Maximum Mean Discrepancy on Exponential Windows for Online Change Detection

Anonymous authors
Paper under double-blind review

Abstract

Detecting changes is of fundamental importance when analyzing data streams and has many applications, e.g., in predictive maintenance, fraud detection, or medicine. A principled approach to detect changes is to compare the distributions of observations within the stream to each other via hypothesis testing. Maximum mean discrepancy (MMD), a (semi-)metric on the space of probability distributions, provides powerful non-parametric two-sample tests on kernel-enriched domains. In particular, MMD is able to detect any disparity between distributions under mild conditions. However, classical MMD estimators suffer from a quadratic runtime complexity, which renders their direct use for change detection in data streams impractical. In this article, we propose a new change detection algorithm, called Maximum Mean Discrepancy on Exponential Windows (MMDEW), that combines the benefits of MMD with an efficient computation based on exponential windows. We prove that MMDEW enjoys polylogarithmic runtime and logarithmic memory complexity and show empirically that it outperforms the state of the art on benchmark data streams.

1 Introduction

Data streams are possibly infinite sequences of observations that arrive over time. They can have different sources: sensors in industrial settings, online transactions from financial institutions, click monitoring on websites, online feeds, etc. Quickly detecting when a change takes place can yield useful insights, for example, about machine failure, malicious financial transactions, changes in customer preferences, and public opinions.

A *change* occurs if the underlying distribution of the data stream changes at a certain point in time. We call this moment *change point* (Gama, 2010); it is sometimes also referred to as *concept drift*. A principled and widely-used approach to detect changes is to use two-sample tests. The null hypothesis of such tests is that the data before and after the potential change point follow the same distribution. If the test rejects the hypothesis, one assumes that a change occurred.

One way to construct these tests is to use the kernel-based maximum mean discrepancy (MMD; Smola et al. 2007; Gretton et al. 2012), which one can interpret as a (semi-)metric on the space of probability distributions.[1] In the statistics literature, MMD is also known as energy distance (Székely & Rizzo, 2004; 2005); see Sejdinovic et al. (2013) for the equivalence. MMD relies on the kernel mean embedding (Berlinet & Thomas-Agnan, 2004, Ch. 4); it uses a kernel function to map a probability distribution to a reproducing kernel Hilbert space (RKHS; Aronszajn 1950) and quantifies the discrepancy of the two distributions as their distance in the RKHS. MMD is a metric if the kernel mean embedding is injective; the kernel is then called characteristic (Fukumizu et al., 2008; Sriperumbudur et al., 2010). When using a characteristic kernel, the MMD two-sample test allows to distinguish any distributions given that their kernel mean embeddings exist, which is guaranteed under mild conditions.

Two-sample tests based on MMD are widely applicable, as there exist kernel functions for a multitude of Euclidean and non-Euclidean domains, for example, strings (Watkins, 1999; Cuturi & Vert, 2005), graphs (Gärtner et al., 2003; Borgwardt et al., 2020), or time series (Cuturi, 2011; Király & Oberhauser, 2019).

[1] A function is a semimetric if it is a metric but can be zero for distinct elements.

Another benefit of kernel-based two-sample tests is their high power. While, for Euclidean data, it has been shown that the power of such tests generally decreases in the high-dimensional setting (Ramdas et al., 2015), recent results (Cheng & Xie, 2024) establish that the power rather depends on the intrinsic dimensionality of the data. The intrinsic dimensionality is typically low in real-world settings so that the kernel-based two-sample tests there do not suffer the curse of dimensionality.

Despite these benefits, a well-known bottleneck of MMD-based approaches is their computational complexity. When comparing the distributions of two sets of data of sizes m and n, respectively, the computation of MMD with classical estimators is in $\mathcal{O}\left(m^2 + n^2\right)$, with a memory complexity in $\mathcal{O}\left(m + n\right)$. Naively computing MMD for each possible change point on a data stream with $t = m + n$ observations has a complexity in $\mathcal{O}\left(t^3\right)$ for each new observation. These properties render the direct application of MMD to change detection in data streams impractical.

In this paper, we introduce Maximum Mean Discrepancy on Exponential Windows (MMDEW), a change detection algorithm for data streams that solves the above bottleneck. Specifically, our **contributions** include the following.

- Our main contribution is MMDEW, a change detector based on an efficient online approximation of MMD. When considering the entire history of t observations, the proposed method has a memory requirement of $\mathcal{O}\left(\log t\right)$ and a runtime complexity of $\mathcal{O}\left(\log^2 t\right)$ for each new observation. Otherwise, the algorithm has constant runtime and memory requirements.
- To achieve these complexities, we introduce a new data structure, which allows to approximate the quadratic time MMD in an online setting. We accomplish the speedup by introducing windows that store summaries of the observations seen so far, and by storing a sample of logarithmic size of the observations per window.
- Our experiments on standard benchmark data sets show that MMDEW performs better than state-of-the-art change detectors on four out of the five tested data sets using the F_1-score. For the more challenging setting of short detection delays, the proposed algorithm is better on three out of six data sets.[2]

Outline. Section 2 summarizes related work. Section 3 introduces the definitions and Section 4 presents the proposed algorithm. We detail the experiments in Section 5. Section 6 concludes. We include illustrative proofs in the main text but defer technical proofs and additional details to the appendices.

2 Related work

A principled approach for comparing distributions in a stream is to use a statistical test. ADWIN (Bifet & Gavaldà, 2007) is a classic example but it is limited to univariate data and only detects changes in mean. ADWINK (Faithfull et al., 2019) alleviates the former by running one instance of ADWIN per feature and issues a change if a predefined number of the instances agree that a change occurred. Hence, the approach can only detect changes in the means of the marginal distributions; changes in higher moments or the covariance structure can not be detected. Still, the authors find that such an ensemble of univariate change detectors often outperforms multivariate detectors. WATCH (Faber et al., 2021) is a recent approach that uses a two-sample test based on the Wasserstein distance. However, the estimation of the Wasserstein distance requires density estimation, which is difficult for high-dimensional data (Scott, 1991). The method Dasu et al. (2009) is conceptually similar to our method, as it also relies on two-sample tests and is non-parametric, but it also requires density estimation.

In contrast, the computation of MMD-based two-sample tests does not become more difficult on high-dimensional data, which renders their usage for change detection on such data promising. We refer to Muandet et al. (2017) for a general overview of kernel mean embeddings and MMD.

There exist methods to compute MMD in the streaming setting, for example, linear time tests (Gretton et al., 2012), but their statistical power is low. Zaremba et al. (2013) introduce B-tests, which have higher

[2]We make our code available on `https://anonymous.4open.science/r/mmdew-change-detector-5FE7` during review and will make it publicly available after acceptance.

Table 1: Comparison of change detectors. Complexity — runtime complexity per new observation, ARL / MTD — type of known results, domain — data types, t — total number of observations, d — dimensionality (for Euclidean spaces), k — parameter, W — window length / block size, N — number of windows.

Algorithm	Complexity	ARL / MTD	Domain
ADWINK	$\mathcal{O}(dk \log W)$	empirical	$\mathbb{R}^d$
WATCH	unknown[a]	empirical	$\mathbb{R}^d$
Scan B	$\mathcal{O}(NW^2)$	analytical	topological
NEWMA	$\mathcal{O}(md)$[b]	analytical	$\mathbb{R}^d$
D3	$\mathcal{O}(W^3)$[c]	none	$\mathbb{R}^d$
IBDD	$\mathcal{O}(pq)$[d]	none	$\mathbb{R}^d$
MMDEW	$\mathcal{O}(\log^2 t)$	empirical	topological

power. However, both can not directly be used for change detection. Li et al. (2019) enable the estimation of MMD on data streams for change detection by introducing Scan B-statistics. In a similar spirit, Keriven et al. (2020) introduce NEWMA, which is based on random Fourier features (Rahimi & Recht, 2007), a well-known kernel approximation, to detect changes with MMD on streaming data. Harchaoui & Cappé (2007) apply kernel-based tests for offline change point detection on audio and brain-computer-interface data.

