Mach Learn (2010) 79: 283–306
DOI 10.1007/s10994-009-5122-x

Fast adaptive algorithms for abrupt change detection

Daniel Nikovski · Ankur Jain

Received: 19 December 2007 / Revised: 22 May 2009 / Accepted: 22 May 2009 /
Published online: 16 July 2009
Springer Science+Business Media, LLC 2009

Abstract We propose two fast algorithms for abrupt change detection in streaming data that can operate on arbitrary unknown data distributions before and after the change. The first algorithm, MB-GT, computes efficiently the average Euclidean distance between all pairs of data points before and after the hypothesized change. The second algorithm, MB-CUSUM, computes the log-likelihood ratio statistic for the data distributions before and after the change, similarly to the classical CUSUM algorithm, but unlike that algorithm, MB-CUSUM does not need to know the exact distributions, and uses kernel density estimates instead. Although a straightforward computation of the two change statistics would have computational complexity of $O(N^4)$ with respect to the size N of the streaming data buffer, the proposed algorithms are able to use the computational structure of these statistics to achieve a computational complexity of only $O(N^2)$ and memory requirement of $O(N)$. Furthermore, the algorithms perform surprisingly well on dependent observations generated by underlying dynamical systems, unlike traditional change detection algorithms.

Keywords Event detection · Distribution monitoring · CUSUM

1 Introduction

One of the main events that is often useful to detect in a sensor data stream is an abrupt change in the nature of the streaming data. For example, the temperature of an industrial process might depart from its normal values, and this might signal that the process is out of control. However, at other times the temperature of the process might vary due to random noise, for example caused by measurement error or unmodeled variables, without necessarily being out of control. Distinguishing between these two situations is often a challenging problem, and the field of statistical process control (SPC) is concerned with devising methods and algorithms for detecting such changes.

Editor: Weng-Keen Wong.

D. Nikovski (✉) · A. Jain
Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA 02139, USA
e-mail: nikovski@merl.com

From a mathematical point of view, the problem reduces to detecting a departure from the in-control distribution of the data towards some other, out-of-control distribution. For in-control and out-of-control distributions of known parametric form and known distribution parameters, the CUSUM algorithm originally due to Page (1954) has been used with much success. Moreover, it has been proven that it is optimal, i.e., no other algorithm guarantees faster change detection for a pre-specified probability of false alarm (Basseville and Nikiforov 1993). However, the in-control and especially all possible out-of-control distributions can very rarely be modeled explicitly. This necessitates the use of methods that can detect abrupt changes only by inspecting the data streams and reasoning about the probability distributions implied by the data readings themselves. In this paper, we are proposing algorithms based on memory-based machine learning methods for quick estimation of probability distributions from data.

Our change-detection framework assumes that there is only a *single* change in the entire length of the stream, and this change persists for the remainder of the data stream. This is the typical situation when the change is destructive (e.g., a burnout, motor failure, etc.)

The change point is detected based on a quantitative *figure of merit* produced by the change detection algorithm. The figure of merit serves as an estimate of the difference between the two data distributions on the two sides of the change point. For example, in Page's CUSUM algorithm (Page 1954), the figure of merit is the log-likelihood ratio corresponding to a specific change point. Other popular measures are distances between distributions, e.g. Kullback-Leibler divergence, Rényi divergence, etc. (Guha et al. 2006). Each of these figures of merit has its computational advantages and disadvantages; for example, the mentioned distances between distributions all suffer from the need to compute multiple integrals over the entire domain of the data collected from the data stream.

