A METHOD TO BENCHMARK HIGH-DIMENSIONAL PROCESS DRIFT DETECTION

EDGAR WOLF AND TOBIAS WINDISCH

University of Applied Sciences Kempten, Germany

Abstract. Process curves are multi-variate finite time series data coming from manufacturing processes. This paper studies machine learning methods for drifts of process curves. A theoretic framework to synthetically generate process curves in a controlled way is introduced in order to benchmark machine learning algorithms for process drift detection. A evaluation score, called the temporal area under the curve, is introduced, which allows to quantify how well machine learning models unveil curves belonging to drift segments. Finally, a benchmark study comparing popular machine learning approaches on synthetic data generated with the introduced framework shown.

1. Introduction

Manufacturing lines are typically decomposed into a sequential arrangement of individual steps, called *processes*, each utilizing one or more manufacturing techniques, such as casting, forming, separating, joining, or coating, to develop the component in accordance with its final specifications. Modern and encompassing sensor technology allows to monitor process behavior precisely, typically by measuring key performance indicators, like force, pressure, or temperature, over time. In modern IoT-enabled manufacturing lines, persisting these *process curves* has become the general case, allowing to monitor not only the process behavior when working on a single component, but also globally over multiple sequential executions [32]. Malign events like anomalous component batches, tool wear, or wrong calibrations can affect the line performance badly. Such events proceed subtle and often lead to a slow deformation of the resulting process curve iteration by iteration. Detecting such *process drifts*, that is time periods where machine behavior changes, is key to keep unplanned downtimes and costly bad parts at bay [19].

Specifically in high-volume production settings, where processes yield curves with multiple variables at a small rate and with short cycle times between subsequent executions, detecting process drifts is challenging due to the vast amount of data that is generated in short time. Thus, these settings have been an ideal application for machine learning methods [9, 15, 19, 30, 31, 2, 21]. Process drifts should not confused with concept or data drifts, where the goal is typically to analyse the declining performance of a trained machine learning model when new data starts to differ from the train data [22]. Thus, methods to detect concept drifts often have access to train data and trained machine learning models that can be used.

E-mail address: `edgar.wolf@hs-kempten.de, tobias.windisch@hs-kempten.de`.

In our setting, however, we are interested in machine learning models that detect the drift itself. Generally speaking, process curve datasets are datasets holding finitely many multivariate finite time series. Although process curves are time series by definition, many analysis techniques to analyse time series data do not apply because many methods, like ARMA [7] methods, assume infinite time series, often with strong assumptions on their stationarity.

When working with high-dimensional data, multivariate functional data analysis [16, 24] is the statistical workhorse. Deep learning techniques, particularly dimensionality reduction methods like *autoencoder* [25, 14], become increasingly popular [20]. A popular approach in practice is to learn an low-dimensional embedding of the high-dimensional input and then analyse the learned latent variables with classic machine learning models. There exist some architectures that also can directly consume temporal context, like the index of the process iteration, as an auxiliary variable to learn causal meaningful latent variables, like the iVAE [12] or the independent component analysis [11]. In [13], a derivate of an variational autoencoder has been developed which particularly deals with process drifts. In [31], Bayesian autoencoders where used to detect drifts in industrial environments. Moreover, more general neural network based systems, like for casting [15] or milling [9] have been developed to analyse process curves. Often, also expert knowledge can be utilized, like by extracting deterministic features which then are further analyzed by machine learning methods, like in [2] for process curve analysis of wire bounds. Once the dimensionality has been reduced, either by hand-crafted feature extraction of representation learning, classic methods for drift detection apply, like a sliding KS-test [23]. There are also methods that work in high-dimensional settings for tabular data, like hellinger-distance based techniques [5].

A common challenge when applying machine learning methods on applications from manufacturing is data imbalance as interesting events like scrap parts or drifts are rare. In [27], a deep generative model was used to synthetically generate process curves for casting processes, like to increase observations from the minority class.

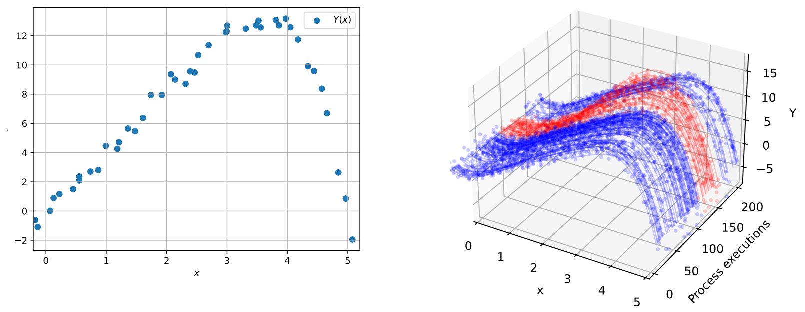

Fig. 1. Samples from a process curve (left) as well as a sequence of curve samples (right).

Overall, the analysis of process curves with machine learning techniques has been studied in depth for lots of applications from manufacturing. However, some inherent challenges remain. First, whether a model that works well in one application also does in another is by far not guaranteed. Thus, releasing data to the public and allowing to benchmark multiple models on the same data is important to make machine learning research more transparent for process curve analysis. Most papers mentioned above, no data has been released to the public, making it impossible to test other architectures. Often, this is due to privacy issues and fear of leaking information to competitors. Public real-world datasets to study process drifts are rare. For the special situation of milling, two process curve datasets [1, 28] exist that hold ground truth quantifying the health of the tool used. A comprehensive overview of degradation data sets for tasks with for tasks within prognostics and health management can be found in [18]. For causal discovery, a few approaches exist that realistically generated synthetic data, like for time-series [4] and tabular production data [8], however, the data generated is often identically and independently distributed (iid) and hence inappropriate for drift detection. The second challenge is how models should be evaluated. In most drift detection settings, data is imbalanced, meaning that there are far less drifts than normal data points. In [13], the false alarm rate and accuracy have been used to evaluate the model performance. A commonly used metric to measure the statistical performance of a binary classifier, like an anomaly detector, is the *area under the ROC curve* - short AUC. However, the usage of the AUC is typically only applicable in settings where data is assumed to be iid, which is not the case in drift detection.

In this work, we address the issue of benchmarking machine learning models for process drift detection. We present a simple, flexible and effective theoretic framework to generate synthetic process curves including drifts with a validated ground truth (Section 2 and Section 3). Its possible to feed data of real process curves making the data realistic. Moreover, we construct a metric called *temporal area under the curve* (TAUC) in Section 4, which aims to take the temporal context of a detection into account. Finally, we present a short benchmarks study in Section 5 as a proof of concept for the effectiveness of our score in measuring the predictive power of drift detectors. Our work is based on preliminary results of the first author [29]. In this work, we provide additionally insights into the introduced score, introduce a variant called *soft TAUC* and compare it in depth with existing scores. Moreover, we substantially generalize the data synthetization framework, for instance by allowing higher-order derivatives, and we generate more sophisticated datasets for the benchmark study. We also release the code that helps generating process curves to benchmark drift detectors, which is freely available under `https://github.com/edgarWolf/driftbench`. Its optimization back-end is implemented in `Jax` [3] allowing a fast GPU-based curve generation.

2. Statistical framework to analyse process drifts

A process curve is a finite time-series $C = (Y(x))_{x \in I}$ with $Y(x) \in \mathbb{R}^c$ and $I \subset \mathbb{R}$ a finite set, where $Y : \mathbb{R} \to \mathbb{R}^c$ represent physical properties of the process to be measured and x an independent variable, often the time (see Remark 2.1 for concrete examples). We call $c \in \mathbb{N}$ the *dimension* of the process curve. Whenever a manufacturing process finishes its work on a component, a process curve is yielded. Thus, when the same process is executed on multiple parts sequentially, a sequence $C_1, \dots, C_T$ is obtained where each C_t arises under slightly different physical conditions, i.e., $C_t = (Y_t(x))_{x \in I_t}$. The different conditions can be due to different properties of the components or due to degradation and wareout effects of tools within the machine running the process. Also, the elements in I_t can vary from execution to execution, for instance due to different offsets.

Remark 2.1. In staking processes Y is a measured force and x the walked path of the press. In pneumatic test stations, Y might be a pressure where x might be time. In bolt fastening processes, Y holds torque and angle and x is the time [21].