A conceptually different approach to find changes is using classifiers. D3 (Gözüaçik et al., 2019) maintains two consecutive sliding windows and trains a classifier to distinguish their elements. It reports a change if the classifier performance, measured by AUC, drops below a threshold. Another recent algorithm is IBDD (de Souza et al., 2021), which scales well with the number of features.

In our experiments, we compare MMDEW to ADWINK, WATCH, Scan B-Statistics, NEWMA, D3, and IBDD as these allow change detection on multivariate streams (in $\mathbb{R}^d$). These algorithms differ w.r.t. their runtime complexity, their theoretical properties, the data types that they can handle, and the types of changes that they can detect. We summarize their main properties in Table 1.[3] We consider the dimensionality d as constant for the complexities where its influence is dominated by other terms and for approaches not restricted to Euclidean domains.

3 Definitions and background

This section defines our problem and recalls kernels, the mean embedding, maximum mean discrepancy, and two-sample testing.

Problem definition. Let $(\mathcal{X}, \tau_{\mathcal{X}})$ be a topological space, $\mathcal{B}(\tau_{\mathcal{X}})$ the Borel sigma-algebra induced by $\tau_{\mathcal{X}}$, and $\mathcal{M}_1^+(\mathcal{X})$ the set of probability measures on $\mathcal{X}$ that are measurable w.r.t. $(\mathcal{X}, \mathcal{B}(\tau_{\mathcal{X}}))$. We consider a data stream, that is, a possibly infinite sequence of observations, $x_1, x_2, \ldots, x_t, \ldots$ for $t = 1, 2, \ldots$, and $x_t \in \mathcal{X}$. Each x_t is generated independently following some distribution $D_t \in \mathcal{M}_1^+(\mathcal{X})$. If there exists t^* such that for $i < t^*$ and $j \geq t^*$ we have $D_i \neq D_j$, then t^* is a change point, and our task is to detect it. We note that these definitions place few assumptions on the type of data, that is, we only require the data to reside in a topological space.

Kernel mean embedding. Let $\mathcal{H}$ be a reproducing kernel Hilbert space (RKHS) on $\mathcal{X}$, which means that the linear evaluation functional $\delta_x : \mathcal{H} \to \mathbb{R}$ defined by $\delta_x(f) = f(x)$ is bounded for all $x \in \mathcal{X}$ and $f \in \mathcal{H}$. By the Riesz representation theorem (Reed & Simon, 1972), there exists for each $x \in \mathcal{X}$ a unique vector $\phi(x) \in \mathcal{H}$ such that for every $f \in \mathcal{H}$ it holds that $f(x) = \delta_x(f) = \langle f, \phi(x) \rangle$. The function $\phi(x)$ is the reproducing kernel for x and also called feature map; it has the canonical form $x \mapsto k(\cdot, x)$, with the function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ the reproducing kernel associated to $\mathcal{H}$. With this kernel, it holds that $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle = \langle k(\cdot, x_1), k(\cdot, x_2) \rangle$ for all $x_1, x_2 \in \mathcal{X}$ (Steinwart & Christmann, 2008). The mean

[3] [a] We refer to their used implementation of the Wasserstein distance computation and the discussion therein (Mérigot, 2011, Ch. 6). [b] m is the number of random Fourier features and $m \ll d$. [c] The complexity results from the matrix inversion of the logistic regression model, which has cubic runtime cost. [d] Size of the constructed $q \times p$ image.

embedding of a probability measure $\mathbb{P} \in \mathcal{M}_1^+(\mathcal{X})$ is the element $\mu(\mathbb{P}) \in \mathcal{H}$ such that $\mathbb{E}_{X\sim\mathbb{P}}[f(X)] = \langle f, \mu(\mathbb{P})\rangle$ for all $f \in \mathcal{H}$. The mean embedding $\mu(\mathbb{P})$ exists if k is measurable and bounded (Sriperumbudur et al., 2010, Prop. 2), which we assume throughout the article.

Maximum mean discrepancy. MMD is defined by $\mathrm{MMD}(\mathbb{P}, \mathbb{Q}) = \|\mu(\mathbb{P}) - \mu(\mathbb{Q})\|$, where $\mu(\mathbb{P}), \mu(\mathbb{Q}) \in \mathcal{H}$ are the mean embeddings of $\mathbb{P}, \mathbb{Q} \in \mathcal{M}_1^+(\mathcal{X})$, respectively.

Let $X \sim \mathbb{P}$, $Y \sim \mathbb{Q}$ and X', Y' independent copies of X, Y, respectively. The squared population MMD (Gretton et al., 2012, Lemma 6) then takes the form

$$\mathrm{MMD}^2(\mathbb{P}, \mathbb{Q}) = \mathbb{E}\left[k(X, X')\right] + \mathbb{E}\left[k(Y, Y')\right] - 2\mathbb{E}\left[k(X, Y)\right],$$

where the expectations are taken w.r.t. to all sources of randomness. For observations $\hat{\mathbb{P}}_m = \{x_1, \ldots, x_m\} \overset{\text{i.i.d.}}{\sim} \mathbb{P}$ and $\hat{\mathbb{Q}}_n = \{y_1, \ldots, y_n\} \overset{\text{i.i.d.}}{\sim} \mathbb{Q}$, a biased estimator is obtained by replacing the population means with their empirical counterparts

$$\mathrm{MMD}^2\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) = \frac{1}{m^2}\sum_{i,j=1}^{m} k(x_i, x_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(y_i, y_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(x_i, y_j). \tag{1}$$

The runtime complexity of (1) is in $\mathcal{O}\left(m^2 + n^2\right)$. We will base our proposed approximation on (1).

Two-sample testing. To decide whether the value of $\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right)$ indicates a significant difference between $\mathbb{P}$ and $\mathbb{Q}$, one tests the null hypothesis $H_0 : \mathbb{P} = \mathbb{Q}$ versus its alternative $H_1 : \mathbb{P} \neq \mathbb{Q}$ by defining an acceptance region for a given level $\alpha \in (0, 1)$, which takes the form $\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) < \epsilon_\alpha$. One rejects H_0 if the test statistic exceeds the threshold. The level α is a bound for the probability that the tests rejects H_0 incorrectly (Casella & Berger, 1990). Gretton et al. (2012, Corollary 9) provides the distribution-free threshold ϵ_α for the case that both samples $\hat{\mathbb{P}}_m$ and $\hat{\mathbb{Q}}_m$ have the same size ($m = n$) as

$$\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_m\right) < \sqrt{\frac{2K}{m}}\left(1 + \sqrt{2\log\frac{1}{\alpha}}\right). \tag{2}$$

Computing (2) costs $\mathcal{O}(1)$. As the change detection setting requires the case that $m \neq n$, we extend their threshold accordingly in what follows.

4 Our proposed algorithm

We introduce MMDEW in three steps. We first extend the threshold for the MMD two-sample test (2) to samples of unequal sizes (Section 4.1). We then introduce our data structure that enables the efficient computation of MMD on data streams (Section 4.2). Last, we describe the complete algorithm in Section 4.3.

4.1 Threshold for the hypothesis test

Given a sequence of observations $\{x_1, \ldots, x_t\}$ up until time t our goal is to test the null hypothesis for any two neighboring windows $X \cdot Y = \{x_1, \ldots, x_k\} \cdot \{x_{k+1}, \ldots, x_t\}$, with $k = 1, \ldots, t-1$. Our following proposition extends Gretton et al. (2012, Theorem 8), which considers the case $m = n$, giving the distribution-free acceptance region for $m \neq n$ (corresponding to the setting that one generally encounters in change detection).

Proposition 1. *Let $\mathbb{P}, \mathbb{Q} \in \mathcal{M}_1^+(\mathcal{X})$, $\hat{\mathbb{P}}_m = \{x_1, \ldots, x_m\} \overset{i.i.d.}{\sim} \mathbb{P}$, $\hat{\mathbb{Q}}_n = \{y_1, \ldots, y_n\} \overset{i.i.d.}{\sim} \mathbb{Q}$. Assume that $0 \leq k(x, y) \leq K$ for all $x, y \in \mathcal{X}$ and $t > 0$. Then a hypothesis test of level $\alpha > 0$ for $\mathbb{P} = \mathbb{Q}$ has the acceptance region*

$$\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) < \sqrt{\frac{K}{m} + \frac{K}{n}}\left(1 + \sqrt{2\log\alpha^{-1}}\right) =: \epsilon_\alpha.$$

Note that, when considering multiple possible change points, one needs to account for multiple testing in order to achieve a level of size α. For example, one may adjust ϵ_α through Bonferroni correction ($\epsilon'_\alpha = \epsilon_\alpha/(t-1)$) by dividing by the total number of tests.