In contrast, the focus of our work has been to develop a general and *parameter-free* framework that offers scalable performance and operates in limited memory and in real time. Since a sensor stream can potentially be unbounded in size, and any processing machine has limited computational and memory resources, our algorithms use a sliding window over the data stream. The length of the sliding window is the amount of historical data that an algorithm stores and considers during the computation of the figure of merit. If we denote a d-dimensional data vector from the sensor stream at time instant t as $\mathbf{x}_t$, and N is the length of the sliding window, then a change-detection algorithm considers only the data points $\{\mathbf{x}_{t-N+1}, \dots, \mathbf{x}_t\}$ to answer the following questions: "Given the data seen so far, what is the likelihood that a change has occurred on or before time instant t? If a change has occurred, when was the most likely time it occurred?" In the rest of the discussion, we denote the sliding window at time t by Γ^t. Since we use only the last N elements seen in the stream, we omit the global time reference notation within the window, and refer to the elements of the sliding window as $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ where $\mathbf{x}_N$ is the latest element obtained from the stream.

As new data become available, the sliding window slides by one element, discarding the oldest data element and incorporating the latest one. Each time a sliding operation is made, a change-detection algorithm α searches for a pair (i, j) that splits Γ^t into two *sub-windows* $\gamma^t_{i,j-1}$ and $\gamma^t_{j,N}$ (as shown in Fig. 1) such that

$$\gamma^t_{p,q} = \{\mathbf{x}_p, \dots, \mathbf{x}_q\}, \quad \text{where } \Gamma^t = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \wedge (1 \le i < j \le N). \tag{1}$$

This split is made in such a way that point j has the highest likelihood of being a change point, if one did occur in Γ^t. Note that the size of the two sub-windows may or may not be the same.

Under the operation of algorithm α, the possibility of the occurrence of a change for any time instant t is quantified by the value of the figure of merit Υ^α_t, which is a measure of

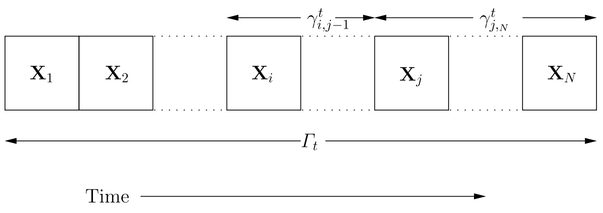


Fig. 1 An instance of a sliding window and its probable sub-windows

the difference between the data distributions contained in the sub-windows $\gamma^t_{i,j-1}$ and $\gamma^t_{j,N}$. Different algorithms compute figures of merit in different ways, which affects their ability to detect the true change point accurately.

Since the search for the change point is made over all possible sub-windows within Γ^t, any change-detection algorithm Υ has a time complexity of at least $O(N^2)$. We designed our change detection algorithms with the following considerations in mind:

1. Generality, i.e. they do not make any assumptions about the shape and parameters of the underlying data distributions.
2. Scalability, i.e. they offer time complexity comparable to that of $O(N^2)$.
3. Operation in limited memory.

We propose the following two memory-based abrupt change detection algorithms:

– **Memory Based Graph Theoretic (MB-GT)** This algorithm approaches the problem from a graph-theoretic perspective. Qualitatively, the figure of merit ($\Upsilon_t^{\text{MB-GT}}$) is based upon the spatial (Euclidean) distance between pairs of data elements from different sub-windows. It can also be perceived as a memory-based clustering approach where the objective is to minimize the intra-cluster distance while maximizing the inter-cluster distances. We present more details of this algorithm in Sect. 2.
– **Memory Based CUmulative SUM (MB-CUSUM)** This algorithm is inspired by the classic CUSUM algorithm. It iteratively computes the likelihood of a data element coming from the two distributions in the sub-windows and uses the cumulative likelihood ratio as its figure of merit ($\Upsilon_t^{\text{MB-CUSUM}}$). The details of this algorithm are presented in Sect. 3

2 A memory-based graph theoretic algorithm

One straightforward solution to the problem of computing the distance between two distributions indirectly specified by means of two sample sets is to compute the average distance between the samples themselves. Since each sample is a point in a multi-dimensional Euclidean space, a natural distance measure between pairs of points $\mathbf{x}_k$ and $\mathbf{x}_l$ is their Euclidean distance $d_{k,l} \doteq \|\mathbf{x}_k - \mathbf{x}_l\|$. For a particular split defined by the index pair (i, j) specified in Fig. 1, we can compute the average distance between the two sub-windows as

$$C_{i,j} = \frac{\sum_{k=i}^{j-1} \sum_{l=j}^{N} d_{k,l}}{(j-i)(N-j+1)}. \tag{2}$$