As new components are assembled each time the process is executed, the effect of a part only affects a given curve, whereas wareout of the tool, which is used in every process, affects all curves. Let $C_1, \dots, C_T$ be process curves from T-many executions and $Y_1, \dots, Y_T$ the corresponding physical relations. We write $[T]$ for the set $\{1, \dots, T\}$. In our approach, we assume that there exists a function $f : \mathbb{R}^k \times \mathbb{R} \to \mathbb{R}^c$ as well as a function $w : [T] \to \mathbb{R}^k$, such that for all $t \in [T]$ and $x \in I_t$:

$$f(w(t), x + \epsilon_x) + \epsilon_y = Y_t(x)$$

where the function f is a proxy for the physics underneath the process and ϵ_x and ϵ_y denote white noise relating to measurement inaccuracies that is independent of t. The vector $w(t)$ represents environmental properties of the t-th execution, and some of its coordinates correspond to component properties, some to properties of the machine.

Assuming no tool degradation but only component variance, we could assume that $w(t)$ is sampled in each process from a fixed but unknown distribution on $\mathbb{R}^k$, like $w(t) \sim \mathcal{N}_{\mu,\sigma}$ with fixed $\mu \in \mathbb{R}^k$ and $\sigma \in \mathbb{R}^{k \times k}$ for all $t \in [T]$. Tool degradation, however, affects the curve substantially, often by letting certain *support points* of the curve move. For instance, in a staking process, the position and value of the maximal force, i.e., where the first derivative is zero, starts shifting (see Figure 1). Often, these support points can be formulated in terms of derivatives of f. More formally, let ∂_x^i be the (i)-th derivative of f according to the second argument and let for $t \in [T]$, $x^i(t), y^i(t) \in \mathbb{R}^{n_i}$ be such support points of the t-curve C_t, i.e.:

$$\partial_x^i f(w(t), x_j^i(t)) = y_j^i(t) \tag{2.1}$$

for all $j \in [n_i]$. Particularly, the properties $w(t)$ deform in a way such that the derivatives of $f(w(t), \cdot)$ satisfies all support points. As mentioned before, if underlying physical properties of the process or components change, these support points start to change their position. More formally, at the t-th process execution, the support points are sampled according to a distribution, whose statistical parameter depend on t, for instance, for a particular support point $j \in [n_i]$ of y^i, the following temporal change could happen, i.e. $y_j^i(t) \sim \mathcal{N}_{\mu^i(t),\sigma}$, with

$$\mu^i(t) = \begin{cases} b, & \text{if } t < t_0 \\ d \cdot \frac{t - t_0}{t_1 - t_0} + b, & \text{if } t_0 \le t \le t_1 \\ d, & \text{if } t_1 < t \end{cases}$$

with $b \neq d$. That is, the mean of the support point drifts linearly from b to d between t_0 and t_1 (see also Figure 1).

3. Data generation

In this section, we explain the synthetization method of process curves. Here, we neither focus on how the $w(t)$ behave in latent space, nor on how f is formulated exactly. Instead, our key idea is to model the behavior of support points over time and seek for parameters $w(t)$ using non-linear optimization satisfying the support point conditions in (2.1). That way, we obtain a generic and controllable representation of the process curves that is capable of modelling physical transformations over time by letting the physically motivated support points drift over time. Without restricting generality and to keep notation simple, we will assume for the remainder that $c = 1$. The multivariate case is a straight-forward application of our approach by modeling each variable in Y individually (see Remark 3.1). For this, let $f : \mathbb{R}^k \times \mathbb{R} \to \mathbb{R}$ be an l-times differentiable map and let for each $i \in [l]$, $x^i \in \mathbb{R}^{n_i}$ and $y^i \in \mathbb{R}^{n_i}$ be two n_i-dimensional vectors. We want to find w^* such that

$$\partial_x^i f(w^*, x_j^i) \approx y_j^i \quad \forall j \in [n_i] \quad \forall i \in [l]. \tag{3.1}$$

Particularly, x^i and y^i can be considered as support points surpassed by the graph of the map $f(w^*, \cdot) : \mathbb{R} \to \mathbb{R}$ (see Figure 2). Assuming that $f \in \mathcal{C}^{l+2}(\mathbb{R}^k \times \mathbb{R})$, i.e. f is $l+2$-times

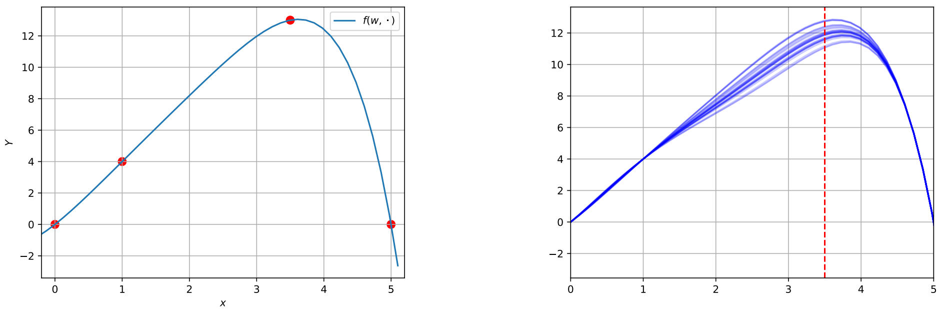

Fig. 2. Visualization of the data synthetization given a function $f(w, x) = \sum_{i=0}^{5} w_i \cdot x^i$. Left figure shows $f(w, \cdot)$ solved for concrete x^i, y^i (red points). Right figure shows sequence $f(w_1, \cdot), \ldots, f(w_{100}, \cdot)$ where gaussian noise was added on on coordinate in $y^1(t)$ (corresponding coordinate $x^1(t)$ is marked with a dashed line).

differentiable, and given $x^i \in \mathbb{R}^{n_i}$ and $y^i \in \mathbb{R}^{n_i}$ for each $i \in [l]$, we can solve (3.1) using second-order quasi-Newton methods [6, Chapter 3] for the objective function

$$\underset{w \in \mathbb{R}^k}{\arg\min} \sum_{i=1}^{l} \sum_{j=1}^{n_i} D_i \cdot \|\partial_x^i f(w, x_j^i) - y_j^i\|^2 \tag{3.2}$$

where $D_1, \ldots, D_l$ are constants to account for the different value ranges of the functions $\partial_x^i f$. Now, let $x^i(1), \ldots, x^i(T) \in \mathbb{R}^{n_i}$ and $y^i(1), \ldots, y^i(T) \in \mathbb{R}^{n_i}$ be sequences of T-many n_i-dimensional vectors respectively. Solving the optimization problem (3.2) for the each $t \in [T]$, i.e. for the $x^1(t), y^1(t), \ldots, x^l(t), y^l(t)$, we get a sequence of parameters $w_1, \ldots, w_T \in \mathbb{R}^k$ and a sequence of maps $f(w_1, \cdot), \ldots, f(w_T, \cdot)$ from $\mathbb{R}$ to $\mathbb{R}$.

Fixing now a finite and not necessarily equidistant set of points of interests $I \subset \mathbb{R}$ and setting $C_i = f(w_i, I) \in \mathbb{R}^{|I|}$, we obtain a sequence of process curves $C_1, \ldots, C_T$. In Appendix A, we showcase in depth an example where f is a polynomial.

Remark 3.1 (Multivariate data)**.** Of course, our theoretic framework extends naturally to multi-variate time series data, where each signal has either its own or multiple signals share same latent information.

Remark 3.2 (Feeding real-data)**.** Clearly, the support points $x^1(t), y^1(t), \ldots, x^l(t), y^l(t)$ can come from a real process curve dataset. Choosing a universal function approximator f like a feed-forward neural network [10] or a Kolmogorov-Arnold network [17].