We now introduce our novel data structure that allows considering multiple possible change points efficiently.

4.2 Proposed data structure

One common method to obtain a good runtime complexity in change detection algorithms is to slice the data into windows of exponentially increasing sizes (Bifet & Gavaldà, 2007). Recent observations are collected in smaller windows, and older observations are grouped into larger windows. This leads to a fine-grained change detection in the recent past and more coarse-grained change detection in the distant past.

Our new data structure adopts this concept and, at the same time, facilitates the computation of MMD. In what follows, we first describe the properties of the proposed data structure. Then, we show how to update the data structure and explain its use for change detection.

4.2.1 Properties

We use 2 as the basis for the exponential slicing. Then, after observing t elements, the number of windows stored in the data structure corresponds to the number of ones in the binary representation of t. We may thus index the windows as $B_l, \ldots, B_0$ (in decreasing order), with the largest position being $l = \lfloor \log_2 t \rfloor$. A window does not exist if the binary representation of t at this position is zero.

If it exists, a window B_s at position $s = 0, \ldots, l$ stores 2^s observations

$$\mathrm{X}_s = \{x_1^s, \ldots, x_{2^s}^s\}, \tag{3}$$

together with the summaries

$$\mathrm{XX}_s = \sum_{i,j=1}^{2^s} k(x_i^s, x_j^s), \tag{4}$$

$$\mathrm{XY}_s = \Bigg\{ \underbrace{\sum_{i=1}^{2^s} \sum_{j=1}^{2^{s+1}} k\left(x_i^s, x_j^{s+1}\right)}_{=:\mathrm{XY}_s^{s+1}}, \ldots, \underbrace{\sum_{i=1}^{2^s} \sum_{j=1}^{2^l} k\left(x_i^s, x_j^l\right)}_{=:\mathrm{XY}_s^{l}} \Bigg\}, \tag{5}$$

where XX_s is the sum of the kernel k evaluated on all pairs of the window's own observations, and XY_s stores a list of sums of the kernel evaluated on the window's own observations and the observations in windows coming before it. Storing a list enables the efficient merging of windows, elaborated in Lemma 2. The length of the list XY_s equals the number of windows having observations older than window B_s and is at most $\lfloor \log_2 t \rfloor$. We use XY_i^j to represent the entry in XY_i that refers to the window B_j. Specifically, in (5), XY_s^{s+1} stores the interaction of B_s with B_{s+1}; similarly, XY_s^l stores its interaction with B_l.

We summarize two of the main properties of the data structure as lemmas. Lemma 1 establishes that one can compute the value of MMD between two windows with constant complexity and follows from comparing (4) and (5) with (1). Lemma 2 shows that windows can be merged with logarithmic runtime complexity. These results provide our first steps towards efficiently computing MMD in a data stream.

Lemma 1. *Let B_{s+1} and B_s be any two neighboring windows with elements $\mathrm{X}_{s+1} = \left\{x_1^{s+1}, \ldots, x_{2^{s+1}}^s\right\}$ and $\mathrm{X}_s = \{x_1^s, \ldots, x_{2^s}^s\}$, and sums as defined by (4) and (5), respectively. Then*

$$\mathrm{MMD}^2\left(X_{s+1}, X_s\right) = \frac{1}{(2^{s+1})^2}\mathrm{XX}_{s+1} + \frac{1}{(2^s)^2}\mathrm{XX}_s - \frac{2}{(2^{s+1})(2^s)}\mathrm{XY}_s^{s+1},$$

with a computational complexity of $\mathcal{O}(1)$.

Lemma 2. *Merging two windows B_{s+1} and B_s into a new window B', such that B' stores (3), (4), and (5) costs $\mathcal{O}(\log t)$.*

Besides showing the result, the proof of Lemma 2 illustrates the steps that allow merging windows efficiently.

Proof. For computing XX', we use the symmetry of k to obtain

$$\begin{aligned}\mathrm{XX}' &= \sum_{i,j=1}^{2^{s+1}} k\left(x_i^{s+1}, x_j^{s+1}\right) + \sum_{i,j=1}^{2^s} k\left(x_i^s, x_j^s\right) + \sum_{i=1}^{2^{s+1}}\sum_{j=1}^{2^s} k\left(x_i^{s+1}, x_j^s\right) + \sum_{i=1}^{2^s}\sum_{j=1}^{2^{s+1}} k\left(x_i^s, x_j^{s+1}\right) \\ &= \sum_{i,j=1}^{2^{s+1}} k\left(x_i^{s+1}, x_j^{s+1}\right) + \sum_{i,j=1}^{2^s} k\left(x_i^s, x_j^s\right) + 2\sum_{i=1}^{2^{s+1}}\sum_{j=1}^{2^s} k\left(x_i^{s+1}, x_j^s\right) = \mathrm{XX}_{s+1} + \mathrm{XX}_s + 2\mathrm{XY}_s^{s+1}, \end{aligned} \tag{6}$$

which has a runtime complexity in $\mathcal{O}(1)$.

To compute XY', we note that B_{s+1} stores the list XY_{s+1} of kernel evaluations corresponding to all windows coming before it. The same holds for B_s, for which the list has one more element, XY_s^{s+1}, which was used in (6). All the elements in XY_s and XY_{s+1} are sums and thus additive; it suffices to merge both lists by adding their values element-wise, omitting XY_s^{s+1}, and storing the result in XY'. As each list has at most $\log t$ elements, merging them is in $\mathcal{O}(\log t)$. □

Specifically, the scheme facilitates the merging of windows of equal size, enabling us to establish the exponential structure outlined in the next section.

4.2.2 Insertion of observations

The structure is set up recursively. For each new observation, we create a new window B_0, with XX_0 as defined by (4) and XY_0 computed w.r.t. the already existing windows. If two windows have the same size, we merge them by Lemma 2, which costs $\mathcal{O}(\log t)$. This yields $\lfloor \log t \rfloor$ windows of exponentially increasing sizes.

We illustrate the scheme in the following Example 1. Example 2 (shown later) will extend upon the example below and also gives a visualization.

Example 1. *To set up the structure, we start with the first observation x_1 and create the first window B_0, with XX_0 as defined by (4) and $\mathrm{XY}_0 = \emptyset$. When observing x_2, we similarly create a new window B_0', now also computing $\mathrm{XY}_{0'}^0 = \{\mathrm{XY}_{0'}^0\}$. As B_0 and B_0' have the same size, we merge them into B_1, computing XX_1 with (6). No previous window exists so that $\mathrm{XY}_1 = \emptyset$. We repeat this for all new observations, for example, for x_3, one creates (a new) B_0, computing XX_0 and $\mathrm{XY}_0 = \{\mathrm{XY}_0^1\}$, which results in two windows, B_1 and B_0.*

4.2.3 MMD computation and change detection

We now show that we can compute the MMD statistic (1) at positions between windows with a runtime complexity of $\mathcal{O}(\log t)$.

Proposition 2. *Let $B_l, \ldots, B_{s+1}, B_s, \ldots, B_0$ be a given list of windows with corresponding elements X_i, $i = 0, \ldots, l$, as defined in (3). The computation of*

$$\mathrm{MMD}^2\left(\bigcup_{i=s+1}^{l} \mathrm{X}_i, \bigcup_{i=0}^{s} \mathrm{X}_i\right) \tag{7}$$

has a runtime complexity on the order of $\mathcal{O}(\log t)$ for $0 < s < l$, with $s, l \in \mathbb{N}$.