We will call the corresponding change detection algorithm MB-GT (Memory-Based Graph Theoretic). Its overall figure of merit $\Upsilon_t^{\text{MB-GT}}$ can be computed as described above: $\Upsilon_t^{\text{MB-GT}} = \max_{1\le i<j\le N} C_{i,j}$. Since computing each $C_{i,j}$ is of complexity $O(N^2)$, and there are $O(N^2)$ such terms to be considered, the overall complexity of computing $\Upsilon_t^{\text{MB-GT}}$ seems

to be $O(N^4)$, if implemented directly. This complexity is unacceptable for practical applications.

However, the computation of individual $C_{i,j}$ terms has certain redundancy and repetitive structure that can be exploited to bring the computational complexity of MB-GT back down to $O(N^2)$. If we define

$$C'_{i,j} \doteq \sum_{k=i}^{j-1}\sum_{l=j}^{N} d_{k,l}, \qquad \beta_{i,j} \doteq \sum_{l=j}^{N} d_{i,l}, \tag{3}$$

one can verify that the following recurrent relationships hold: $\beta_{i,j-1} = \beta_{i,j} + d_{i,j-1}$, with $\beta_{i,N+1} = 0$, and $C'_{i-1,j} = C'_{i,j} + \beta_{i-1,j}$, with $C'_{j,j} = 0$ for all $1 \le j \le N$. These recurrences suggest the following efficient computational algorithm. If the values $C'_{i,j}$ are placed in a tableau that is conceptually similar to a matrix, this matrix would be upper triangular due to the constraint $i < j$. Computation can start with the bottom row of this matrix that has a single element $C'_{N,N}$ which is zero by definition. For each row $1 \le i < N$ above the last one, proceeding from bottom to top, the following two steps are performed:

1. All values $\beta_{i,j}$ are computed recurrently from their immediate neighbor to the right, proceeding right to left, and using the recurrence $\beta_{i,j-1} = \beta_{i,j} + d_{i,j-1}$.
2. All values $C'_{i,j}$ are computed from the respective values $\beta_{i,j}$ and the values $C'_{i+1,j}$ in the row immediately below the current one, using the recurrence $C'_{i,j} = C'_{i+1,j} + \beta_{i,j}$.

Computing $\Upsilon_t^{\text{MB-GT}}$ can be done simultaneously with the computation of the individual terms $C'_{i,j}$, since it involves only normalization and maximization. For this reason, it is *not* necessary to keep all values $\beta_{i,j}$ and $C'_{i,j}$ in memory; it suffices to keep a buffer of size N elements for the current row i of values $\beta_{i,j}$, and two buffers of the same size for $C'_{i,j}$ and $C'_{i+1,j}$. Thus, the memory requirement of this algorithm is only $O(N)$. The computation of $\beta_{i,j}$ and $C'_{i,j}$ for each row i is only of $O(N)$, and since there are N such rows, the overall computational complexity is only $O(N^2)$, as opposed to $O(N^4)$ for a naive implementation. Algorithm 1 shows in detail how the MB-GT algorithm computes its figure of merit given a sliding window of observation data.

3 A Memory-Based CUSUM Algorithm

Unlike MB-GT, the second algorithm we propose, MB-CUSUM, has probabilistic foundations identical to that of the original CUSUM algorithm, which potentially allows it to achieve optimal change detection under certain modeling conditions. At the same time, despite its very different theoretical foundation, MB-CUSUM has similar computational structure to MB-GT, and we will demonstrate how this structure can be leveraged to achieve the same significant improvements in computational complexity.