4. The temporal area under the curve

In this section, we construct a score to measure the predictive power of machine learning models for process drift detection. In order to do so, we first formalize what we understand as a process drift and which assumptions we require. Let $C_1, \ldots, C_T$ be a sequence of process curves and let $\mathcal{D} \subset [T]$ be the set of curve indices belonging to drifts. Our first assumption is that drifts, different than point anomalies, appear sequentially and can be uniquely decomposed into disjoint segments:

Definition 4.1 (Drift segments)**.** Let $\mathcal{D} \subset [T]$. Then a series of subsets $\mathcal{D}_1, \ldots, \mathcal{D}_k \subset \mathcal{D}$ is a *partition of drift segments* if there exists $1 \leq l_1 < h_1 < l_2 < h_2 < \ldots, < l_k < h_k \leq T$ such that for all i, we have $\mathcal{D}_i = [l_i, h_i]$ and $\mathcal{D} = \cup_{i=1}^k \mathcal{D}_i$.

The drift segments can be considered as a partition of the smallest consecutive drifts which cannot decomposed any further into smaller segments. Now, assume we also have the output $s \in \mathbb{R}^T$ of a detector where each coordinate s_t quantifies how likely the curve C_t of the i-t-h process execution belongs to a drift, that is, the higher s_t the more likely the detector classifies $t \in \mathcal{D}$. By choosing a threshold $\tau \in \mathbb{R}$, we can construct a set

$$\hat{\mathcal{D}}(s, \tau) := \{t \in [T] : s_t \geq \tau\}$$

which serves as a possible candidate for $\mathcal{D}$. Clearly, if $\tau_1 \geq \tau_2$, then $\hat{\mathcal{D}}(s, \tau_1) \subseteq \hat{\mathcal{D}}(s, \tau_2)$. Its also straight-forward to see that for every τ, the set $\hat{\mathcal{D}}(s, \tau)$ decomposes uniquely into drift segments $\hat{\mathcal{D}}_1, \ldots, \hat{\mathcal{D}}_l$ as defined in Definition 4.1 and that the length and number of these atomic segments depends on τ. Now, to quantify the predictive power of the detector yielding s, one needs to quantify how *close* $\hat{\mathcal{D}}(s, \cdot)$ is to $\mathcal{D}$ when τ varies. There are many established set-theoretic measurements that are widely used in practice to quantify the distance between two finite and binary sets A and B, like the Jaccard index $\frac{|A \cap B|}{|A \cup B|}$, the Hamming distance $|A \backslash B| + |B \backslash A|$, or the Overlap coefficient $\frac{|A \cap B|}{\min(|A|, |B|)}$ just to name a few. Most scores, however, have as a build-in assumption that the elements of the set are iid and hence the temporal context is largely ignored making them unsuitable for process drift detection. Moreover, for most detectors we have to select a discrimination threshold τ, making evaluation cumbersome as it requires to tune the threshold on a separate held-out dataset. Moreover, in most practical scenarios, $\mathcal{D}$ is only a small subset and thus the evaluation metric has to consider highly imbalanced szenarios as well.

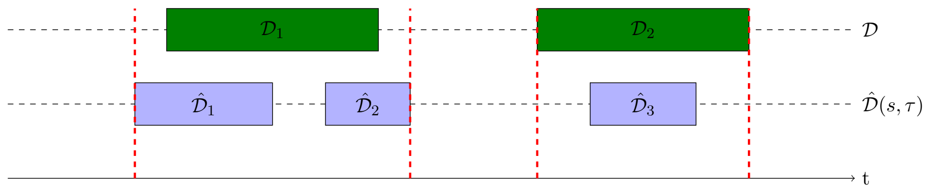

Fig. 3. Temporal arrangements of true and predicted drift segments as input for Algorithm 1.

Clearly, detectors are required where all true drift segments $\mathcal{D}_i$ are overlapped by predicted drift segments. For this, let $T_i := \{j \in [l] : \mathcal{D}_i \cap \hat{\mathcal{D}}_j \neq \emptyset\}$. Clearly, $T_i \cap T_{i+1} \neq \emptyset$ if $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ both intersect with a predicted drift segment. Now, the set $\mathcal{T}_i := \cup_{j\in T_i} \hat{\mathcal{D}}_j$ which is the union of all predictive segments intersecting with $\mathcal{D}_i$ serves as a candidate for $\mathcal{D}_i$. To measure how well $\mathcal{D}_i$ is covered - or overlapped - by $\mathcal{T}_i$ we define the *soft overlap score* inspired by the Overlap coefficient as follows:

$$\text{sOLS}(\mathcal{D}_i, s, \tau) := \frac{|\mathcal{T}_i|}{\max(\mathcal{T}_i \cup \mathcal{D}_i) - \min(\mathcal{T}_i \cup \mathcal{D}_i) + 1} \tag{4.1}$$

Obviously, an sOLS of 1 is best possible and this is reached if and only if $\mathcal{T}_i = \mathcal{D}_i$. It is easy to see that for fixed $\mathcal{D}_i$, the enlargement of $\mathcal{T}_i$ beyond the boundaries of $\mathcal{D}_i$ improves the overlap score, as $|\mathcal{T}_i|$ increases and one of either $\max(\mathcal{T}_i \cup \mathcal{D}_i)$ or $-\min(T_i \cup \mathcal{D}_i)$ increases as well. A special case is if $\mathcal{D}_i$ is completely covered by $\mathcal{T}_i$, i.e. $\mathcal{D}_i \subseteq \mathcal{T}_i$, then it follows that $\mathcal{T}_i$ is an interval as well and thus $\text{sOLS}(\mathcal{D}_i, s, \tau) = 1$. When $\mathcal{T}_i$ enlarges, then the number of false positives, i.e. the time points t contained in some $\hat{\mathcal{D}}_i$ and in the complement $\overline{\mathcal{D}} := [T] \setminus \mathcal{D}$ of the ground truth $\mathcal{D}$, enlarges as well. Thus, the predictive power of a detector is shown in the overlap score as well as the created false positive rate

$$\text{FPR}(\mathcal{D}, s, \tau) := \frac{|\hat{\mathcal{D}}(s,\tau) \cap \overline{\mathcal{D}}|}{|\overline{\mathcal{D}}|}.$$

into account. To also take false negatives into account, the enumerator in (4.2) could be changed as follows, yielding our final definition of the *overlap score*:

$$\text{OLS}(\mathcal{D}_i, s, \tau) := \frac{|\mathcal{T}_i \cap \mathcal{D}_i|}{\max(\mathcal{T}_i \cup \mathcal{D}_i) - \min(\mathcal{T}_i \cup \mathcal{D}_i) + 1}. \tag{4.2}$$

Algorithm 1 illustrates in detail how the *Overlap score* $\text{OLS}(\mathcal{D}, s, \tau)$ can be computed algorithmically.

Our score considers both, the OLS as well as the FPR, which mutually influence each other. In the computation of the AUC, any threshold τ from $[\min_t(s_t), \max_t(s_t)]$ yields a pair of false positive rate $\text{FPR}(\mathcal{D}, s, \tau)$ and true positive rate $\text{TPR}(\mathcal{D}, s, \tau)$ which can be drawn as a curve in the space where FPR is on the x-axis and TPR on the y-axis. Similarly, we define the *temporal area under the curve*, or just TAUC, as the area under the FPR-OLS curve

Algorithm 1 Overlap score OLS$(\mathcal{D}, s, \tau)$

```
Input: $\mathcal{D} \subset [T],\ s \in \mathbb{R}^T, \tau \in \mathbb{R}$
Output: Overlap score
1: $\mathcal{D}_1, \dots, \mathcal{D}_k \leftarrow$ find drift segments of $\mathcal{D}$
2: $\hat{\mathcal{D}}_1, \dots, \hat{\mathcal{D}}_l \leftarrow$ find drift segments of $\hat{\mathcal{D}}(s, \tau)$
3: $s \leftarrow \mathbf{0} \in \mathbb{R}^k$
4: for $i \in [k]$ do
5:     $T_i \leftarrow \{j \in [l] : \hat{\mathcal{D}}_j \cap \mathcal{D}_i \neq \emptyset\}$     ▷ All predicted drift segments overlapping with $\mathcal{D}_i$
6:     $\mathcal{T}_i \leftarrow \cup_{j \in T_i} \hat{\mathcal{D}}_j$     ▷ Union of all segments intersecting with $\mathcal{D}_i$
7:     $s_i \leftarrow \frac{|\mathcal{T}_i \cap \mathcal{D}_i|}{\max(\mathcal{T}_i \cup \mathcal{D}_i) - \min(\mathcal{T}_i \cup \mathcal{D}_i) + 1}$     ▷ fraction of overlap
8: end for
9: return $\frac{1}{k} \sum_{i=1}^{k} s_i$
```

while the discrimination threshold τ varies. We refer to the *soft* TAUC, or just sTAUC, to the area under the FPR-sOLS curve (see Figure 4).