Proof. To obtain (7), one recursively merges $B_s, \ldots, B_0$ to B_s' using Lemma 2, starting from the right, and similarly $B_l, \ldots, B_{s+1}$ to B_l'. One then obtains the statistic with Lemma 1, and by setting $\mathrm{XY}_{s'}^{l'} = \sum_{i=1}^{l-s} \mathrm{XY}_{s'}^i$, that is, by summing all elements in the XY_s'-list of B_s'. This concludes the proof as the logarithmic complexity was already established. □

The application of the presented data structure for change detection is as follows. For each new observation, we estimate MMD at any position between windows and compare it to the threshold $\epsilon'_\alpha = \frac{\epsilon_\alpha}{l}$ (with Bonferroni correction) from Proposition 1. We report a change when the value of MMD exceeds the threshold. As there are at most $\log t$ windows, we have at most $\log t - 1$ positions. Computing MMD for a position is in $\mathcal{O}(\log t)$ by Proposition 2, and so the procedure has a total runtime complexity of $\mathcal{O}\left(\log^2 t + t\right)$ per insert operation, where the term linear in t results from computing XY_0 when inserting a new observation.

While the data structure in its current form allows to obtain the precise values of (1) in an incremental fashion, its runtime and memory complexity are $\mathcal{O}(t)$ for each new observation; these complexities are unsuitable for deploying the algorithm in the streaming setting. We reduce this by subsampling within the windows, which we present together with the complete algorithm in the following section.

4.3 MMDEW Algorithm

Our algorithm builds upon the data structure discussed previously. But, we suggest that each window of size 2^s, $s = 0, \ldots, l$, samples s observations (of the total 2^s), that is, a logarithmic amount, while keeping everything else as before.

In this section, we first analyze such subsampling and discuss its benefits. Afterwards, we present the complete algorithm.

Proposition 3. *With subsampling, the number of terms in the sum* XX_l *for a window at position* l, $1 \leq l$, $l \in \mathbb{N}$ *is*

$$n_{\mathrm{XX}_l} = 2^{l-1}\left(l^2 - l + 4\right) = \frac{t}{2}\left(\log_2^2 t - \log_2 t + 4\right),$$

with $t = 2^l$ *the number of observations of* B_l*. The number of terms of* XY_l^l *for windows of the same size, which occur prior to merging, is*

$$n_{\mathrm{XY}_l^l} = 2^l l = t \log_2 t.$$

Remarks. The number of terms in the sums of (1) acts as a proxy for the quality of the estimate. It is optimal when no subsampling takes place; this number is $\mathcal{O}\left(t^2\right)$. When subsampling a logarithmic number of observations per window with our data structure (as we propose), one achieves polylogarithmic runtime and logarithmic memory complexity. At the same time, one achieves a better approximation quality than naively sampling a logarithmic number of observations without the summary data structure. While such sampling would also yield a memory complexity of $\mathcal{O}(\log t)$ when using the naive approach for change detection—that is, splitting the sample into two neighboring windows and computing MMD^2—the number of terms in (1) would be $\mathcal{O}\left(\log^2 t\right)$. The summary data structure improves upon this by a factor of approximately $t/2$ for n_{XX_l} and a factor of $t/\log_2 t$ for $n_{\mathrm{XY}_l^l}$ (we neglect logarithmic and constant terms in the former due to their small contribution).

Algorithm 1 now summarizes the complete algorithm, with MMD in Line 9 referring to the computation of MMD as in Proposition 2. MMDEW stores only a uniform sample of size $l+1$, that is, of size logarithmic in the number of observations, while keeping the respective XX_s and XY_s, $s = 0, \ldots, l$, computed before. With this approach, the number of samples in a window increases by one each time the window is merged, and the memory complexity is logarithmic in the number of observations. Note that one recovers the previous algorithm (Section 4.2.3) and therefore the precise value of (7) if one omits Line 14. Further, changes in Line 14 allow to adjust the subsampling, for example, the user may defer the sampling until windows contain a minimum number of observations, or choose a different function to control the sample size.

The following example illustrates the procedure.

Example 2. *We assume that there is a stream of i.i.d. observations* $x_1, x_2, \ldots$*. Note that the i.i.d. assumption implies that there are no changes. MMDEW receives the first observation,* x_1 *and creates a window* B_0 *storing* x_1*,* $\mathrm{XX}_0 = k(x_1, x_1)$*, and* $\mathrm{XY}_0 = \emptyset$*. For the next observation,* x_2*, it creates a new window* $B_{0'}$*, storing* x_2*,* $\mathrm{XX}_{0'} = k(x_2, x_2)$*, and* $\mathrm{XY}_{0'} = \{k(x_1, x_2)\}$ *and detects no change. As* B_0 *and* $B_{0'}$ *have the*

Input: Data stream $x_1, x_2, \ldots$, level α
Output: Change points in $x_1, x_2, \ldots$; detection times

windows $\leftarrow \emptyset$ ▷ List of windows
for each $x_i \in \{x_1, x_2, \ldots\}$ **do**
  $\mathrm{XX}_0 \leftarrow k(x_i, x_i)$ ▷ Initialize B_0
  $X_0 \leftarrow x_i$
  for each $B_j \in$ *windows* **do**
    $\mathrm{XY}_0^j \leftarrow \sum_{x_k^j \in B_j} k(x_i, x_k^j)$
  windows $\leftarrow$ *windows* $\cup B_0$
  for each split s in *windows* $= \{B_l, \ldots, B_0\}$ **do** ▷ Detect changes
    if $\mathrm{MMD}\left(\bigcup_{j=s+1}^{l} \mathrm{X}_j, \bigcup_{j=0}^{s} \mathrm{X}_j\right) \geq \epsilon'_\alpha$ **then**
      print "Change at s detected at time i"
      windows $\leftarrow B_s, \ldots, B_0$ ▷ Drop windows
  while two windows have the same size 2^l **do** ▷ Maintain exponential structure
    Merge windows following Lemma 2 into B_{l+1}
    Store a uniform sample of size $l+1$ in B_{l+1}

Figure 1: Proposed MMDEW change detection algorithm.

same size, MMDEW merges them into window B_1, storing a sample of size $\log_2 2 = 1$, say, it stores x_1 and discards x_2, and computes $\mathrm{XX}_1 = k(x_1, x_1) + k(x_2, x_2) + 2k(x_1, x_2)$, following (4). As no previous window exists, the computation of XY_1 is not required. We see that the number of terms in XX_1 equals four, while B_1 stores only one observation (established in Proposition 3). Next, the algorithm observes x_3 and creates a new window, B_0, storing x_3, $\mathrm{XX}_0 = k(x_3, x_3)$, and computing XY_0 to the window coming before, that is, B_1, so that $\mathrm{XY}_0 = \{\mathrm{XY}_0^1\}$. In the next step, MMDEW receives x_4, again creating a new window $B_{0'}$. The algorithm now recursively merges the windows, that is, B_0 and $B_{0'}$ become $B_{1'}$, and B_1 and $B_{1'}$ then become B_2.

Figure 2 expands upon Example 2 and shows the evolution of the data structure upon observing $x_1, \ldots, x_4$ and when merging windows.

Algorithm 1 has a runtime cost of $\mathcal{O}\left(\log^2 t\right)$ per insert operation and a total memory complexity of $\mathcal{O}\left(\log t\right)$. This allows it to scale to very large data streams. Nevertheless, if one strictly requires constant time and memory, one can simply limit the number of windows at the expense of detecting changes only up to a certain time in the past. In this configuration, MMDEW fulfills the requirements for streaming algorithms laid out by Domingos & Hulten (2003).

5 Experiments

This section showcases our approach on synthetic data (Section 5.1) and on streams derived from real-world classification tasks (Section 5.2). We ran all experiments on a server running Ubuntu 20.04 with 124GB RAM, and 32 cores with 2GHz each.

5.1 Synthetic data

To evaluate the average run length (ARL) and the mean time to detection (MTD) in a controlled environment, we first conduct experiments on synthetic data. We also compare the runtime of MMDEW to that of existing change detectors.

ARL and MTD. The ARL quantifies the false positive rate of MMDEW, that is, H_0 holds, and we want to know how often the change detector reports a change when no change happened. In the static setting, this corresponds to the type I error.

