in order to either find the parameters under the H_0-hypothesis [37] or directly compute a p-value for the point or region [30, 76].

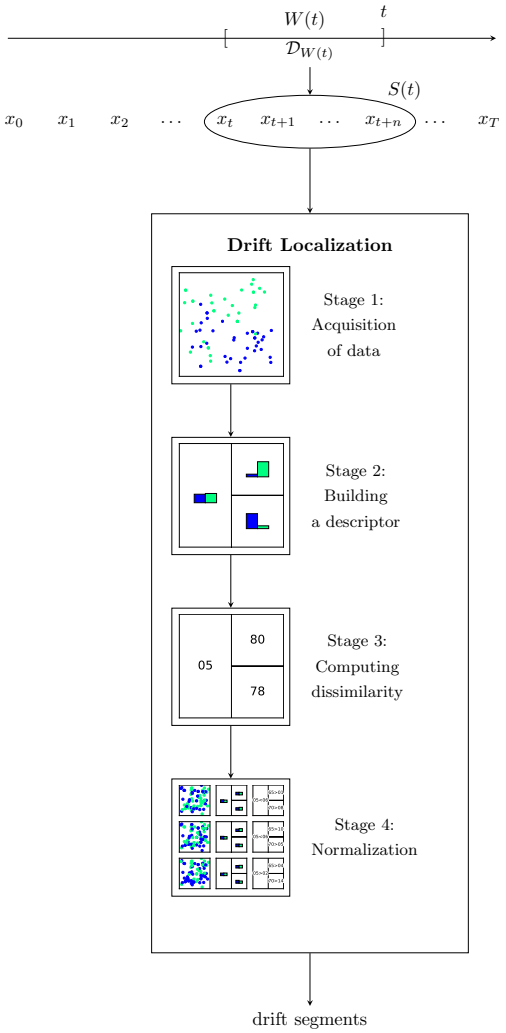


Figure 19: Visualization of drift detection from a data stream. Given a data stream, for each time window $W(t)$ a distribution $\mathcal{D}_{W(t)}$ generates a sample $S(t)$. A drift detection algorithm estimates whether the $S(t)$ contains drift or not by performing a four-stage detection scheme. Illustrated algorithm uses: tow-windows (stage 1), tree-histograms (stage 2), leaf-wise total variation norm (stage 3), and bootstrap normalization (stage 4).

4.3 Approaches

Although, there are several methods for drift detection and some that allow further analysis of the drift, drift localization in the sense described above is a less popular research question. Most methods that admit such an option either use it as a subroutine of drift detection or allow for it as a mere byproduct [30] rather than an explicit aim for drift analysis technology [9]. The majority of algorithms that allow localization are based in one way or another on performing drift detection on a local scale [30, 37, 76], i.e., instead of analyzing the entire dataset at once, only local subspaces are analyzed. This has the drawback that we usually have less data to work with. On the other hand, as we are already local in data space we can make use of far simpler detection schemes.

For example, if we make use of a grid-based binning then the total variation norm is approximated by counting the samples of

each distribution per bin, taking the difference of those numbers, and summing up:

$$\widehat{\|P-Q\|} = \sum_{i=1}^{k} \left| \frac{1}{m} \sum_{j=1}^{m} \mathbf{1}[X_j \in G_i] - \frac{1}{n} \sum_{j=1}^{n} \mathbf{1}[Y_j \in G_i] \right|,$$

where $G_1, \ldots, G_k \subset \mathcal{X}$, $\cup_{i=1}^{k} G_i = \mathcal{X}$, $G_i \cap G_j = \emptyset$ for $i \neq j$ are the grid-cells, $X_1, \ldots, X_m \sim P$, $Y_1, \ldots, Y_n \sim Q$ i.i.d. Similar estimation strategies can be applied to all sorts of distances [32].

Just as in (global) drift detection, the estimates are usually based on two-window approaches where we first select a split point and then compare the distributions before and after. As in drift detection, this split point can be chosen more or less arbitrarily. However, as the algorithms are usually less robust due to the even smaller amounts of data they are working with, it can be beneficial to first determine the actual split point using a suitable drift detector and then perform the localization. The common next step is to use the descriptor, which is constructed to work locally in data space to perform local drift detection. As pointed out above, such descriptors only need to count the number of points in the vicinity of the query point. As pointed out by [76] this can be considered as a probabilistic classification task, where the question of whether we make a statement about the local region or sample depends on the model used. This idea was further analyzed in [76] giving a theoretical justification for the most approaches available. In particular, it was shown that essentially every probabilistic classifier can be used for drift localization by analyzing its output.

In the following we will consider four exemplary approaches in more detail:

***kdq*-Tree** The algorithm is one of the oldest implementations for drift localization [30]. It is a two-window approach (stage 1), designed to work with vectorial data only. The main idea is to grow a *kd*-tree-like data structure to obtain a binning (stage 2). More precisely, the trees are obtained by iterating over each dimension in every recursion step and splitting the area right in the middle of said dimension as long as enough data is available. This assures that the volume of every leaf shrinks exponentially with each recursion. It is important to note that, up to the stopping criterion, the tree does not take the data distribution into account.

Once the tree is grown, it computes the symmetrized Kullback-Leibler divergence to compare the number of samples coming from each window on every leaf which serves as the drift score (stage 3). This way we obtain a score for every leaf and thus region in the data space. Then a scanner statistic is used to compute a threshold (stage 4) which also depends on user-defined parameters. If the score of a leaf exceeds the threshold then the leaf area is considered to be drifting.

LDD-DIS The algorithm [37] is a two-window (stage 1), neighbor-based (stage 2) approach that computes a drift score for every data point. It is based on the *Local Drift Degree* which a the ratio of the number of points in the k-neighbourhood query point. It divides the number of points in the same time window as the query point by the number of points in the other window, minus 1 (stage 3). It is thus close to 0 if the ratio is even and deviates if there are far more samples from one window than the other. By an application of the central limit theorem, the authors show that for large k the scores should follow a normal distribution if there is no drift. The parameters of this normal distribution are computed using a bootstrap permutation scheme. This distribution is then used for normalization (stage 4).

Notice that the ratio that forms the heart of the LDD is closely related to the predicted probability of a k-neighbour classifier.