Note that the integral of the curve can be computed using two different methods, the *step rule* and the *trapezoidal rule* and depending on which method is used, the value of the score may differs. We showcase this behavior in detail for trivial detectors in Appendix C. In Appendix B, we investigate in several synthetic cases in depth the differences and similarities between sTAUC, TAUC, and AUC.

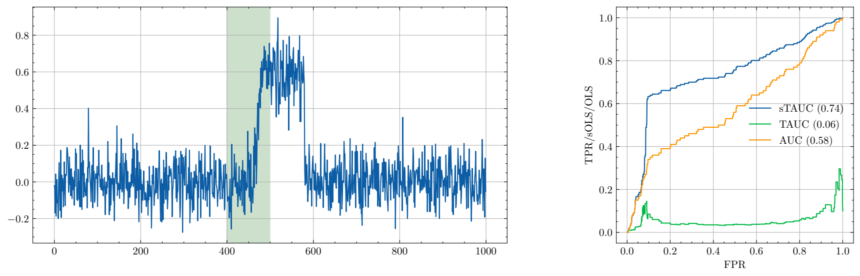

Fig. 4. The TPR, sOLS, and OLS when the FPR varies for the synthetic prediction on the left.

Example 4.2. Consider the situation shown in Figure 3. There, we have two true drift segments $\mathcal{D}_1$ and $\mathcal{D}_2$, and three segments $\hat{\mathcal{D}}_1$, $\hat{\mathcal{D}}_2$ and $\hat{\mathcal{D}}_3$ as drift segments of some detector output $\hat{\mathcal{D}}(s, \tau)$. Clearly, $T_1 = \{1, 2\}$ where $T_2 = \{3\}$ as only $\mathcal{D}_3$ overlaps with $\mathcal{D}_2$. To unveil $\mathcal{D}_1$, the detector needs to separate drift segments, leading to false negatives and positives and thus a relatively low OLS. On the other hand, as $\hat{\mathcal{D}}_3 \subset \mathcal{D}_2$, we have $\mathcal{T}_2 = \hat{\mathcal{D}}_3$.

5. Experiments

Next, we benchmark existing algorithms on data generated with our framework `driftbench` and reporting the TAUC[1]. The goal of the benchmark is to provide a proof of concept for our score and data generation method, not to be very comprehensive on the model side. Thus, based on our literature research in Section 1 we have hand-selected a small set of typically used model patterns drift detectors used in practice consists of (see Section 5.1).

The basic evaluation loop follows a typical situation from manufacturing, where process engineers have to identify time periods within a larger curve datasets where the process has drifted. Thus, all models consume as input a process curve dataset $C_1, \dots, C_T$ and do not have access to the ground truth $\mathcal{D}$, which is the set of curves belonging to a drift (see Section 5.3). Afterwards, each model predicts for each curve C_t from this dataset, a score s_t on which then the TAUC, sTAUC, and AUC are computed. To account for robustness, we generate each dataset of a predefined specification multiple times with different random seeds each, leading to slightly different datasets of roughly same complexity.

5.1. **Algorithms.** The algorithms used can be decomposed into multiple steps (see also Figure 5), but not all algorithms use all steps. First, there are may some features extracted from each curve. Afterwards, a sliding window collects and may aggregates these such that a score is computed.

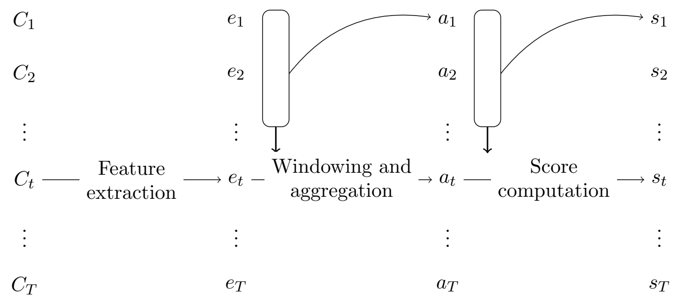

Fig. 5. A high-level overview of the elementary tasks of the detectors used.

5.1.1. *Feature extraction.* In this steps, we use autoencoders [25] to compute a k dimensional representation $e_t \in \mathbb{R}^k$ for each high-dimensional process curve C_t with k small. The indention behind is to estimate an inverse of the unknown function f and to recover information about the support points used. Moreover, we also apply deterministic aggregations over the x-information of each curve C_t,

5.1.2. *Windowing and aggregation.* In this step, the algorithms may aggregate the data from the previous step using a fixed window of size m that is applied in a rolling fashion along the

[1]All datasets and algorithms used are available in the repository of `driftbench`.

process iterations. One aggregation we use is to first compute for each coordinate $j \in [k]$ of $e_t \in \mathbb{R}^k$ with $t \geq m$ the rolling mean:

$$a_{t,j} = \frac{1}{m} \sum_{i=t-m+1}^{t} e_{i,j}. \tag{5.1}$$

These values can then further be statistically aggregated, like by taking the maximum $a_t := \max\{a_{t,j} : j \in [k]\}$.

5.1.3. *Score computing.* Goal of this step is to compute a threshold which correlates with the ground truth, that is, the larger the higher the possibility of a drift. Here, we may also aggregate previous features in a rolling fashion. The simplest aggregation we use is to compute the euclidean distance of subsequent elements $s_t = \|a_t - a_{t-1}\|_2$ which is just the absolute difference if a_t and a_{t-1} are scalars. If a_t is a scalar, we also can compute the rolling standard deviation, again over a window of size m, like this:

$$s_t = \sqrt{\frac{1}{m-1} \sum_{j=t-m+1}^{t} \left(a_j - \left(\frac{1}{m} \sum_{i=t-m+1}^{t} a_i \right) \right)^2}.$$

Another approach follows a probabilistic path by testing if a set of subsequent datapoints $\{a_{t-m+1}, \dots, a_t\}$ come from the same distribution as a given reference set. In our study, we use a windowed version [23] of the popular Kolmogorov-Smirnov test [26], often called KSWIN, which makes no assumption of the underlying data distribution. However, this can only be applied when a_t is a scalar. More particularly, we define two window sizes, m_r for the reference data and m_o for the observation. The windows are offset by constant $\delta > 0$. We then invoke the KS-test and receive a p-value p_t, which is small if the datasets come from different distributions. Thus, one way to derive a final score is to compute $s_t = \log(1 + \frac{1}{p_t})$. We also evaluate algorithms that derive their score based on a similarity search within $\{a_1, \dots, a_t\}$. Here, we use clustering algorithms, like the popular k-means algorithm, and use the euclidean distance to the computed cluster center of a_t as s_t. Another way is to fit a probability density function on s_t, like a mixture of Gaussian distributions, and to set s_t as the log likelihood of a_t within this model.

5.2. **Algorithm Overview.** Here is a short summary of the algorithms used in our benchmark study:

- `RollingMeanDifference`(m_r) First, the rolling mean over a window of size m_r is computed over all values for the respective curves in the window. Afterwards, the maximum value for each curve is taken and the absolute difference between two consecutive maximum values is computed.
- `RollingMeanStandardDeviation`(m_r) First, the rolling mean over a window of size m_r is computed over all values for the respective curves in the window. We also choose the maximum value of these computed values per curve. Then, we compute the standard deviation using the same window for this one-dimensional input.

- `SlidingKSWIN(`m_r, m_o, δ`)`: We compute the mean value for each curve and apply a sliding KS-test on this aggregated data. We use two windows of size m_r and m_o where the windows are offset by δ.
- `Cluster(`n_c`)`: A cluster algorithm performed on the raw curves using n_c clusters where score is distance to closest cluster center.
- `AE(`k`)-mean-KS(`m_r, m_o, δ`)`: First, an autoencoder is applied extracting computing k many latent dimensions. Afterwards, the mean across all k latent dimensions is computed. Finally, a sliding KS-test is applied with two windows of sizes m_r and m_o, where the windows are offset by δ.