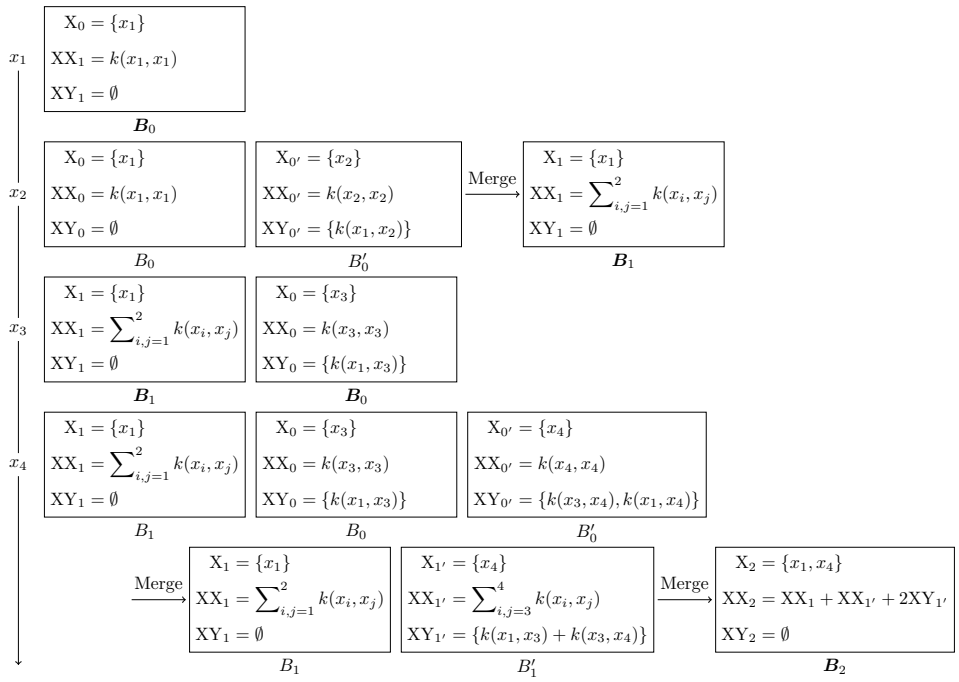

Figure 2: Set up of data structure with subsampling upon inserting $x_1, \ldots, x_4$. MMDEW stores the windows in bold face at the end of the merge operations. Observations x_2 and x_3 are not stored explicitly due to the sampling applied. x_4 is split into two lines for readability.

The error under the alternative (H_1 holds) is captured by the expected detection delay (EDD), also called "mean time to detection (MTD)". Specifically, MMDEW processes a stream that contains a change at a known observation and we want to know the delay until the change is reported, that is, how many samples of the post-change distribution need to be processed. In a static setting, this corresponds to the type II error.

We simulate 5-dimensional data distributed according to the multivariate normal $\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$, the uniform $\mathcal{U}[-\sigma\mathbf{1}_5, \sigma\mathbf{1}_5]$, the Laplace $(0, \sigma\mathbf{I}_5)$, and a mixed distribution. The mixed distribution is taken to be $\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$ with probability 0.3 and $\mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}_5\right)$ with probability 0.7, where $\mathbf{1}_d$ denotes a vector of d ones and $\mathbf{I}_d$ is the d-dimensional identity matrix. For all distributions, we set $\sigma = 3$.

To compute ARL, we consider 10,000 observations distributed according to either the uniform, the Laplace, or the mixed distribution. Hence, the data does not contain any changes. For MTD, we run our algorithm on 512 ($= 2^9$) observations, leading to MMDEW summarizing the data in one window. The observations are distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$ and then followed by either the uniform, Laplace, or mixed distribution. That is, we induce a change point, and then count the number of observations processed from the new distribution until MMDEW reports a change.

Figure 3 collects the results. The left plot shows that an increase in the level $\alpha \in (0, 1)$ leads to a decrease in ARL. This is expected as the test becomes more sensitive, leading to more false positives. The MTD plot on the right mirrors this observation: The MTD decreases with increasing α. We further observe that the detection delay depends on the post-change distribution. The delay is comparably large when changing from the multivariate standard normal to the mixed distribution. This matches our intuition: the mixed distribution is relatively similar to the pre-change distribution, rendering it difficult to detect a change between them. Overall, the results on these synthetic streams indicate that MMDEW is (i) robust to the choice of α and (ii) that α has the expected influence on the behavior of the algorithm.

Runtime. We now compare the runtime of MMDEW to that of its contenders and additionally validate the runtime guarantees that we derived analytically in Section 4.3.

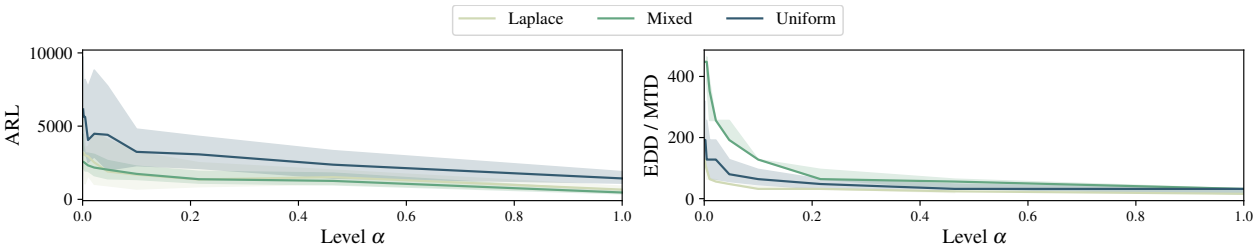

Figure 3: Average run length (ARL) and expected detection delay / mean time to detection (EDD / MTD) of MMDEW on synthetically generated data.

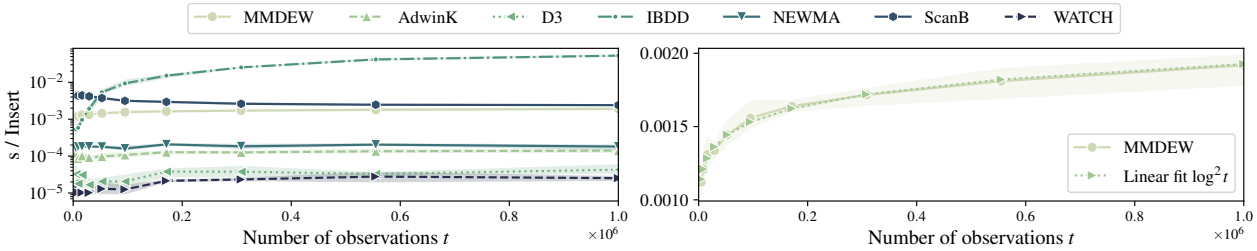

Figure 4: Comparison of runtimes per insert operation (l.h.s.) and least squares fit validating the theoretical runtime complexity of MMDEW w.r.t. the runtime observed in practice (r.h.s.).

To this end, we generate a constant stream of 10^6 one-dimensional observations, that is, the observed stream contains no change. Note that, while the dimensionality of the data affects the runtime depending on the used kernel, its influence is the same across all kernel-based algorithms, hence we limit our considerations to the univariate case.

Figure 4 shows the results. The left plot reveals that the fixed cost per insert of MMDEW is relatively large, as processing a small number of observations requires comparably much time. However, the runtime does not increase by much with the number of observations. The figure also shows that the proposed algorithm's runtime is better than that of an alternate kernel-based method, Scan B-statistics, where we use a window size of $\omega = 100$ in the runtime experiments. For $t > 0.05 \cdot 10^6$, MMDEW also outperforms IBDD. Still, the other algorithms run faster than MMDEW but achieve a lower F_1 score in our later experiments.

The right plot of Figure 4 verifies the analytically derived runtime of $\mathcal{O}\left(\log^2 t\right)$ by fitting the corresponding curve $(t \mapsto c \log^2 t)$ to the measured data with the least squares method. The resulting mean squared error is approximately 10^{-6}, which confirms the preceding asymptotic runtime analysis.