Following the derivation of CUSUM in (Basseville and Nikiforov 1993), we consider the following hypotheses about a possible change within the current buffer of N readings kept in memory:

$$\begin{aligned} \mathbf{H}_{i,0}:&\quad \mathbf{x}_k \sim p_0 \quad \text{for } i \le k \le N,\\ 1 \le i < j \le N, \mathbf{H}_{ij}:&\quad \mathbf{x}_k \sim p_0 \quad \text{for } i \le k \le j-1,\\ &\quad \mathbf{x}_l \sim p_1 \quad \text{for } j \le l \le N, \end{aligned}$$

where p_0 and p_1 are the data distributions before and after the change, respectively.

Algorithm 1 Compute $\Upsilon_t^{\text{MB-GT}}$

```
Inputs:
        Sliding window size:          $N$
        Sliding window :              $\Gamma^t = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
        Distance function :           $d(\mathbf{x}_1, \mathbf{x}_2)$ /* e.g., $d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|$ */
Local variables:
        Partial-sum vector:           $\beta = \{b[i] | 1 \le i \le N\}$
        Recurrent figure of merit $\mathcal{C}$:      /* Max. figure of merit for an $(i, j)$ pair */
Outputs:
        Figure of merit:              $\Upsilon_t^{\text{MB-GT}}$      /* Max. figure of merit for $\Gamma^t$ */
 1: for $i = 1$ to $N$ do
 2:    $\beta[i] = 0$                              /*Reset the array values to zero*/
 3: end for
 4: for $j = (N-1)$ to 2 do
 5:    $\mathcal{C} = 0$
 6:    for $i = (j-1)$ to 1 do
 7:       $\beta[i] = \beta[i] + d(\mathbf{x}_i, \mathbf{x}_j)$          /*Incrementally update the partial sums*/
 8:       $\mathcal{C} = \mathcal{C} + \beta[i]$
 9:       $\Upsilon_t^{\text{MB-GT}} = \max(\frac{\mathcal{C}}{(j-i)(N-j)}, \Upsilon_t^{\text{MB-GT}})$ /*Updating figure of merit*/
10:    end for
11: end for
12: return $\Upsilon_t^{\text{MB-GT}}$
```

Here we are considering the null hypotheses $\mathbf{H}_{i,0}$ that no change has occurred while the latest $N-i+1$ samples were collected, vs. multiple hypotheses $\mathbf{H}_{i,j}$ that such a change has occurred. Compared to CUSUM, by introducing the starting index i, we are expanding the set of hypotheses to be tested to those that do not necessarily use all N samples in the window. According to the Neyman-Pearson lemma, the most powerful test that we can perform when testing each particular hypothesis $\mathbf{H}_{i,j}$ vs. $\mathbf{H}_{i,0}$ (i.e., the test that has the highest probability of rejecting a false null hypothesis), is the likelihood ratio

$$\Lambda_{ij} = \frac{\prod_{k=i}^{j-1} p_0(\mathbf{x}_k) \cdot \prod_{l=j}^{N} p_1(\mathbf{x}_l)}{\prod_{k=i}^{N} p_0(\mathbf{x}_k)}. \tag{4}$$

For convenience, the log-likelihood ratio $S_{ij} = \log(\Lambda_{ij})$ is commonly used. In our algorithm, we replace the true *pdfs* p_0 and p_1 with their Parzen kernel density estimates (Hastie et al. 2001), as described by Eq. 5:

$$p(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} w(\mathbf{x} - \mathbf{x}_i), \tag{5}$$

where w is a suitably chosen kernel, and $\mathbf{x}_i$, $i = 1, n$ is a sample of data points from the distribution to be modeled. Popular choices for the kernel are Gaussian, tri-cubic, etc. (Hastie et al. 2001). This results in the following figure of merit for a particular split (i, j):

$$\mathcal{S}_{i,j} = \sum_{l=j}^{N} \log \frac{\frac{1}{N-j+1}\sum_{k=j}^{N} w_{l,k}}{\frac{1}{j-i}\sum_{k=i}^{j-1} w_{l,k}}, \quad w_{l,k} \doteq w(\mathbf{x}_l - \mathbf{x}_k). \tag{6}$$

Here $w_{l,k}$ is a kernel weight for the pair of samples $(\mathbf{x}_l, \mathbf{x}_k)$. By using the maximum likelihood principle, the figure of merit for this algorithm will be $\Upsilon_t^{\text{MB-CUSUM}} = \max_{1 \le i < j \le N} S_{i,j}$. Again, a direct computation of $\Upsilon_t^{\text{MB-CUSUM}}$ would have computational complexity of $O(N^4)$.