5.3. **Datasets.** We benchmark the algorithms listed in 5.1 on three different datasets (see Figure 6) created with our framework `driftbench`, all designed to comprise different inherent challenges. To generate `dataset-1` and `dataset-2`, we used

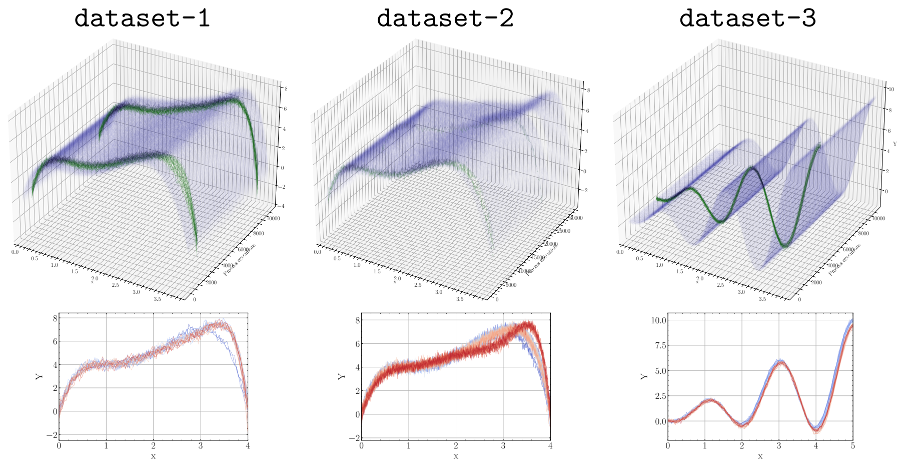

Fig. 6. The datasets used in our benchmark study. The true drift segments are marked in green. Lower figures show selected curves, whose color encodes the process iteration $t \in [T]$ - blue marks smaller t values, red larger ones.

$$f(w, x) = \sum_{i=0}^{7} w_i \cdot x^i$$

as function. The dataset `dataset-1` consists of $T = 10.000$ curves, each called on $|I| = 100$ equidistant values between $[0, 4]$. On the other hand, `dataset-2` consists of $T = 30.000$ curves each having $|I| = 400$ values. Both datasets have drifts that concern a movement of the global maximum together with drifts where only information of first order changes over time. In the generation process of `dataset-3`, we used

$$f(w, x) = w_0 \cdot x \cdot \sin(\pi \cdot x - w_1) + w_2 \cdot x$$

and generated $T = 10.000$ many curves, each having $|I| = 100$ datapoints. It only holds a single drift, where the global minimum at drifts consistently over a small period of time along the x-axis. In all datasets, the relative number of curves belonging to a drift is very small: roughly 2 percent in `dataset-1`, 0.1 percent in `dataset-2` and 1 percent in `dataset-3`.

5.4. **Results.** The result of our benchmark study is shown in Figure 7. We again want to mention that our benchmark study is not to show whether one model is superior over another. Instead, its purpose is to highlight how well the TAUC and our synthetization system allows to explore their performance. Surprisingly, the `RandomGuessDetector` reached the highest AUC score on `dataset-3`, where it ranges on all three datasets among the last ranks in the TAUC score. The respective predictions of the best detectors are plotted in Figure 9. Moreover, cluster based systems reach good AUC scores, but are not competitive when using the TAUC as quantification measurement (see also Figure 8). Unsurprisingly,

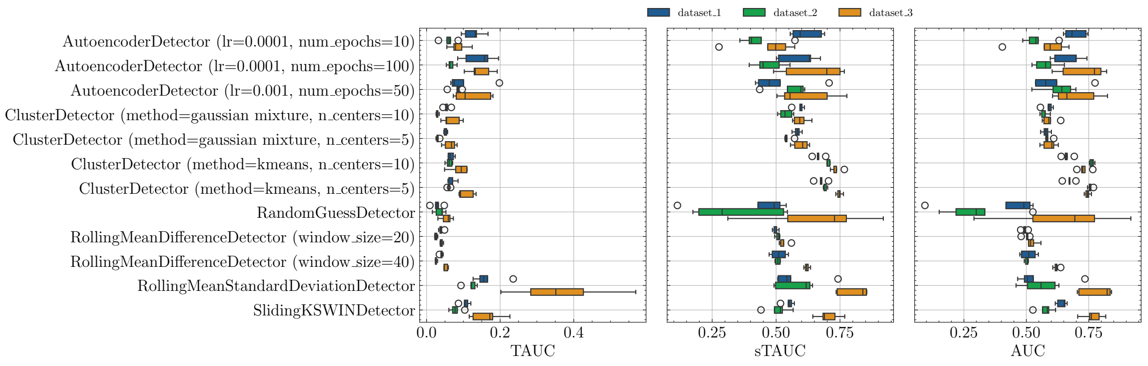

Fig. 7. Benchmark results on `dataset-1`, `dataset-2`, and `dataset-3`

the performance of autoencoders-based systems depends heavily on their hyperparameters, leading to large variations in the AUC score.

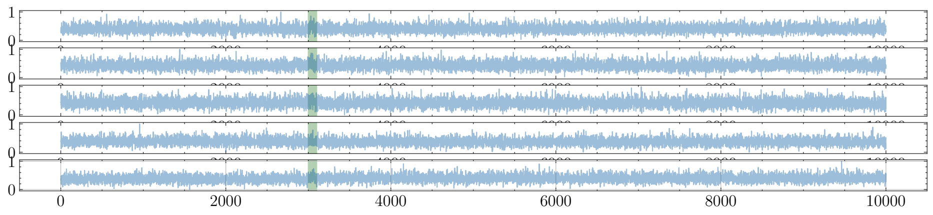

Fig. 8. Performance of cluster based systems on `dataset-3`, ranging among the models with highest AUC, but lowest TAUC.

Acknowledgements. This work is supported by the Hightech Agenda Bavaria. The authors are grateful to Matthias Burkhardt, Fabian Hueber, Kai Müller, and Ulrich Göhner for helpful discussions.

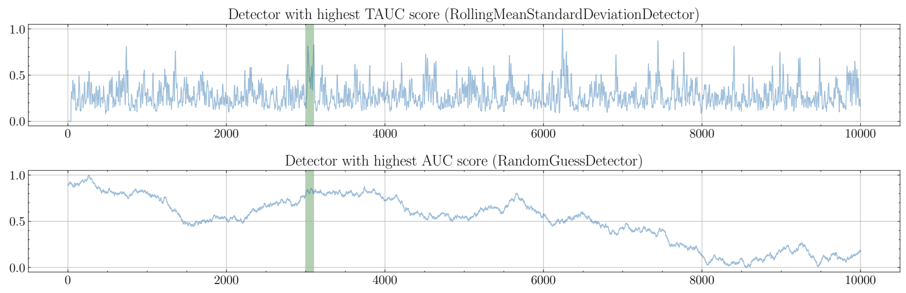

Fig. 9. Comparison of the prediction $s \in \mathbb{R}^T$ of the best detectors on `dataset-3`.

Data availability. The generated data and implemented algorithms are implemented in a python package `driftbench` which is freely available under `https://github.com/edgarWolf/driftbench`.

Conflict of interests. The authors provide no conflict of interest associated with the content of this article.

References

[1] A. Agogino and K. Goebel. Milling data set, BEST Lab, UC Berkeley, NASA Prognostics Data Repository, NASA Ames Research Center, 2007. URL: `https://data.nasa.gov/Raw-Data/Milling-Wear/vjv9-9f3x/about_data`.

[2] Anwar Al Assadi, David Holtz, Frank Nägele, Christof Nitsche, Werner Kraus, and Marco F. Huber. Machine learning based screw drive state detection for unfastening screw connections. 65:19–32. URL: `https://www.sciencedirect.com/science/article/pii/S0278612522001248`, `doi:10.1016/j.jmsy.2022.07.013`.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL: `http://github.com/google/jax`.

[4] Yuxiao Cheng, Ziqian Wang, Tingxiong Xiao, Qin Zhong, Jinli Suo, and Kunlun He. Causaltime: Realistically generated time-series for benchmarking of causal discovery. In *The Twelfth International Conference on Learning Representations*, 2024. URL: `https://openreview.net/forum?id=iad1yyyGme`.

[5] 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)*, pages 41–48, 2011. `doi:10.1109/CIDUE.2011.5948491`.