Model-Based Drift Localization In [76] a family of algorithms that make explicit use of machine learning models has been introduced. The algorithms can be classified as multiple-window-based approaches (stage 1), i.e., two windows or more. For simplicity and comparability, we will consider the two-window case here.

Very similar to virtual classifiers [64, 65] and LDD a probabilistic classifier is trained to predict the window or time point each sample belongs to. This model serves as a descriptor (stage 2). The drift score, which the authors refer to as informativity, is then derived from the classifier prediction. In order to cope with class imbalance, i.e.,

different numbers of samples per window, the prediction is compared to the prediction of the constant model (stage 3), i.e., size ratio of windows, using the normalized Kullback-Leibler divergence. As pointed out by the authors, from a Bayesian point of view, informativity can be interpreted as "how much information on the time is lost, if we do not take the place into account". It then is shown by the authors that informativity takes on values between 0 and 1. The mean informativity is 0 if and only if there is no drift, and the informativity at one point is larger than 0 if and only if that point belongs to the minimal drift locus. This serves as a theoretical justification for the presented method and other methods like LDD-DIS or *kdq*-trees.

Algorithmically, the authors suggest making use of a permutation bootstrap test that uses informativity as a point-wise statistic in order to cope with effects like overfitting (stage 4). However, it was also discussed that for many commonly used models like k-nearest neighbor, decision trees, or random forests, the corresponding H_0 distribution can be computed analytically. Thus, we do not have to refer to explicit computations. Furthermore, due to the supervised training scheme, we can make use of techniques like cross-validation to automatically choose model parameters.

Drift Segmentation As stated in section 4.1 drift localization can be considered as a downstream-tasks of what is referred to as drift segmentation [57], i.e., the decomposition of the data space into regions with homogeneous drifting behavior. Assuming a drift segmentation is provided to us, we can easily check for each segment whether there is drift using a simple drift detector (stages 3 & 4). Drift segmentation can be approached using similar ideas as applied in [76]. However, instead of performing probabilistic classification, conditional density estimation is employed on a single sliding window (stages 1 & 2), i.e., the model is trained to predict $x \mapsto \mathbb{P}_{T|X=x}$. This way, the necessity of a split point or several windows is no longer given. In this sense, drift segmentation relates to block-based drift detectors.

Algorithmically, the different methods mainly differ in how they approach the construction of the descriptor (stage 2): The authors of [57] approached drift segmentation using a specially trained kind of decision tree, called the Kolmogorov tree, that uses the Kolmogorov-Smirnov test as a split criterion that reduces the statistical dependency of data and time very similar to DAWIDD. The leaves of the resulting tree then approximate the drift segments. In [22] it was pointed out that similar results can be obtained using any segmentation-based model. Also, using ideas from [31, 77] instead of special training allows a large variety of models. Furthermore, even non-segmentation-based models can be used to derive an informed metric that can then be used for clustering, the resulting clusters then approximate the segments [22].

4.4 An analysis

So far, we analyzed the discussed drift localization methodologies on a conceptual level. In the remainder of this part, we focus on conducting a more practical analysis and providing guidelines for the practical usage of these methods.

Experimental Setup *Dataset:* We consider a single, 2-dimensional, synthetic dataset that is sampled from a uniform distribution on a square. Drift is induced by a shift along the diagonal, i.e., in x- and y-direction, with different lengths (*intensity*). Notice that this corresponds to the uniform setup in the drift detection experiment. Besides drift intensity, we again also consider the total number of *dimensions* (as before), random *rotations*, and the number of *samples*. We assume that we know the time point of drift and that we are provided with an equal amount of samples before and after the drift. For the random rotation we first generate the dataset, center at 0, and then multiply with a $\lambda O + (1 - \lambda)I$ for various $\lambda \in [0, 1]$, where O is a randomly sampled orthogonal matrix (random rotation) and I is the identity matrix (data is axis aligned). We consider all combinations of parameters.

Method: We consider the *kdq*-Tree, LDD-DIS, model-based drift localization based on random forests [76]. As two out of three methods are tree-based and thus align the segments with the coordinate axis, we also consider the effect of random rotation.

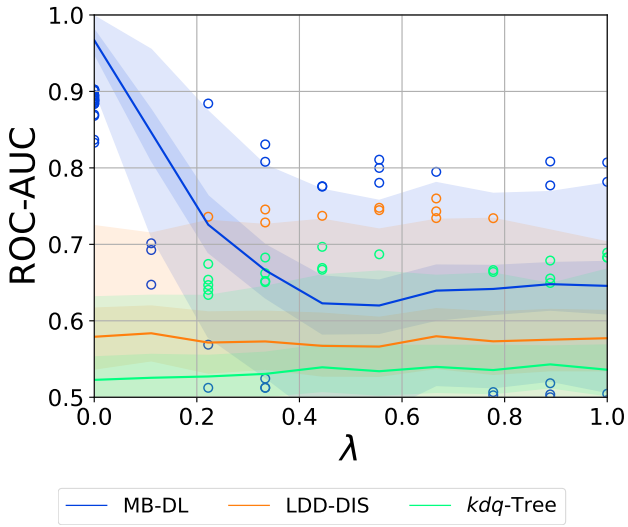


Figure 20: Effect of axis alignment (λ) on localization performance (intensity: 0.05, samples: 750, dimensions: 5). Graphic shows median (line), 25% − 75%-quantile (inner area), min − max-quantile (outer area), and outliers (circles).

Evaluation: We perform a sample-based evaluation, i.e., we want to predict for each data point whether or not it is affected by drift. Here, we consider every sample in the overlap of the square as non-drifting and every other data point as drifting. To evaluate the methods we again make use of the ROC-AUC for the same reasons. In particular, in contrast to many other scores like F1 or accuracy, the ROC-AUC is not affected by the expected class imbalance.

Overall results The overall results of the experiment (see Fig. 20-22) show that the problem of drift localization is a comparably hard one and still requires additional research. The overall ranking of the methods in our simplified experiments places the MB-DL approach at the top, followed by LDD-DIS and *kdq*-Trees at the last place. This is consistent with the findings in the original paper where more complex datasets were analyzed[76]. In nearly all parameters *kdq*-Trees perform only slightly better than random, LDD-DIS barely ever reaches a score 0.6 or higher. This, together with the very high variance makes the analysis comparably hard.