However, $\Upsilon_t^{\text{MB-CUSUM}}$ has a similar structure to $\Upsilon_t^{\text{MB-GT}}$ that can again be exploited to reduce its computational complexity. Again, we can conceptually organize the values of $S_{i,j}$ in a tableau, and define the following auxiliary variables:

$$\mu_j^l \doteq \sum_{k=j}^{N} w_{l,k}, \qquad \nu_{i,j}^l \doteq \sum_{k=i}^{j-1} w_{l,k}. \tag{7}$$

Although there appear to be $O(N^3)$ $\nu_{i,j}^l$ terms to be computed, the recurrent reformulation of Eq. 6

$$S_{i,j} = S_{i,j+1} + \log \mu_j^j - \log \nu_{i,j}^j + \log(j-i) - \log(N-j+1) \tag{8}$$

can convince us that not all of them are needed. By further defining $\mu_j' \doteq \mu_j^j$, and $\nu_{i,j}' \doteq \nu_{i,j}^j$, we can use the following equations as a basis for an efficient algorithm:

$$\mu_j' = \sum_{k=j}^{N} w_{j,k}, \qquad \nu_{i,j}' = \nu_{i+1,j}' + w_{j,i}. \tag{9}$$

Note that only the one for $\nu_{i,j}'$ happens to be recurrent; the other, for μ_j', is computed directly. These equations suggest the following efficient algorithm (described in detail in Algorithm 2):

S1: Compute μ_j' for $j = 1, N$ directly, per Eq. 9. This computation takes $O(N^2)$, but the results can be stored in $O(N)$ space.

S2: For each row $i = N, 1$ of the matrix $S_{i,j}$, starting from the bottom row $i = N$ and moving upwards to the first row $i = 1$, perform the following two steps:

- S2.1: For each value of j between $i+1$ and N, compute $\nu_{i,j}'$ from the corresponding $\nu_{i+1,j}'$ in the row below, and $w_{j,i}$, per Eq. 9.
- S2.2: For each value of j between N and $i+1$, compute $S_{i,j}$ from the value $S_{i,j+1}$ immediately to the right, using the equation $S_{i,j} = S_{i,j+1} + \log \mu_j' - \log \nu_{i,j}' + \log(j-i) - \log(N-j+1)$, starting with $S_{i,N+1} = 0$ for all $i = 1, N$. The computation in this step proceeds strictly right to left ($j = N, i+1$).

4 Other figures of merit

In addition to the novel schemes discussed so far, some other known methods can also be adapted such that their incremental versions can be applied to the change-detection problem. In order to provide a comparative analysis, we used the popular *Student's t-test* (MB-TSTAT) and the Kolmogorov-Smirnov (MB-KS, Massey 1951) procedures to verify the performance of our novel techniques. While we could not find an incremental version of the MB-KS method (an $\mathcal{O}(N^3 \log N)$ procedure), we realized that the MB-TSTAT procedure can naturally be extended using recurrent calculations. In the rest of this section, we provide the details of the incremental version of the MB-TSTAT method (details in Algorithm 2).