[6] Roger Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, NY, USA, second edition, 1987.

[7] John Gurland. Hypothesis testing in time series analysis., 1954. `doi:10.2307/2281054`.

[8] Konstantin Göbler, Tobias Windisch, Tim Pychynski, Mathias Drton, Steffen Sonntag, and Martin Roth. `causalAssembly`: Generating realistic production data for benchmarking causal discovery. In Francesco Locatello and Vanessa Didelez, editors, *3rd Conference on Causal Learning and Rasoning*, volume 236 of *Proceedings of Machine Learning Research*, pages 609–642, 2024.

[9] Daniel Frank Hesser and Bernd Markert. Tool wear monitoring of a retrofitted cnc milling machine using artificial neural networks. *Manufacturing Letters*, 19:1–4, 2019. URL: `https://www.sciencedirect.com/science/article/pii/S2213846318301524`, `doi:https://doi.org/10.1016/j.mfglet.2018.11.001`.

[10] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL: `https://www.sciencedirect.com/science/article/pii/0893608089900208`, `doi:https://doi.org/10.1016/0893-6080(89)90020-8`.

[11] Aapo Hyvarinen, Hiroaki Sasaki, and Richard E. Turner. Nonlinear ICA Using Auxiliary Variables and Generalized Contrastive Learning. URL: `http://arxiv.org/abs/1805.08651`, `arXiv:1805.08651`, `doi:10.48550/arXiv.1805.08651`.

[12] Ilyes Khemakhem, Diederik Kingma, Ricardo Monti, and Aapo Hyvarinen. Variational Autoencoders and Nonlinear ICA: A Unifying Framework. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 2207–2217. PMLR. URL: `https://proceedings.mlr.press/v108/khemakhem20a.html`.

[13] Youngju Kim, Hoyeop Lee, and Chang Ouk Kim. A variational autoencoder for a semiconductor fault detection model robust to process drift due to incomplete maintenance. *Journal of Intelligent Manufacturing*, 34(2):529–540, 2023. `doi:10.1007/s10845-021-01810-2`.

[14] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. `arXiv:http://arxiv.org/abs/1312.6114v10`, `doi:https://doi.org/10.48550/arXiv.1312.6114`.

[15] Jeongsu Lee, Young Chul Lee, and Jeong Tae Kim. Migration from the traditional to the smart factory in the die-casting industry: Novel process data acquisition and fault detection based on artificial neural network. *Journal of Materials Processing Technology*, 290:116972, 2021. URL: `https://www.sciencedirect.com/science/article/pii/S0924013620303939`, `doi:https://doi.org/10.1016/j.jmatprotec.2020.116972`.

[16] Yehua Li, Yumou Qiu, and Yuhang Xu. From multivariate to functional data analysis: Fundamentals, recent developments, and emerging areas. *Journal of Multivariate Analysis*, 188:104806, 2022. 50th Anniversary Jubilee Edition. URL: `https://www.sciencedirect.com/science/article/pii/S0047259X21000841`, `doi:https://doi.org/10.1016/j.jmva.2021.104806`.

[17] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

[18] Fabian Mauthe, Christopher Braun, Julian Raible, Peter Zeiler, and Marco F. Huber. Overview of publicly available degradation data sets for tasks within prognostics and health management, 2024. `arXiv:2403.13694`.

[19] Andreas Mayr, Dominik Kißkalt, Moritz Meiners, Benjamin Lutz, Franziska Schäfer, Reinhardt Seidel, Andreas Selmaier, Jonathan Fuchs, Maximilian Metzner, Andreas Blank, and Jörg Franke. Machine learning in production – potentials, challenges and exemplary applications. *Procedia CIRP*, 86:49–54, 2019. 7th CIRP Global Web Conference – Towards shifted production value stream patterns through inference of data, models, and technology (CIRPe 2019). URL: `https://www.sciencedirect.com/science/article/pii/S2212827120300445`, `doi:https://doi.org/10.1016/j.procir.2020.01.035`.

[20] Moritz Meiners, Marlene Kuhn, and Jörg Franke. Manufacturing process curve monitoring with deep learning. *Manufacturing Letters*, 30:15–18, 2021. URL: `https://www.sciencedirect.com/science/article/pii/S2213846321000742`, `doi:https://doi.org/10.1016/j.mfglet.2021.09.006`.

[21] Moritz Meiners, Andreas Mayr, and Jörg Franke. Process curve analysis with machine learning on the example of screw fastening and press-in processes. 97:166–171. URL: `https://www.sciencedirect.com/science/article/pii/S2212827120314414`, `doi:10.1016/j.procir.2020.05.220`.

[22] Yuri Thomas P. Nunes and Luiz Affonso Guedes. Concept drift detection based on typicality and eccentricity. *IEEE Access*, 12:13795–13808, 2024. `doi:10.1109/ACCESS.2024.3355959`.

[23] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. Reactive soft prototype computing for concept drift streams. *Neurocomputing*, 416:340–351, 2020. URL: `https://www.sciencedirect.com/science/article/pii/S0925231220305063`, `doi:https://doi.org/10.1016/j.neucom.2019.11.111`.

[24] Saptarshi Roy, Raymond K. W. Wong, and Yang Ni. Directed cyclic graph for causal discovery from multivariate functional data. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine,

editors, *Advances in Neural Information Processing Systems*, volume 36, pages 42762–42774. Curran Associates, Inc., 2023. URL: `https://proceedings.neurips.cc/paper_files/paper/2023/file/854a9ab0f323b841955e70ca383b27d1-Paper-Conference.pdf`.

[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[26] Richard Simard and Pierre L'Ecuyer. Computing the two-sided kolmogorov-smirnov distribution. *Journal of Statistical Software*, 39(11):1–18, 2011. URL: `https://www.jstatsoft.org/index.php/jss/article/view/v039i11`, `doi:10.18637/jss.v039.i11`.

[27] Jiyoung Song, Young Chul Lee, and Jeongsu Lee. Deep generative model with time series-image encoding for manufacturing fault detection in die casting process. *Journal of Intelligent Manufacturing*, 34(7):3001–3014, 2023. `doi:10.1007/s10845-022-01981-6`.

[28] Mohamed-Ali Tnani, Michael Feil, and Klaus Diepold. Smart data collection system for brownfield cnc milling machines: A new benchmark dataset for data-driven machine monitoring. *Procedia CIRP*, 107:131–136, 2022. Leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022. URL: `https://www.sciencedirect.com/science/article/pii/S2212827122002384`, `doi:https://doi.org/10.1016/j.procir.2022.04.022`.

[29] Edgar Wolf. Anomalieerkennung in hoch-dimensionalen Sensordaten. Master's thesis, University of Applied Sciences, Kempten, April 2024. `doi:https://doi.org/10.60785/opus-2338`.

[30] Pinku Yadav, Vibhutesh Kumar Singh, Thomas Joffre, Olivier Rigo, Corinne Arvieu, Emilie Le Guen, and Eric Lacoste. Inline drift detection using monitoring systems and machine learning in selective laser melting. *Advanced Engineering Materials*, 22(12):2000660, 2020. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/adem.202000660`, `arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/adem.202000660`, `doi:https://doi.org/10.1002/adem.202000660`.

[31] Bang Xiang Yong, Yasmin Fathy, and Alexandra Brintrup. Bayesian autoencoders for drift detection in industrial environments. In *2020 IEEE International Workshop on Metrology for Industry 4.0 and IoT*, pages 627–631, 2020. `doi:10.1109/MetroInd4.0IoT48571.2020.9138306`.

[32] Ray Y. Zhong, Xun Xu, Eberhard Klotz, and Stephen T. Newman. Intelligent manufacturing in the context of industry 4.0: A review. *Engineering*, 3(5):616–630, 2017. URL: `https://www.sciencedirect.com/science/article/pii/S2095809917307130`, `doi:https://doi.org/10.1016/J.ENG.2017.05.015`.