5.2 Real-world classification data

To obtain our change detection quality estimates, we use well-known classification data sets and interpret them as streaming data.[4] This is common in the literature, for example, (Faithfull et al., 2019; Faber et al., 2021), as only few high-dimensional annotated change detection data sets are publicly available.

For each data set, we first order the observations by their classes; a change occurs if the class changes. To introduce variation into the order of change points, we randomly permute the order of the classes before each run but use the same permutation across all algorithms. For preprocessing, we apply min-max scaling to all data sets. Table 2 summarizes the data sets, where n is the number of observations, d is the data dimensionality, and #CP is the number of change points.

We run a grid parameter optimization per data set and algorithm and report the best result w.r.t. the F_1-score. We note that such an optimization is difficult to perform in practice—here one typically prefers

[4]While MMDEW is not limited to Euclidean data, Euclidean data is the type of data most frequently encountered in practice, and our experiments target at this setting.

Table 2: Overview of data sets.

Data set	n	d	#CPs
CIFAR10 (Krizhevsky et al., 2009)	60,000	1,024	9
FashionMNIST (Xiao et al., 2017)	70,000	784	9
Gas (Vergara et al., 2012)	13,910	128	5
HAR (Anguita et al., 2013)	10,299	561	5
MNIST (Deng, 2012)	70,000	784	9

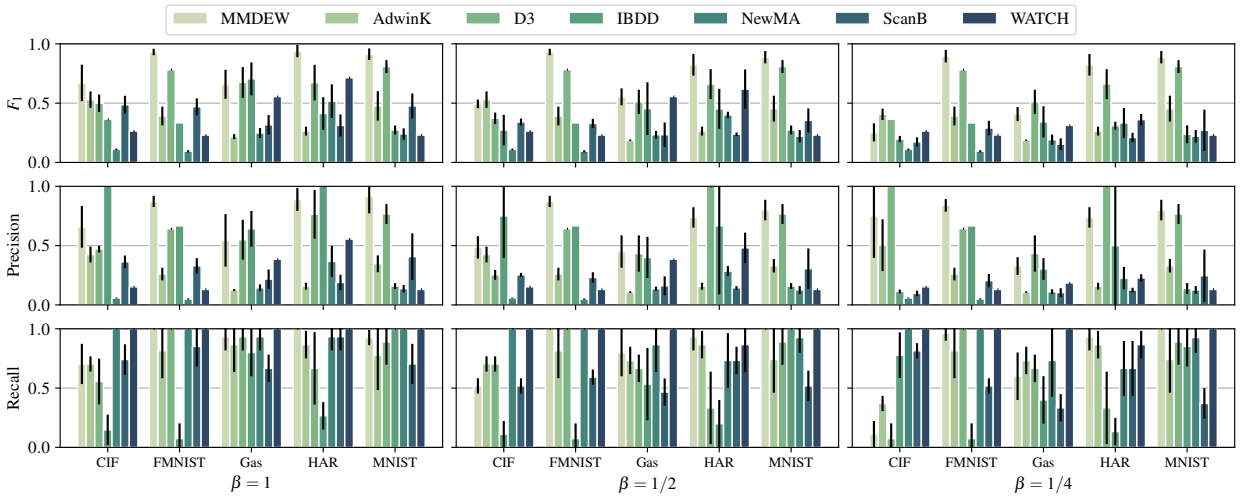

Figure 5: Average F_1-score, precision and recall. The bars show the standard deviation over 10 permutations of the data.

approaches with fewer or easy-to-set parameters—but allows a fair comparison. Table 3 in Appendix C lists all the parameters we tested. We note that the grid parameter optimization allowed us to obtain better F_1-scores than the heuristics proposed in Keriven et al. (2020) for NEWMA and Scan B-statistics.

We exclude the squared time estimator of MMD due to its prohibitive runtime. For kernel-based algorithms (MMDEW, NEWMA, and Scan B-statistics) we use the Gaussian kernel $k(x, y) = \exp\left(-\gamma\|x - y\|^2\right)$ $(\gamma > 0)$ and set γ using the median heuristic (Garreau et al., 2018) on the first 100 observations. The Gaussian kernel is universal (Steinwart & Christmann, 2008; Szabó & Sriperumbudur, 2017) and allows, given enough data, to detect any change in distribution as a universal kernel on a compact domain is characteristic (Gretton et al., 2012, Theorem 5). We also supply the first 100 observations to competitors requiring data to estimate further parameters (IBDD, WATCH) upfront.

F_1-score, precision, and recall. We compute the precision, the recall, and the F_1-score, which are common to evaluate change detection algorithms (Li et al., 2019; Keriven et al., 2020; van den Burg & Williams, 2020; Faber et al., 2021). Specifically, for a fixed $\Delta_T \in \mathbb{N}_{>0}$, we proceed as follows. If a change is detected, and there is an actual change point within the Δ_T previous time steps, we consider it a true positive (tp). If a change is detected, and there is no change point within the Δ_T previous steps, we consider it a false positive (fp). If no change is detected within Δ_T steps of a change point, we consider it a false negative (fn). We count at most one true positive for each actual change point. With these definitions, the precision is $\mathrm{Prec} = \mathrm{tp}/(\mathrm{tp} + \mathrm{fp})$, the recall is $\mathrm{Rec} = \mathrm{tp}/(\mathrm{tp} + \mathrm{fn})$, and the F_1-score is their harmonic mean $F_1 = 2 \cdot (\mathrm{Prec} \cdot \mathrm{Rec}) \,/\, (\mathrm{Prec} + \mathrm{Rec})$. Note that, while some algorithms allow to infer where in the data a change happens, including the proposed MMDEW, we only evaluate the time at which they report a change, as all tested approaches allow reporting this value.

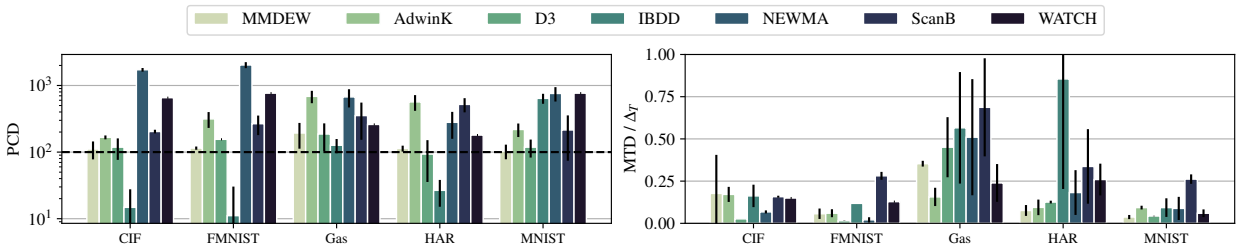

Figure 6: Average of percentage of changes detected (PCD) and of mean time to detection (MTD). The dashed line indicates the optimum for PCD. For MTD lower values are better.

Figure 5 shows our results. As Δ_T is an evaluation-specific parameter, we vary it relative to the average distance between change points by a factor $\beta > 0$: Given a data set of length N with n changes, we set $\Delta_T = \beta \cdot N/(n+1)$. For $\beta = 1$ (Δ_T is equal to the average number of steps between change points per respective data set), MMDEW achieves a higher F_1-score than all competitors on all data sets except for Gas, where it still obtains a competitive result. Throughout, the proposed algorithm obtains a good balance between precision and recall. Other approaches either have very low precision (for example, less than 20%), or an inferior recall and precision, down to a few exceptions. With a reduced β, that is, we allow only a shorter detection delay, the performance of all algorithms decreases on average. For $\beta = 1/2$, MMDEW achieves the best F_1 score also on four data sets, and, for $\beta = 1/4$ (the most challenging setting) on three of the tested data sets.

We conclude that the proposed method achieves very good results across all these experiments—especially when taking into account the fewer hyperparameters compared to the other approaches that we tested.

Percentage of changes detected and detection delay. To obtain a complete picture of the performance of MMDEW, we also report the "percentage of changes detected" (PCD), that is, the ratio of the number of reported changes and the number of actual change points, and its MTD on the data streams derived from real-world data. In our context, MTD coincides with the expected detection delay.