Axis-alignment As one can see in Fig. 20, axis alignment is one of the most crucial parameters for MB-DL, for the other two approaches it is nearly irrelevant. For the MB-DL applied to a window of 150 samples or more, we observe an extreme decline in performance when we switch $\lambda = 0$ (perfectly axis-aligned) to 0.5, after that, the performance stays at a constant, low level. This is to be expected as random forests use axis-aligned splits and thus can hardly cope with classifications that require making use of correlations as is the case if $\lambda > 0$. In the following, we will thus explicitly discuss the cases $\lambda = 0$ and $\lambda = 1$ separately.

We thus suggest applying some kind of preprocessing or other model in case of drift that is mainly present in the correlation.

Sample size and dimensions As is expected all methods profit from more samples (see Fig. 21). However, the increase in performance of *kdq*-Tree is not significant and might be due to random chance. In the case of MB-DL the increase for $\lambda = 1$ is only moderate and comparable to LDD-DIS, for $\lambda = 0$ the increase of performance is very strong and nearly linear until it reaches nearly perfect classification.

Similar results can be found for the number of noise dimensions. While all methods suffer from high dimensionality, MB-DL does rather moderately in case $\lambda = 1$, this is to be expected as the detection scheme is trained in a supervised fashion and thus can perform feature selection in this case. Similar effects cannot be observed for *kdq*-Trees as they do not optimize the tree structure for the problem at hand, LDD-DIS which is not capable of feature selection in the first place, or MB-DL if $\lambda = 0$ as in this case feature selection is not possible.

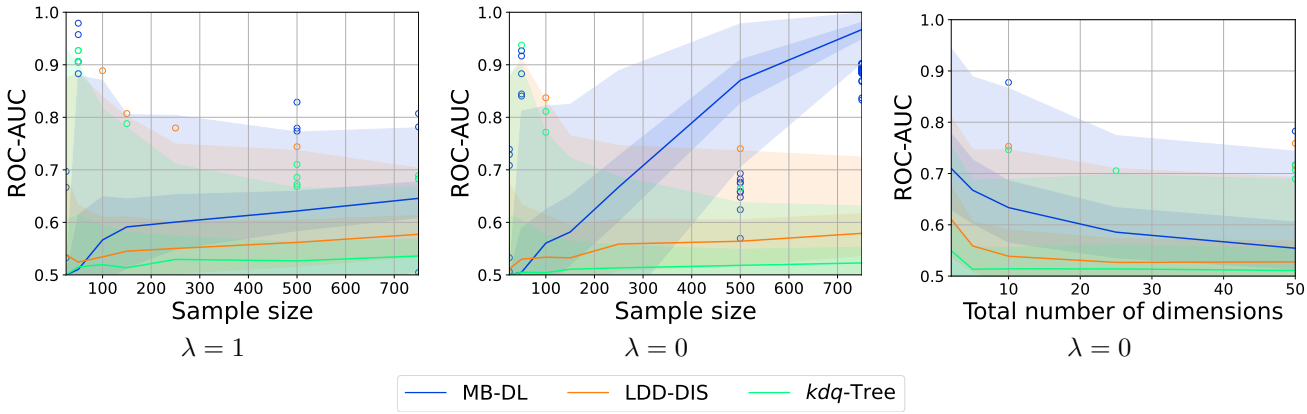


Figure 21: Effect of number of samples and dimensionality on localization performance for various choices of λ (intensity: 0.05, samples: 250, dimensions: 5). Graphic shows median (line), 25% – 75%-quantile (inner area), min – max-quantile (outer area), and outliers (circles).

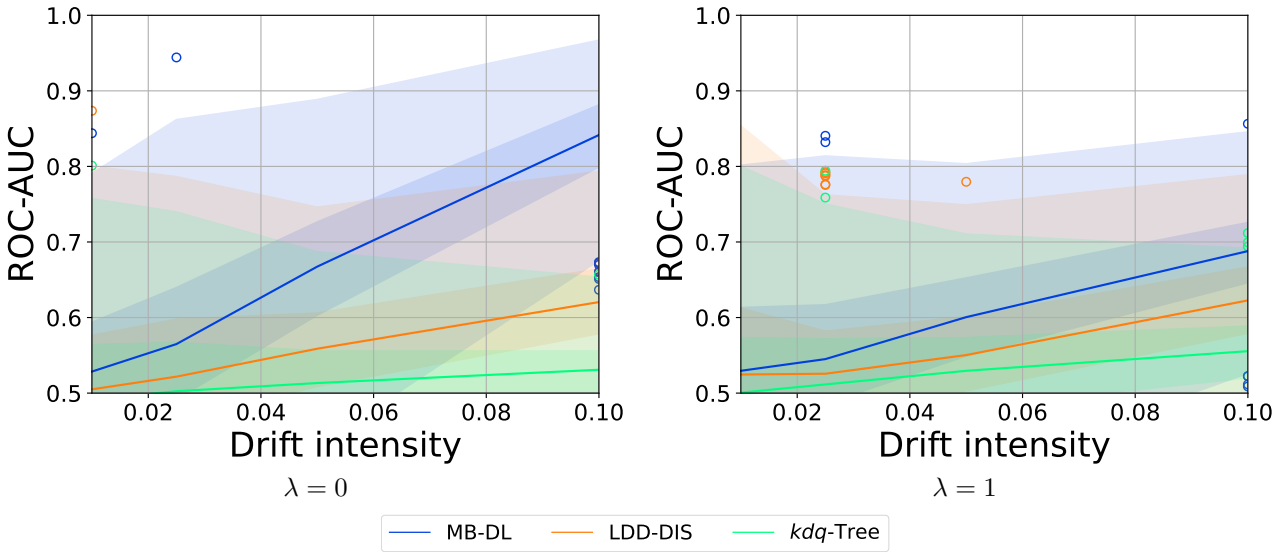


Figure 22: Effect of drift intensity on localization performance for various choices of λ (samples: 250, dimensions: 5). Graphic shows median (line), 25% – 75%-quantile (inner area), min – max-quantile (outer area), and outliers (circles).

To conclude, drift localization requires a comparably large amount of data and high dimensionality poses a problem in particular if no feature selection is possible. Still, even in this case model-based approaches might still be the best choice.