Appendix A. Data generation with polynomials

In this section, we demonstrate the data generation method introduced in Section 3 along an example involving a polynomial $f : \mathbb{R}^6 \times \mathbb{R} \to \mathbb{R}$ of degree five, i.e.

$$f(w,x) = \sum_{i=0}^{5} w_i \cdot x^i$$

with $w \in \mathbb{R}^6$. We simulate 2000 process executions and thus sample 2000 process curves. The shape of each curve is defined by its support points. We are only interested in its curvature in $I = [0,4]$. First, we want to add a condition onto the begin and end of the interval, namely that $f(w,0) = 4$ and $f(w,4) = 5$. Moreover, we would like to have a global maximum at $x = 2$, which means the first order derivative

$$\partial_x^1 f(w,2) = \sum_{i=1}^{4} i \cdot w_i \cdot 2^{i-1}$$

should be zero and its second order derivate

$$\partial_x^2 f(w,2) = \sum_{i=1}^{3} i \cdot (i-1) \cdot w_i \cdot 2^{i-2}$$

should be smaller than zero. Here, we want it to be -1. Finally, we want to the curve to be concave at around $x = -1$. All in all, these conditions result into the following equations, some of them are visualized in Figure 10:

$$\begin{aligned} &\partial_x^0 f(w,2) = 7 \qquad && \partial_x^1 f(w,2) = 0 \qquad && \partial_x^2 f(w,2) = -1 \\ &\partial_x^0 f(w,0) = 4 \qquad && \partial_x^0 f(w,4) = 5 \qquad && \partial_x^2 f(w,1) = -1 \end{aligned}$$

Then, we let the data drift at some particular features. We simulate a scenario, where the peak at x_1^0 and x_0^1 moves from the x-position 2 to 3 during the process executions $t = 1000$ until $t = 1300$. Thus, we let x_1^0 and x_0^1 drift from 2 to 3, resulting in a change of position of the peak. We let the corresponding y-values $y_1^0 = 7$ and $y_0^1 = 0$ unchanged. Now, we can solve each of the 2000 optimization problems, which results in 2000 sets of coefficients for each process curve, such that the conditions are satisfied. By evaluating f with the retrieved coefficients in our region of interest $[0,4]$, we get 2000 synthesized process curves with a drift present at our defined drift segment from $t = 1000$ until $t = 1300$.

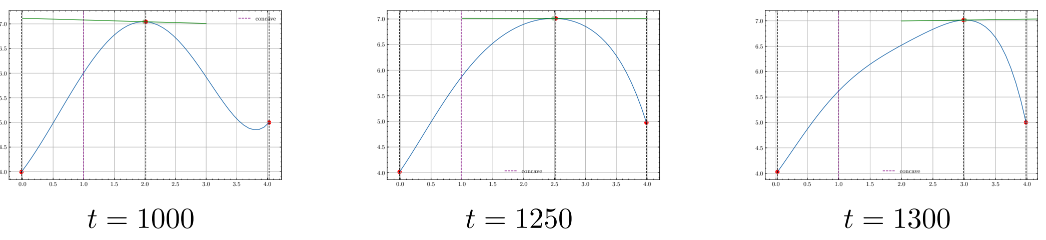

Fig. 10. Visualization of some process curves in the example dataset. The red dots indicate support points with first order information given. The green line visualizes the slope at the green dot, encoded by the condition for the first derivative. The purple dashed line indicates the curvature at the corresponding x-value, encoded by the condition for the second derivative. From $t = 1000$ to $t = 1300$, the x-value of the maximum moves from 2 to 3.

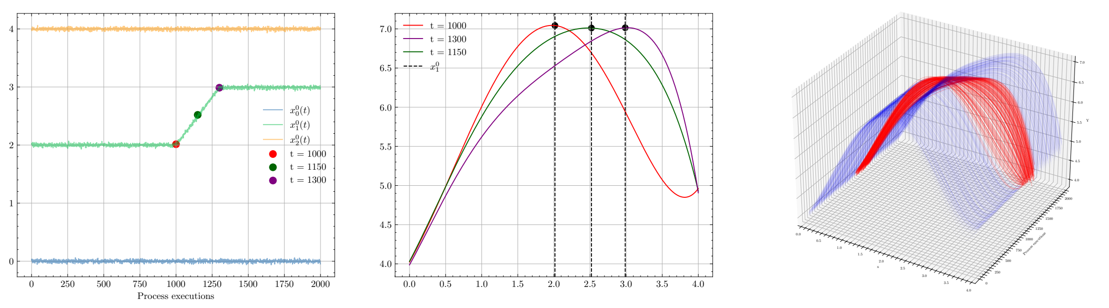

Fig. 11. Visualization of the drift applied on x_1^0 in this example, with respective curves. The left figure shows how the x_i^0 values change over time. Only x_1^0 changes, as by our drift definition from the process executions $t = 1000$ until $t = 1300$ linearly from 2 to 3, the others remain unchanged. The middle figure shows the respective curves, color-coded to the dots in the left figure.

Appendix B. TAUC vs AUC

In this section we explore in depth the similarities and differences of the TAUC introduced in Section 4 and the established AUC. This is done along synthetic predictions.

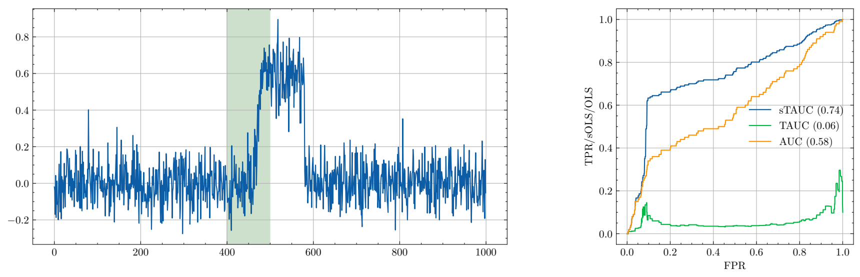

Fig. 12. Prediction of a detector that lags behind the ground truth (left) and its curves underneath the TAUC and AUC (right).

B.1. **Lagged prediction.** The first example we look at is a typical scenario that appears if window-based approaches are used, namely that the prediction lags a bit behind of the true window, but still the detector overlaps a significant proportion of the drift segment (see Figure 12. Other than the TPR, the sOLS rewards these predictors and thus the sTAUC shows a larger value than the AUC.

B.2. **Change point detection.** Another typical scenario is that a detector shows significantly large values at the start and end of the true drift segment, but sag in between (see Figure 13). This could appear when using methods based on detecting change points. In principal, the detector correctly identifies the temporal context of the drift segment, although showing lower scores while the curves drift. Such predictions also score higher values in the sTAUC than the AUC.

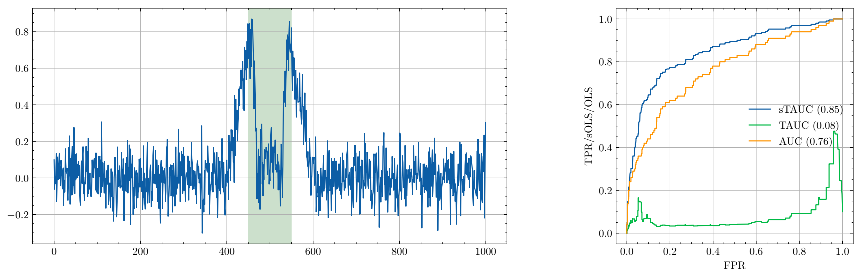

Fig. 13. Prediction of a detector that shows high scores at the boundary of the true drift segment only (left) and its curves underneath the TAUC and AUC (right).

B.3. **Varying length and position of predicted segments.** A situation where the sTAUC coincides with the AUC mostly is in when only one true and predicted drift segment exist (see Figure 14). In cases where the center of the predicted segment coincides with the center of the true segment, the AUC and sTAUC match almost exactly when the length of the predicted segment is varied (see left graphic in Figure 15). If the predicted segment has fixed length that equals the length of the true segment and the position of its center is varied from 50 to 350, AUC and sTAUC coincide mostly, but the sTAUC shows a faster rise when the predicted segment overlaps with the true segment due to the effects explained in Section B.1.

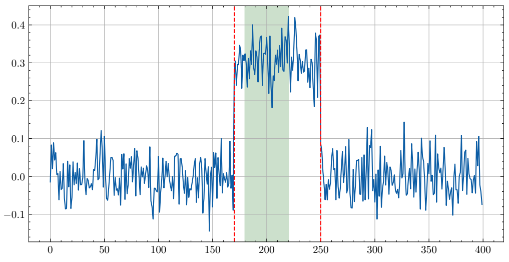

Fig. 14. Situation with single predicted segment (red dashed) and single true segment (green area).