Figure 6 collects our results. For PCD, results closer to 100% are better. Here, MMDEW is on par with the closest competitors and consistently, that is, across all data sets, detects an approximately correct number of change points. D3, NEWMA, Scan B-statistics, and WATCH detect too many change points in all cases. This behavior is also reflected in their comparably large recall in Figure 5.

For MTD, lower values are better. Here, the classification-based D3 performs best in most of the cases. MMDEW performs a bit worse than D3 but better than the other algorithms on most data sets, with the Gas data set the major exception. As the experiments in Figure 5 show, a lower Δ_T tends to lead to a lower F_1-score of MMDEW. In other words, MMDEW tends to detect changes with some delay, but it detects them consistently.

6 Conclusions

We introduced a novel change detection algorithm, MMDEW, that builds upon two-sample testing with MMD, which is known to yield powerful tests on many domains. To facilitate the efficient computation of MMD, we presented a new data structure, which allows to estimate MMD with polylogarithmic runtime and logarithmic memory complexity. Our experiments on standard benchmark data show that MMDEW obtains the best F_1-score on most data sets. At the same time, MMDEW only has two parameters—the level of the statistical test and the choice of kernel. This simplifies the proposed algorithm's application in real-world use cases.

References

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *European Symposium on Artificial Neural Networks (ESANN)*, 2013.

Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Kluwer, 2004.

Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining (SDM)*, pp. 443–448, 2007.

Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Riec. Graph kernels: State-of-the-art and future challenges. *Foundations and Trends in Machine Learning*, 13(5-6):531–712, 2020.

George Casella and Roger L. Berger. *Statistical inference*. Wadsworth & Brooks/Cole, 1990.

Xiuyuan Cheng and Yao Xie. Kernel two-sample tests for manifold data. *Bernoulli*, 30(4):2572–2597, 2024.

Marco Cuturi. Fast global alignment kernels. In *International Conference on Machine Learning (ICML)*, pp. 929–936, 2011.

Marco Cuturi and Jean-Philippe Vert. The context-tree kernel for strings. *Neural Networks*, 18(8):1111–1123, 2005.

Tamraparni Dasu, Shankar Krishnan, Dongyu Lin, Suresh Venkatasubramanian, and Kevin Yi. Change (detection) you can believe in: Finding distributional shifts in data streams. In *International Symposium on Intelligent Data Analysis (IDA)*, volume 5772, pp. 21–34, 2009.

Vinícius M. A. de Souza, Antonio Rafael Sabino Parmezan, Farhan Asif Chowdhury, and Abdullah Mueen. Efficient unsupervised drift detector for fast and high-dimensional data streams. *Knowledge and Information Systems*, 63(6):1497–1527, 2021.

Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, pp. 141–142, 2012.

Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

Kamil Faber, Roberto Corizzo, Bartlomiej Sniezynski, Michael Baron, and Nathalie Japkowicz. WATCH: Wasserstein change point detection for high-dimensional time series data. In *IEEE International Conference on Big Data*, pp. 4450–4459, 2021.

William J. Faithfull, Juan José Rodríguez Diez, and Ludmila I. Kuncheva. Combining univariate approaches for ensemble change detection in multivariate data. *Information Fusion*, 45:202–214, 2019.

Kenji Fukumizu, Arthur Gretton, Xiaohai Sun, and Bernhard Schölkopf. Kernel measures of conditional dependence. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 498–496, 2008.

João Gama. *Knowledge discovery from data streams*. CRC Press, 2010.

Damien Garreau, Wittawat Jitkrittum, and Motonobu Kanagawa. Large sample analysis of the median heuristic. Technical report, 2018. `https://arxiv.org/abs/1707.07269`.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. *Computational Learning Theory and Kernel Machines (COLT)*, 2777:129–143, 2003.

Ömer Gözüaçik, Alican Büyükçakir, Hamed R. Bonab, and Fazli Can. Unsupervised concept drift detection with a discriminative classifier. In *International Conference on Information and Knowledge Management (CIKM)*, pp. 2365–2368, 2019.

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.

Zaïd Harchaoui and Olivier Cappé. Retrospective multiple change-point estimation with kernels. In *IEEE/SP Workshop on Statistical Signal Processing*, pp. 768–772, 2007.

Nicolas Keriven, Damien Garreau, and Iacopo Poli. NEWMA: A new method for scalable model-free online change-point detection. *IEEE Transactions on Signal Processing*, 68:3515–3528, 2020.

Franz J. Király and Harald Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 20:1–45, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009. `https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf`.

Shuang Li, Yao Xie, Hanjun Dai, and Le Song. Scan B-statistic for kernel change-point detection. *Sequential Analysis*, 38(4):503–544, 2019.

Quentin Mérigot. A multiscale approach to optimal transport. In *Computer Graphics Forum*, volume 30, pp. 1583–1592, 2011.

Krikamol Muandet, Kenji Fukumizu, Bharath K. Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10(1-2): 1–141, 2017.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1177–1184, 2007.

Aaditya Ramdas, Sashank Jakkam Reddi, Barnabás Póczos, Aarti Singh, and Larry A. Wasserman. On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions. In *Conference on Artificial Intelligence (AAAI)*, pp. 3571–3577, 2015.

Michael Reed and Barry Simon. *Methods of modern mathematical physics. I. Functional analysis*. Academic Press, 1972.

David W Scott. Feasibility of multivariate density estimates. *Biometrika*, 78(1):197–205, 1991.

Dino Sejdinovic, Bharath Sriperumbudur, Arthur Gretton, and Kenji Fukumizu. Equivalence of distance-based and RKHS-based statistics in hypothesis testing. *Annals of Statistics*, 41:2263–2291, 2013.

Neil James Alexander Sloane. Entry A001788 in The On-Line Encyclopedia of Integer Sequences, 1999a. `https://oeis.org/A001788`.

Neil James Alexander Sloane. Entry A036289 in The On-Line Encyclopedia of Integer Sequences, 1999b. `https://oeis.org/A036289`.

Alexander Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A Hilbert space embedding for distributions. In *Algorithmic Learning Theory (ALT)*, volume 4754, pp. 13–31, 2007.

Bharath Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert Lanckriet. Hilbert space embeddings and metrics on probability measures. *Journal of Machine Learning Research*, 11:1517–1561, 2010.

Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer, 2008.

Zoltán Szabó and Bharath K. Sriperumbudur. Characteristic and universal tensor product kernels. *Journal of Machine Learning Research*, pp. 233:1–233:29, 2017.

Gábor Székely and Maria Rizzo. Testing for equal distributions in high dimension. *InterStat*, 5:1249–1272, 2004.

Gábor Székely and Maria Rizzo. A new test for multivariate normality. *Journal of Multivariate Analysis*, 93:58–80, 2005.

Gerrit J. J. van den Burg and Christopher K. I. Williams. An evaluation of change point detection algorithms. Technical report, 2020. `https://arxiv.org/abs/2003.06222`.

Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A Ryan, Margie L Homer, and Ramón Huerta. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166:320–329, 2012.

Roman Vershynin. *High-dimensional probability*. Cambridge University Press, 2018.

Chris Watkins. Dynamic alignment kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 39–50, 1999.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. Technical report, 2017. `https://arxiv.org/abs/1708.07747`.

Wojciech Zaremba, Arthur Gretton, and Matthew B. Blaschko. B-test: A non-parametric, low variance kernel two-sample test. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 755–763, 2013.

A Proofs

This section contains additional proofs. The proof of Proposition 1 is in Section A.1. Proposition 3 is proved in Section A.2.

A.1 Proof of Proposition 1

Proposition 1 follows from the more general result that we state below. The statement and proof are similar to Gretton et al. (2012, Theorem 8) but do not assume $m = n$. Note that we recover Gretton et al. (2012, Theorem 8) in the case that $m = n$. We prove Proposition 1 afterwards.