Drift intensity Just as in drift detection, the larger the drift the easier the task becomes. This finding is systematic across consistent across all methods as summarized in Fig. 22. Furthermore, we see a nearly linear relationship between the drift intensity and the increase of performance. This is true for both cases $\lambda = 0$ and $\lambda = 1$, but for the latter the increase is much steeper.

In particular, we suggest making use of a split point selection by using an appropriate drift detector like ShapeDD in order to increase localization performance. This seems to be the best way to increase performance in a general practical setup where other preprocessing or model choices are not available.

4.5 Conclusion and Guidelines

Investigating the task of drift localization, we again provided a formal definition of the task and classified existing approaches into the presented four-staged framework. Overall, one can say that research on this task is still very

limited with few approaches existing. In our experiments we only reported good results for few of them. Thus, further research is needed.

When applying drift localization methods in practical applications one should consider the following key findings of our experimental evaluation and analysis: First, of all one should use as much data as possible for the localization task. Besides, avoiding high dimensional data where feasible is desirable. To obtain good results it is crucial to select suitable drift points, e.g., by choosing reliable drift detection methods as discussed in the previous part. Finally, when reyling on tree-based methods, it is crucial to design an appropriate preprocessing if the drift inflicts itself in data correlations.

5 Drift Explanations

So far, working on the *whether*, *when*, and *where*, we focused on the problems of drift detection and localization. However, the knowledge which samples are affected by the drift at which time is still not sufficient in many real-world monitoring settings. Therefore, we will focus on a more detailed analysis of the drift in this section.

5.1 Problem and Setup

The question of *explaining* drift, i.e., describing the potentially complex and high dimensional change of distribution in a human-understandable fashion, is a relevant problem as it enables an inspection of the most prominent characteristics of how and where drift manifests itself. Hence, it enables human understanding of the change and thus is a key ingredient for the informed decision-making of human operators.

In contrast to the problems discussed so far, drift explanations are an inherently ill-defined problem. This is partly caused by the fact the term "explaining" is inherently ill-posed as can be seen by considering the wide range of different explanation schemes, methods, and frameworks present in the literature. Furthermore, the choice of a suitable explanation is highly domain and problem-specific.

Providing a detailed, complete, and understandable description of ongoing drift, requires a large amount of information covering all relevant aspects. This usually surpasses the level of information which is required to select change points or to estimate the rate of change: While drift can be detected based on a single drifting feature, its explanation might need to address the interplay of all drifting features.

This leads us to two general insights on the topic: First, it is not possible to provide a formal restriction let alone a definition of what drift explanations are. Second, a large number of different explanation schemes – one for every potential use case – is actually a desirable state of affairs.

5.2 A General Scheme for Drift Explanations

Even though we cannot provide a formal definition of drift explanations, we can still analyze approaches using a similar functional scheme as we did for detection and localization. Usually, the normalization stage is not required.

Stage 1: Acquisition of data: As a first step, we again need a strategy for selecting which data points are to be used for further analysis. Most approaches rely on some instantiation of sliding window strategies. Again we are free to consider sliding, fixed, or growing, as well as implicit reference windows. Similar preprocessing steps to drift detection and localization, such as a deep latent space embedding, are reasonable tools that have been applied successfully in the literature [22].

Stage 2: Building a descriptor: Just as drift detection and localization, drift explanation algorithms split the data processing into two steps building a descriptor from data first and then analyzing it. Similar to drift localization, those are usually chosen with respect to the explanation task at hand. Depending on the desired explanation, a large variety of descriptors is used, but binning approaches are very common [41, 78, 79]. However, as pointed out by [22] nearly every machine learning model can be used as descriptors.

Stage 3: Computing explanations: In the last of the explanation scheme, the descriptor is analyzed. This is indeed comparable to the computation of dissimilarity as a simple quantity derived from the descriptor. Indeed, many methods simply derive such numbers such as feature-wise change intensity or change in correlation [41, 78, 80, 79]. Here, a further analysis by means of normalization is not necessary as the data is usually directly presented to and judged by a human operator. However, also some more advanced explanation methods are available [22].

5.3 Exemplary Cases

While explainability has been a major research interest in recent years [24, 81], more complex explanation methods for drift are still limited. Quite a number of approaches aim for the detection and quantification of drift [9, 78], or its visualization [80, 40, 79]. Furthermore, several methods focus on feature-wise representations of drift [80, 78, 40, 79]. However, these methods face challenges if high-dimensional data or non-semantic features are dealt with. To our knowledge, there is only one approach that directly targets concept drift using more complex XAI methods for explaining drift [22].

In the following, we will group the methods based on the question of whether they focus on feature-wise analysis only, or allow for the application of more complex XAI technologies.

Feature-Based Drift Explanations In [78, 40] the authors make use of the (conditional) *drift magnitude* to visualize the intensity and change of correlation of certain features. For sets of features F, F' the drift magnitude is defined as

$$\sigma^{F}_{\mathcal{D}_\cdot,l}(s,t) = \|\mathcal{D}_{W_l(s)}(X_F) - \mathcal{D}_{W_l(t)}(X_F)\|_{\mathrm{TV}}$$
$$\sigma^{F|F'}_{\mathcal{D}_\cdot,l}(s,t) = \int \|\mathcal{D}_{W_l(s)}(X_F \mid X_{F'}) - \mathcal{D}_{W_l(t)}(X_F \mid X_{F'})\|_{\mathrm{TV}} \mathrm{d}\mathcal{D}_{W_l(s)\cup W_l(t)}(X_{F'})$$

where $W_l(t) = (t - l/2, t + l/2)$ is the window around t with length l, $\mathcal{D}_W(X_F)$ is the projection of the drift process onto the features F, and $\mathcal{D}_W(X_F \mid X_{F'})$ is the conditioning of the conditioning of $\mathcal{D}_W(X_F, X_{F'})$ on $\mathcal{D}_W(X_{F'})$. The theoretical properties of the drift magnitude was analyzed in [53]. Notice that the drift magnitude on consecutive windows also forms the basis for the Shape Drift Detector.

To estimate the drift magnitude, [78, 40] use sliding windows (stage 1), and bin histograms (stage 2) which are used to compute the total variation norm for different time-points (stage 3). The results of this computation are directly presented to the user.