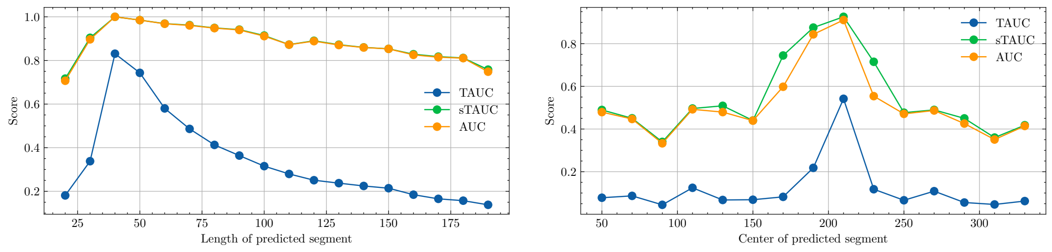

Fig. 15. Behavior of sTAUC, TAUC, and AUC when length and position of predicted segment varies.

Appendix C. TAUC for trivial detector

To get a better understanding of the TAUC, we showcase the behavior on trivial detectors based on the structure of the ground truth. Suppose two pair of points $(\mathrm{FPR}_i, \mathrm{OLS}_i)$ and $(\mathrm{FPR}_{i+1}, \mathrm{OLS}_{i+1})$ of the constructed curve. Then the two methods for computing the TAUC are the following:

- **Trapezoidal rule:**
  Construct the curve by linearly interpolating OLS_i and OLS_{i+1} in between FPR_i and FPR_{i+1} and then calculate the area under the curve by using the trapezoidal integration rule.
- **Step rule:**
  Construct the curve by filling the values in between FPR_i and FPR_{i+1} with a constant value of OLS_i and then calculate the area under the curve by using the step rule.

For example, take the trivial detector that always predicts a drift, called `AlwaysGuesser`. Then we receive the two points $(0, 0)$ and $(1, \frac{P}{k})$ as the only two points of the curve, where

P denotes the portion of drifts in y and k denotes the number of drift segments in y. In case of the step function, the computed score will always be 0, since the constructed curve only contains one step from $[0, 1)$ with a OLS-value of 0, and only reaches a OLS-value of $\frac{P}{k}$ when reaching a FPR of 1 on the x-axis. Hence, the area under this constructed curve is always 0. When using the trapezoidal rule, we linearly interpolate the two obtained trivial points of the curve, thus constructing a line from $(0, 0)$ to $(1, \frac{P}{K})$. The TAUC is then given by the area under this line, which is equal to $\frac{P}{2k}$. Now suppose a detector which never indicates a drift, called `NeverGuesser`. Then we receive $(0, 0)$ as our only point, which does not construct a curve and thus does not have an area under it. Hence, the TAUC for this trivial detection is 0 in both cases.

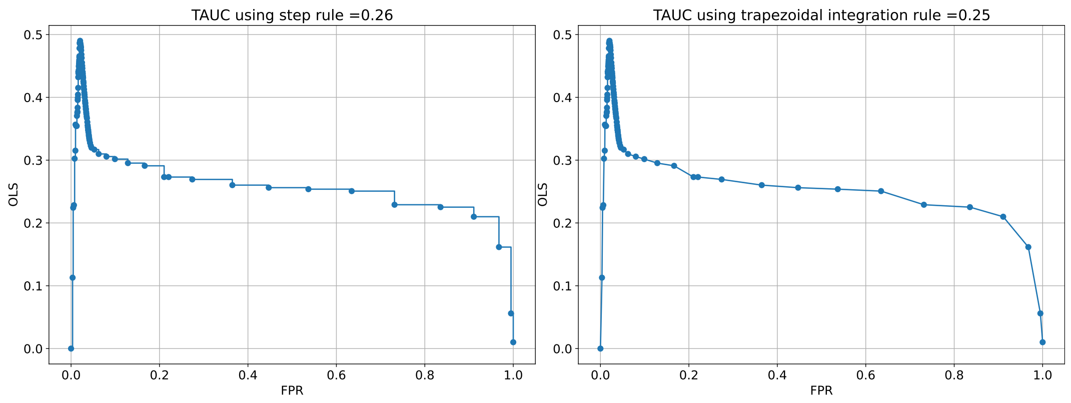

Fig. 16. Visualization of a concrete curve used to calculate the TAUC with its TAUC-score. Left figure shows the constructed curve when using the step rule, while the right figure shows the curve when calculating the TAUC using the trapezoidal integration rule.

In order to investigate how the TAUC behaves with an increasing number of segments k in y, we simulate such inputs with a trivial detection and compute the resulting values for the TAUC. We choose an input length of $n = 1000$. When using the step rule, the TAUC is always 0 as expected, since the only step always retains its area under the curve of 0. But when looking at the obtained TAUC values when using the trapezoidal integration rule, we can clearly see the TAUC decreasing when k increases.

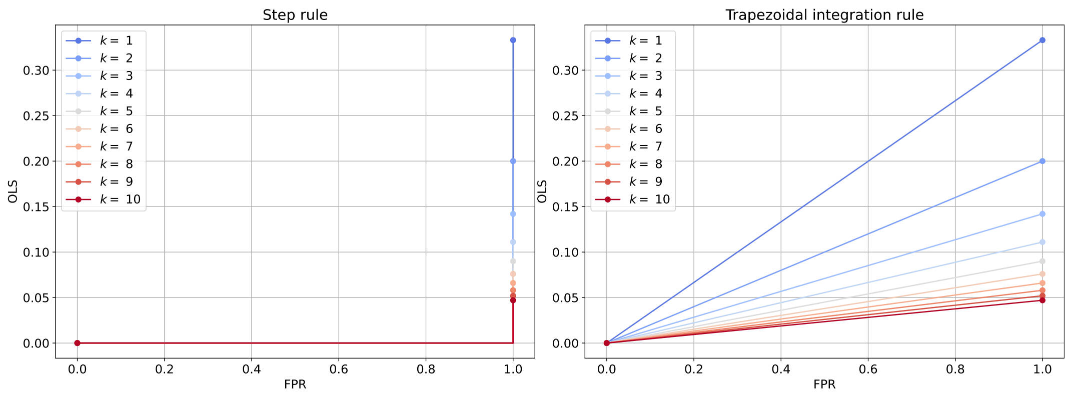

Fig. 17. Visualization of the behaviour of the constructed curve for the TAUC on increasing number of segments k. The left figure shows that the TAUC for the computation with the step rule always remains 0. The right figure shows that the area under the line decreases with increasing k, resulting in a lower TAUC value in case of the trapezoidal integration rule.

This decreasing behaviour can be approximated by $\frac{1}{2k}$, since the TAUC for a trivial detection with k segments in case of the trapezoidal rule can be computed with $\frac{P}{2k}$ and $0 < P \leq 1$. Thus, the limit of the TAUC computed with the trapezoidal integration rule with increasing k follows as:

$$\lim_{k \to \infty} \frac{P}{2k} = 0$$

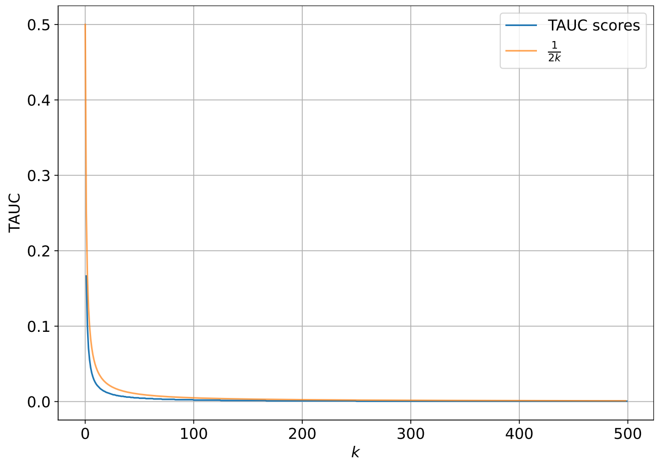

Fig. 18. Visualization of the TAUC with the trapezoidal integration rule, when increasing k, alongside an approximation $\frac{1}{2k}$. The TAUC gets closer to 0 with increasing k.