Proposition 4. *Let $\mathbb{P}$, $\mathbb{Q}$, $\hat{\mathbb{P}}_m$, $\hat{\mathbb{Q}}_n$ be defined as in the main text, assume $0 \leq k(x, y) \leq K$ for all $x, y \in \mathcal{X}$, $\mathbb{P} = \mathbb{Q}$, and $t > 0$. Then*

$$P\left(\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) - \left(\frac{K}{m} + \frac{K}{n}\right)^{\frac{1}{2}} \geq t\right) \leq e^{-\frac{t^2 mn}{2K(m+n)}}.$$

Proof. First, we bound the difference of $\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right)$ to its expected value. Changing a single one of either x_i or y_j in this function results in changes of at most $2\sqrt{K}/m$, and $2\sqrt{K}/n$, giving

$$\sum_{i=1}^{n+m} c_i^2 = 4K\frac{n+m}{nm}.$$

We now apply the bounded differences inequality (recalled in Theorem 1) to obtain

$$P\Big(\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) - \mathbb{E}\,\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) \geq t\Big) \leq e^{-\frac{t^2 mn}{2K(m+n)}}.$$

The last step is to bound the expectation, which yields

$$\begin{aligned}
\mathbb{E}\,\mathrm{MMD}\left(\hat{\mathbb{P}}_m, \hat{\mathbb{Q}}_n\right) &= \mathbb{E}\left(\frac{1}{m^2}\sum_{i,j=1}^{m} k(x_i, x_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(y_i, y_j) - \frac{1}{mn}\sum_{i,j=1}^{m,n} k(x_i, y_j) - \frac{1}{mn}\sum_{j,i=1}^{n,m} k(y_j, x_i)\right)^{\frac{1}{2}} \\
&\leq \left(\frac{1}{m}\mathbb{E}k(X,X) + \frac{1}{n}\mathbb{E}k(Y,Y) + \frac{1}{m}(m-1)\mathbb{E}k(X,Y) + \frac{1}{n}(n-1)\mathbb{E}k(Y,X) - 2\mathbb{E}k(X,Y)\right)^{\frac{1}{2}} \\
&= \left(\frac{1}{m}\mathbb{E}k(X,X) + \frac{1}{n}\mathbb{E}k(Y,Y) - \frac{1}{m}\mathbb{E}k(X,Y) - \frac{1}{n}\mathbb{E}k(X,Y)\right)^{\frac{1}{2}} \\
&= \left(\frac{1}{m}\mathbb{E}\left[k(X,X) - k(X,Y)\right] + \frac{1}{n}\mathbb{E}\left[k(X,X) - k(X,Y)\right]\right)^{\frac{1}{2}} \leq \left(\frac{K}{m} + \frac{K}{n}\right)^{\frac{1}{2}}.
\end{aligned}$$

Inserting this into the previous inequality, we obtain the stated result. □

Proposition 1 is now a corollary of Proposition 4, which follows by setting $\alpha = e^{-\frac{t^2 mn}{2K(m+n)}}$ and solving for t to obtain a test of level α.

A.2 Proof of Proposition 3

To find $n_{\mathrm{XY}_l^l}$, we use our implementation of MMDEW and the On-Line Encyclopedia of Integer Sequences (OEIS) to discover that $n_{\mathrm{XY}_l^l}$ follows the sequence $1, 2, 8, 24, 64, 160, \ldots$ for $l = 0, 1, 2, \ldots$. Thus

$$n_{\mathrm{XY}_l^l} = 2^l l, \quad \text{for } l > 0 \tag{8}$$

and $n_{\mathrm{XY}_0^0} = 1$ (Sloane, 1999b).

To find n_{XX_l}, notice that n_{XX_l} only changes when one merges two windows, which happens for windows of the same size $n_{\mathrm{XX}_{l-1}}$. The algorithm adds to this $2 \cdot n_{\mathrm{XY}_{l-1}^{l-1}}$ terms, see (6), and, for $l = 0, 1, 2, \ldots$, we obtain the recurrence relation

$$n_{\mathrm{XX}_l} = \begin{cases} 1 & \text{if } l = 0, \\ 4 & \text{if } l = 1, \\ 2 \cdot n_{\mathrm{XX}_{l-1}} + 2 \cdot n_{\mathrm{XY}_{l-1}^{l-1}} & \text{if } l > 1, \end{cases}$$

with $n_{\mathrm{XX}_{-1}} := 0$. Now write

$$n_{\mathrm{XX}_l} = 2 \cdot n_{\mathrm{XX}_{l-1}} + l \cdot 2^l - 2^l + 2 \cdot [l = 0] + 2 \cdot [l = 1], \tag{9}$$

where the brackets are equal to one if their argument is true and zero otherwise (using Iverson's convention; Graham et al. 1994). To find a closed-form expression for (9), we define the ordinary generating function $A(z) = \sum_l a_l z^l$. Now, we multiply (9) by z_l and sum on l, to obtain

$$A(z) = \frac{-8z^3 + 2z - 1}{(2z-1)^3}$$

after some algebra, so that

$$n_{\mathrm{XX}_l} = [z^l]\frac{-8z^3 + 2z - 1}{(2z-1)^3},$$

where $[z^l]$ is the coefficient of z^l in the series expansion of the generating function $A(z)$. To extract coefficients, we first decompose $A(z)$ as

$$A(z) = \frac{3}{1-2z} - \frac{2}{(1-2z)^2} + \frac{1}{(1-2z)^3} - 1,$$

which allows us to then find the coefficients as

$$[z^l]\frac{3}{1-2z} \overset{(a)}{=} 3 \cdot 2^l, \qquad [z^l] - \frac{2}{(2z-1)^2} \overset{(b)}{=} -(l+1)2^{l+1}, \qquad [z^l]\frac{1}{(1-2z)^3} \overset{(c)}{=} (l+1)(l+2)2^{l-1},$$

where Graham et al. (1994, Table 335) implies (a), (b) is (8) shifted, and (c) is Sloane (1999a) shifted. We omit the last term as it corresponds to $[z^0]$, which we do not need. Now, adding all terms gives $3 \cdot 2^l - (l+1)2^{l+1} + (l+1)(l+2)2^{l-1} = 2^{l-1}(l^2 - l + 4)$, concluding the proof.

B External results

To proof Proposition 1, we recall McDiarmid's concentration inequality (Vershynin, 2018).

Theorem 1 (Bounded differences inequality)**.** *Let $X = (X_1, \dots, X_n)$ be a random vector with independent components. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a measurable function. Assume that the value of $f(x)$ can change by at most $c_i > 0$ under an arbitrary change of a single coordinate of $x = (c_1, \dots, c_n) \in \mathbb{R}^n$. Then, for any $t > 0$, we have*

$$P\{f(X) - \mathbb{E}f(X) \geq t\} \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^n c_i^2}\right).$$

C Hyperparameter optimization settings

We collect the hyperparameter choices that we tested in our experiments on real-world classification data (Section 5.2) in Table 3 and refer to the respective original publications for additional information on the parameter settings.

Table 3: Values chosen for the parameter optimization.

Algorithm	Parameters	Parameter values
MMDEW	α	$\alpha \in \{0.001, 0.01, 0.1, 0.2\}$
ADWINK	δ, k	$\delta \in \{0.05, 0.1, 0.2, 0.9, 0.99\}$, $k \in \{0.01, 0.02, 0.05, 0.1, 0.2\}$
D3	$\omega, \rho, \tau, \mathrm{d}$	$\omega \in \{100, 200, 500\}$, $\rho \in \{0.1, 0.3, 0.5\}$, $\tau \in \{0.7, 0.8, 0.9\}$, $\mathrm{d} = 1$
IBDD	m, w	$m \in \{10, 20, 50, 100\}$, $w \in \{20, 100, 200, 300\}$
NEWMA	ω, α	$\omega \in \{20, 50, 100\}$, $\alpha \in \{0.01, 0.02, 0.05, 0.1\}$
Scan B	B, ω, α	$B \in \{2, 3\}$, $\omega \in \{100, 200, 300\}$, $\alpha \in \{0.01, 0.05\}$
WATCH	$\epsilon, \kappa, \mu, \omega$	$\epsilon \in \{1, 2, 3\}$, $\kappa \in \{25, 50, 100\}$, $\mu \in \{10, 20, 50, 100, 1000, 2000\}$, $\omega \in \{100, 250, 500, 1000\}$