ConceptExplorer is a tool presented in [80]. It is designed for visual inspection of drift in particular in time-series data. The tool contains several analysis and visualization tools: drift detection algorithm and event-log-plot, automatic extraction of concepts, visualization, and interaction, feature selection and relevance tools, and cross-data source analysis. For drift detection and feature analysis, standard tools are used. The concept analysis is mainly performed by making use of a time-binned correlation matrix.

In [79] the authors suggest using *brushed, parallel histograms* in order to visualize concept drift. The data distribution for each dimension is displayed using a histogram, correlations are marked by lines connecting the dimension-wise projections. The implementation presented by the authors enables user interaction by allowing the user to select subsets of points, e.g., parts of the histograms, for which more information is desired.

To visualize drift, the authors use sliding windows (stage 1) for which the representation is computed and presented side-by-side (stages 2 & 3).

Model-Based Drift Explanations The notion of model-based drift explanations was coined in [82, 22]. Simply put, the fundamental idea is that drift explanations are supposed to tell us why a drift detector raised an alarm. As stated in section 3 there are several approaches that explicitly make use of machine learning models as a descriptor to detect the drift. In these cases, explaining why the model used by the drift detector obtained its results also provides us with information on why the drift detector raised an alarm or not.

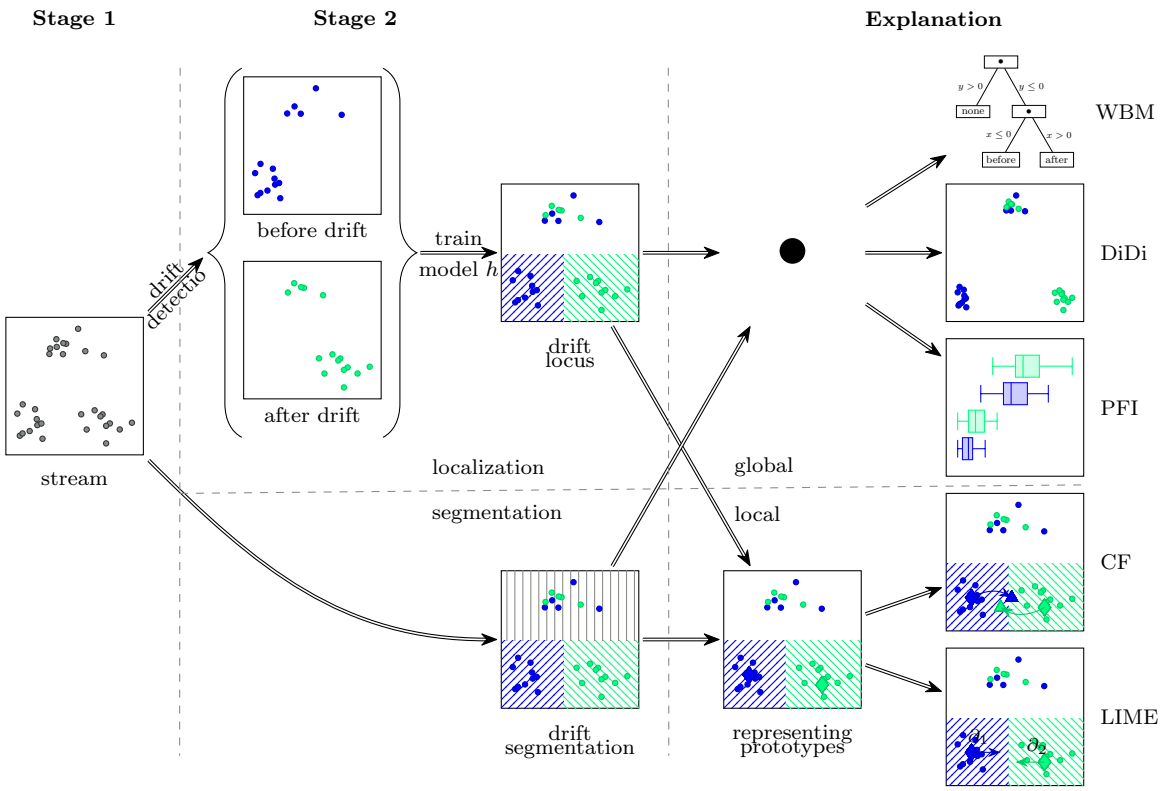


Figure 23: Model-based drift explanation strategy: In the first step a model for drift localization or drift segmentation. Afterward, the obtained model is explained by means of an explanation method.

In order to obtain sufficiently informative explanations we however require certain kinds of models or more precisely training schemes. As discussed before, for drift detection frequently detecting a single drifting feature is sufficient. Since this might be not sufficient to provide a valuable explanation, we instead rely on drift localization or segmentation. [22] pointed out that these training schemes are suitable for explaining the change. Besides, they are applicable to a wide range of models.

Once a model is trained in an appropriate way we can obtain knowledge on the drift by analyzing it. This can be understood as follows: if the model contains all the information regarding the drift, we can analyze it as a proxy for the drift. This again fits into our staged scheme where the model serves as the descriptor: As visualized in Fig. 23, first, data is acquired and a model describing the data is trained (stage 1&2). In the second step, the obtained model is explained by a suitable explanation approach.

There are several ways the explanation can be done. This is another strength of this approach: not only can we choose the model that fits our data best, but also the explanation that fits our problem best. In the original paper the authors considered the non-exhaustive list of the following explanation methods [22]:

- Linear models or decision trees belong to the class of *interpretable models* which have the nice properties of being naturally understandable by humans [24, 83]. Yet, they usually suffer problems due to low complexity.

- More complex explanations are provided by *discriminative dimensionality reduction* which provides a global overview of the model behavior using model-enriched dimensionality reduction techniques [84, 85, 86].

- *Global feature importance and relevance* techniques like permutation feature importance, feature importance, feature relevance, and Shapley-values offer feature-wise explanations based on the model scheme [87, 88, 89]. In contrast to other methods, those usually come with formal descriptions and guarantees on what they can and cannot do. Such have been shown to be useful for various setups with semantic features, in particular sensor networks [22].

- *Local feature importance* techniques like Saliency Maps [90] or Local Interpretable Model-agnostic Explanations (LIME) [91] offer feature-wise analysis on a single instance basis. This can provide more information on the single instance and offer insight into the change of correlations, however, it also requires finding samples that are relevant enough to provide additional insight if analyzing. In [22] a technique for finding such samples was provided.

- *Contrasting explanations and counterfactuals* offer explanations in terms of contrasting sample pairs [24, 92, 93]. In contrast to local feature-wise explanations, those do not only show which features are affected but also how they are affected. Thus, the user is directly confronted with the effect of the drift in exemplary cases. Drawbacks of this approach are that it only works well with about drift, is computationally expensive, and there are usually no guarantees that valid explanations are found in practice.

A further advantage of model-based explanations is that the connection of drift-related problems like drift detection and localization to explanation and analysis techniques can also be used to increase the performance of the other tasks. For example, in [42] this connection can be used not only to transfer ideas of feature relevance theory in order to obtain drift explanations but also to perform feature selection for drift detection which resulted in significant increases in accuracy.

There did exist works prior to [22] that made use of a similar scheme. However, those approaches are hand-tailored for a specific setup rather than a general framework. In [93] a combination of an autoencoder and a distance-based outlier detection in the latent space is used for drift detection. Drift explanations are provided by counterfactuals of the outlier detector. In [86] drift is detected using a Gaussian mixture model in a loss-based fashion. Then, the authors use a discriminative version of t-SNE to create an embedding.

5.4 Conclusion and Guidelines

Focusing on drift explanations, we identified another research gap, as much of the work in this area is still very basic. Much more work is needed to provide user-friendly explanations across different domains and settings. Additionally, evaluations in the form of user studies will be required to evaluate future approaches. Regarding the discussed methodologies, model-based explanations seem the most promising as the framework is very flexible combining model-based localization and segmentation methods with a range of established explanation schemes. The latter can be chosen to fit the real-world scenario that needs to be targeted.

6 Conclusion

In this work, we provided definitions and categorizations of drift detection and drift localization in an unsupervised setting. Furthermore, we categorized state-of-the-art approaches and analyzed them based on a four-staged general scheme we proposed. In addition, we briefly considered drift explanations and showcased some works targeting this task.

Next to providing an overview of existing work, we analyzed the different underlying strategies to contribute guidelines on how to choose methodologies based on the attributes of the setup and the expected drift mechanism. Finally, we found that more research is required, in particular focusing on the localization and explanation tasks.

References

[1] Gregory Ditzler et al. "Learning in Nonstationary Environments: A Survey". In: *IEEE Computational Intelligence Magazine* 10.4 (Nov. 2015), pp. 12–25. ISSN: 1556-603X. DOI: 10.1109/MCI.2015.2471196. (Visited on 07/21/2023).

[2] Stelios G. Vrachimis et al. "Battle of the Leakage Detection and Isolation Methods". In: *Journal of Water Resources Planning and Management* 148.12 (Dec. 2022). ISSN: 1943-5452. DOI: 10.1061/(ASCE)WR.1943-5452.0001601. (Visited on 10/06/2022).

[3] Hongge Chen and Duane Boning. "Online and incremental machine learning approaches for IC yield improvement". In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Irvine, CA: IEEE, Nov. 2017, pp. 786–793. ISBN: 9781538630938. DOI: 10.1109/ICCAD.2017.8203857. (Visited on 09/18/2023).

[4] Hossam A. Gabbar et al. "Incremental Learning-Based Algorithm for Anomaly Detection Using Computed Tomography Data". In: *Computation* 11.7 (July 2023), p. 139. ISSN: 2079-3197. DOI: 10.3390/computation11070139. (Visited on 09/18/2023).

[5] Vasso Reppa, Marios M. Polycarpou, and Christos G. Panayiotou. *Sensor fault diagnosis*. Foundations and trends in systems and control 3, 1-2. Boston Delft Hanover: now publishers, 2016. ISBN: 9781680831283.

[6] João Gama et al. "A survey on concept drift adaptation". In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37.

[7] Paulo M Gonçalves Jr et al. "A comparative study on concept drift detectors". In: *Expert Systems with Applications* 41.18 (2014), pp. 8144–8156.

[8] João Gama et al. "Learning with Drift Detection". In: *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004, Proceedings*. 2004, pp. 286–295.

[9] Jie Lu et al. "Learning under Concept Drift: A Review". In: *IEEE Transactions on Knowledge and Data Engineering* (2018), pp. 1–1. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2018.2876857. (Visited on 09/11/2023).

[10] Matthias Delange et al. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2021.3057446.

[11] Samaneh Aminikhanghahi and Diane J Cook. "A survey of methods for time series change point detection". In: *Knowledge and information systems* 51.2 (2017), pp. 339–367.

[12] Philippe Esling and Carlos Agon. "Time-series data mining". In: *ACM Computing Surveys* 45.1 (Nov. 2012), pp. 1–34. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2379776.2379788. (Visited on 09/11/2023).

[13] Chen Zhang et al. "A survey on federated learning". In: *Knowledge-Based Systems* 216 (Mar. 2021), p. 106775. ISSN: 09507051. DOI: 10.1016/j.knosys.2021.106775. (Visited on 04/06/2023).

[14] Yang Liu et al. "A Secure Federated Transfer Learning Framework". In: *IEEE Intelligent Systems* 35.4 (July 2020), pp. 70–82. ISSN: 1541-1672, 1941-1294. DOI: 10.1109/MIS.2020.2988525.

[15] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010), pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191. (Visited on 06/15/2023).

[16] Viktor Losing, Barbara Hammer, and Heiko Wersing. "Incremental on-line learning: A review and comparison of state of the art algorithms". In: *Neurocomputing* 275 (2018), pp. 1261–1274. DOI: 10.1016/j.neucom.2017.06.084. URL: https://doi.org/10.1016/j.neucom.2017.06.084.

[17] Jacob Montiel et al. "Scikit-Multiflow: A Multi-output Streaming Framework". In: *Journal of Machine Learning Research* 19.72 (2018), pp. 1–5. URL: http://jmlr.org/papers/v19/18-251.html.

[18] Jacob Montiel et al. "River: machine learning for streaming data in python". In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 4945–4952.

[19] Rosana Noronha Gemaque et al. "An overview of unsupervised drift detection methods". In: *WIREs Data Mining and Knowledge Discovery* 10.6 (2020), e1381. DOI: https://doi.org/10.1002/widm.1381. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1381. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1381.

[20] Igor Goldenberg and Geoffrey I. Webb. "Survey of distance measures for quantifying concept drift and shift in numeric data". In: *Knowl. Inf. Syst.* 60.2 (2019), pp. 591–615. DOI: 10.1007/s10115-018-1257-z. URL: https://doi.org/10.1007/s10115-018-1257-z.

[21] Shikha Verma. "Machine Learning for Streaming Data: Overview, Applications and Challenges". In: *Applied Advanced Analytics*. Ed. by Arnab Kumar Laha. Singapore: Springer Singapore, 2021, pp. 1–9. ISBN: 9789813366558 9789813366565. DOI: 10.1007/978-981-33-6656-5_1. URL: https://link.springer.com/10.1007/978-981-33-6656-5_1 (visited on 09/18/2023).

[22] Fabian Hinder et al. "Model Based Explanations of Concept Drift". In: *arXiv preprint arXiv:2303.09331* (2023).

[23] David Gunning et al. "XAI—Explainable artificial intelligence". In: *Science robotics* 4.37 (2019), eaay7120.

[24] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2020.

[25] Fabian Hinder, André Artelt, and Barbara Hammer. "Towards non-parametric drift detection via dynamic adapting window independence drift detection (dawidd)". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4249–4259.

[26] Fabian Hinder et al. "On the Change of Decision Boundary and Loss in Learning with Concept Drift". In: *International Symposium on Intelligent Data Analysis*. Springer. 2023, pp. 182–194.

[27] Fabian Hinder et al. "On the Hardness and Necessity of Supervised Concept Drift Detection". In: (2023).

[28] Valerie Vaquet et al. "Online learning on non-stationary data streams for image recognition using deep embeddings". In: *IEEE symposium series on computational intelligence, SSCI 2021, orlando, FL, USA, december 5-7, 2021*. IEEE, 2021, pp. 1–7. DOI: 10.1109/SSCI50451.2021.9659903.

[29] Gregory Ditzler and Robi Polikar. "Hellinger distance based drift detection for nonstationary environments". In: *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments, CIDUE 2011, Paris, France, April 13, 2011*. 2011, pp. 41–48. DOI: 10.1109/CIDUE.2011.5948491. URL: https://doi.org/10.1109/CIDUE.2011.5948491.

[30] T. Dasu et al. "An Information-Theoretic Approach to Detecting Changes in MultiDimensional Data Streams". In: *Interfaces* (Jan. 2006).

[31] Fabian Hinder et al. "Fast non-parametric conditional density estimation using moment trees". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021, pp. 1–7.

[32] Fabian Hinder, Valerie Vaquet, and Barbara Hammer. "Suitability of different metric choices for concept drift detection". In: *International Symposium on Intelligent Data Analysis*. Springer. 2022, pp. 157–170.

[33] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[34] Cynthia Dwork. "Differential privacy". In: *International colloquium on automata, languages, and programming*. Springer. 2006, pp. 1–12.

[35] Niv Haim et al. "Reconstructing training data from trained neural networks". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 22911–22924.

[36] Fernando Pérez-Cruz. "Estimation of Information Theoretic Measures for Continuous Random Variables". In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller et al. Vol. 21. Curran Associates, Inc., 2009.

[37] Anjin Liu et al. "Regional concept drift detection and density synchronized drift adaptation". In: *IJCAI International Joint Conference on Artificial Intelligence*. 2017.

[38] Arthur Gretton et al. “A Kernel Method for the Two-Sample-Problem”. In: *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. 2006, pp. 513–520. URL: http://papers.nips.cc/paper/3110-a-kernel-method-for-the-two-sample-problem.

[39] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. “Failing loudly: An empirical study of methods for detecting dataset shift”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[40] Geoffrey I Webb et al. “Understanding concept drift”. In: *arXiv preprint arXiv:1704.00362* (2017).

[41] Geoffrey I Webb et al. “Characterizing concept drift”. In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994.

[42] Fabian Hinder and Barbara Hammer. “Feature Selection for Concept Drift Detection.” In: *ESANN*. Ed. by Michel Verleysen. 2023. URL: https://pub.uni-bielefeld.de/record/2982830.

[43] Osman Salem, Farid Naït-Abdesselam, and Ahmed Mehaoua. “Anomaly detection in network traffic using Jensen-Shannon divergence”. In: *2012 IEEE International Conference on Communications (ICC)*. IEEE. 2012, pp. 5200–5204.

[44] Di Zhao and Yun Sing Koh. “Feature drift detection in evolving data streams”. In: *Database and Expert Systems Applications: 31st International Conference, DEXA 2020, Bratislava, Slovakia, September 14–17, 2020, Proceedings, Part II 31*. Springer. 2020, pp. 335–349.

[45] Rodrigo F de Mello et al. “On learning guarantees to unsupervised concept drift detection on data streams”. In: *Expert Systems with Applications* 117 (2019), pp. 90–102.

[46] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. “Reactive soft prototype computing for concept drift streams”. In: *Neurocomputing* 416 (2020), pp. 340–351.

[47] Arthur Gretton et al. “A Kernel Statistical Test of Independence”. In: *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*. 2007, pp. 585–592. URL: http://papers.nips.cc/paper/3201-a-kernel-statistical-test-of-independence.

[48] Ömer Gözüaçık et al. “Unsupervised concept drift detection with a discriminative classifier”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 2365–2368.

[49] Denis Moreira Dos Reis et al. “Fast unsupervised online drift detection using incremental kolmogorov-smirnov test”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1545–1554.

[50] Abdulhakim A Qahtan et al. “A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 935–944.

[51] Li Bu, Cesare Alippi, and Dongbin Zhao. “A pdf-free change detection test based on density difference estimation”. In: *IEEE transactions on neural networks and learning systems* 29.2 (2016), pp. 324–334.

[52] Li Bu, Dongbin Zhao, and Cesare Alippi. “An incremental change detection test based on density difference estimation”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.10 (2017), pp. 2714–2726.

[53] Fabian Hinder et al. “A shape-based method for concept drift detection and signal denoising”. In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021, pp. 01–08.

[54] Fabienne Comte and Yves Rozenholc. “A new algorithm for fixed design regression and denoising”. In: *Annals of the Institute of Statistical Mathematics* 56 (2004), pp. 449–473.

[55] Zaid Harchaoui, Eric Moulines, and Francis Bach. “Kernel change-point analysis”. In: *Advances in neural information processing systems* 21 (2008).

[56] Sylvain Arlot, Alain Celisse, and Zaid Harchaoui. "A kernel multiple change-point algorithm via model selection". In: *Journal of machine learning research* 20.162 (2019).

[57] Fabian Hinder, Barbara Hammer, and M Verleysen. "Concept Drift Segmentation via Kolmogorov-Trees." In: *ESANN*. 2021.

[58] Manuel Baena-Garcıa et al. "Early drift detection method". In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. Citeseer. 2006, pp. 77–86.

[59] Albert Bifet and Ricard Gavaldà. "Learning from Time-Changing Data with Adaptive Windowing". In: *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. 2007, pp. 443–448. DOI: 10.1137/1.9781611972771.42.

[60] Isvani Frias-Blanco et al. "Online and non-parametric drift detection methods based on Hoeffding's bounds". In: *IEEE Transactions on Knowledge and Data Engineering* 27.3 (2014), pp. 810–823.

[61] Michele Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*. Vol. 104. prentice Hall Englewood Cliffs, 1993.

[62] Yoshinobu Kawahara and Masashi Sugiyama. "Change-point detection in time-series data by direct density-ratio estimation". In: *Proceedings of the 2009 SIAM international conference on data mining*. SIAM. 2009, pp. 389–400.

[63] Kenji Yamanishi and Jun-ichi Takeuchi. "A unifying framework for detecting outliers and change points from non-stationary time series data". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, pp. 676–681.

[64] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. "Detecting change in data streams". In: *VLDB*. Vol. 4. Toronto, Canada. 2004, pp. 180–191.

[65] Shohei Hido et al. "Unsupervised change analysis using supervised learning". In: *Advances in Knowledge Discovery and Data Mining: 12th Pacific-Asia Conference, PAKDD 2008 Osaka, Japan, May 20-23, 2008 Proceedings 12*. Springer. 2008, pp. 148–159.

[66] Anton Dries and Ulrich Rückert. "Adaptive concept drift detection". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 2.5-6 (2009), pp. 311–327.

[67] KOLMOGOROV AN. "Sulla determinazione empirica di una legge didistribuzione". In: *Giorn Dell'inst Ital Degli Att* 4 (1933), pp. 89–91.

[68] Zaid Harchaoui et al. "A regularized kernel-based approach to unsupervised audio segmentation". In: *2009 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2009, pp. 1665–1668.

[69] Hao Chen and Nancy Zhang. "Graph-based change-point detection". In: (2015).

[70] Paul R Rosenbaum. "An exact distribution-free test comparing two multivariate distributions based on adjacency". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.4 (2005), pp. 515–530.

[71] Zaid Harchaoui and Olivier Cappé. "Retrospective mutiple change-point estimation with kernels". In: *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*. IEEE. 2007, pp. 768–772.

[72] Eamonn Keogh et al. "An online algorithm for segmenting time series". In: *Proceedings 2001 IEEE international conference on data mining*. IEEE. 2001, pp. 289–296.

[73] Corinne Jones and Zaid Harchaoui. "End-to-End Learning for Retrospective Change-Point Estimation". In: *30th IEEE International Workshop on Machine Learning for Signal Processing*. 2020. DOI: 10.1109/MLSP49062.2020.9231768.

[74] Corinne Jones et al. "A kernel-based change detection method to map shifts in phytoplankton communities measured by flow cytometry". In: *Methods in Ecology and Evolution* 12.9 (2021), pp. 1687–1698. DOI: https://doi.org/10.1111/2041-210X.13647.

[75] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[76] Fabian Hinder et al. “Localization of Concept Drift: Identifying the Drifting Datapoints”. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–9.

[77] Rafael Izbicki and Ann B Lee. “Converting high-dimensional regression to high-dimensional conditional density estimation”. In: *Electronic Journal of Statistics* 11.2 (2017), pp. 2800–2831.

[78] Geoffrey I Webb et al. “Analyzing concept drift and shift from sample data”. In: *Data Mining and Knowledge Discovery* 32 (2018), pp. 1179–1199.

[79] Kevin B Pratt and Gleb Tschapek. “Visualizing concept drift”. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 735–740.

[80] X. Wang et al. “ConceptExplorer: Visual analysis of concept drifts in multi-source time-series data”. In: *2020 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2020).

[81] Katharina J. Rohlfing et al. “Explanation as a Social Practice: Toward a Conceptual Framework for the Social Design of AI Systems”. In: *IEEE Trans. Cogn. Dev. Syst.* 13.3 (2021), pp. 717–728. DOI: 10.1109/TCDS.2020.3044366. URL: https://doi.org/10.1109/TCDS.2020.3044366.

[82] Fabian Hinder et al. “Contrasting explanation of concept drift”. In: *30th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*. 2022.

[83] Mengnan Du, Ninghao Liu, and Xia Hu. “Techniques for interpretable machine learning”. In: *Communications of the ACM* 63.1 (2019), pp. 68–77.

[84] Jarkko Venna et al. “Information retrieval perspective to nonlinear dimensionality reduction for data visualization.” In: *Journal of Machine Learning Research* 11.2 (2010).

[85] Alexander Schulz, Fabian Hinder, and Barbara Hammer. “DeepView: Visualizing Classification Boundaries of Deep Neural Networks as Scatter Plots Using Discriminative Dimensionality Reduction”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. Main track. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 2305–2311. DOI: 10.24963/ijcai.2020/319. URL: https://doi.org/10.24963/ijcai.2020/319.

[86] Weikai Yang et al. “Diagnosing concept drift with visual analytics”. In: *2020 IEEE conference on visual analytics science and technology (VAST)*. IEEE. 2020, pp. 12–23.

[87] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[88] Roland Nilsson et al. “Consistent feature selection for pattern recognition in polynomial time”. In: *The Journal of Machine Learning Research* 8 (2007), pp. 589–612.

[89] Lloyd S Shapley. *Notes on the N-person Game–I: Characteristic-point Solutions of the Four-person Game*. Rand Corporation, 1951.

[90] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. In: *arXiv preprint arXiv:1312.6034* (2013).

[91] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).

[92] A. Looveren and J. Klaise. “Interpretable Counterfactual Explanations Guided by Prototypes”. In: *CoRR* abs/1907.02584 (2019).

[93] Limin Yang et al. “{CADE}: Detecting and explaining concept drift samples for security applications”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 2327–2344.