Algorithm 2 Compute $\Upsilon_t^{\text{MB-CUSUM}}$

```
Inputs:
        Sliding window size:   $N$
        Sliding window :       $\Gamma^t = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$
        Kernel function :      $w(\mathbf{z}) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\|\mathbf{z}\|^2/2\sigma^2}$
Local variables:
        Partial-sum vector:    $\upsilon = \{\upsilon[i] | 1 \le i \le N\}$
        Recurrent figure of merit $\mathcal{S}$:      /* Max. figure of merit for an $(i, j)$ pair */
Global variables:
        Partial-sum vector:    $\mu = \{\mu[i] | 1 \le i \le N\}$
        First run indicator:   init=true /* Changes state when the algorithm
                                         is called the first time */
Outputs:
        Figure of merit:       $\Upsilon_t^{\text{MB-CUSUM}}$ /* Max. figure of merit for $\Gamma^t$ */
 1: if init then
 2:   for $i = 1$ to $N$ do
 3:     $\mu[i] = 0$                                   /*Reset the $\mu$ vector*/
 4:   end for
 5:   init = false
 6: end if
 7: for $i = 1$ to $N - 1$ do
 8:   $\upsilon[i] = 0$                                /*Reset the $\upsilon$ vector*/
 9:   $\mu[i] = \mu[i+1] + w(\mathbf{x}_i - \mathbf{x}_N)$     /*Updating the $\mu$ vector*/
10: end for
11: $\upsilon[N] = 0$
12: $\mu[N] = w(\mathbf{0})$
13: for $i = (N - 2)$ to 1 do
14:   $\mathcal{S} = 0$
15:   for $j = (N - 1)$ to $i + 1$ do
16:     $\upsilon[i] = \upsilon[i] + w(\mathbf{x}_i - \mathbf{x}_j)$    /*Incrementally update the $\upsilon$ vector*/
17:     $\mathcal{S} = \mathcal{S} + \log\left(\frac{\mu[j](j-i)}{\upsilon[j](N-j)}\right)$
18:     $\Upsilon_t^{\text{MB-CUSUM}} = \max((\mathcal{S}, \Upsilon_t^{\text{MB-CUSUM}})$ /*Updating figure of merit*/
19:   end for
20: end for
21: return $\Upsilon_t^{\text{MB-CUSUM}}$
```

4.1 Memory based t-statistic MB-TSTAT

Student's t-test quantifies the difference between two Gaussian distributions using the means and variances in the data. In the sliding-window mode of operation, the distance between two Gaussian distributions (as identified by the t-statistic) can be represented as:

$$\mathcal{T}_{i,j}^t = \frac{\left|\frac{1}{j-i}\sum_{x_k \in \gamma_{i,j}^t} \mathbf{x}_k - \frac{1}{N-j}\sum_{x_k \in \gamma_{j,N}^t} \mathbf{x}_k\right|}{\Delta_{i,N}^t}. \tag{10}$$

In Eq. 10, the numerator is simply the absolute value of the difference in the means of the data and the denominator is the standard deviation of the data over the whole window normalized by the appropriate degrees of freedom. If we denote the variance of the data in the two sub-windows as $\delta^t_{i,j}$ and $\delta^t_{j,N}$, then $\Delta^T_{i,N}$ can be computed as follows:

$$\Delta^t_{i,N} = \sqrt{\frac{(j-i-1)\delta^t_{i,j} + (N-j-1)\delta^t_{j,N}}{(j-i-1)+(N-j-1)}\left(\frac{1}{j-i}+\frac{1}{N-j}\right)}. \tag{11}$$

Let us introduce two new variables such that:

$$s_{i,j} = \sum_{k=i}^{j} \mathbf{x}_k, \qquad \bar{s}_{i,j} = \frac{s_{i,j}}{j-i}, \qquad s^2_{i,j} = \sum_{k=i}^{j} \mathbf{x}^2_k. \tag{12}$$

Equation 10 can then be represented as:

$$\mathcal{T}^t_{i,j} = \frac{\left|\frac{s_{i,j}}{j-i} - \frac{s_{j,N}}{N-j}\right|}{\Delta^t_{i,N}}. \tag{13}$$

The variance of data in a given sub-window can be calculated as follows:

$$\begin{aligned}
\delta^t_{i,j} &= \frac{1}{j-i-1}\sum_{k=i}^{j}(\mathbf{x}_k - \bar{s}_{i,j})^2, \\
&= \frac{1}{j-i-1}\sum_{k=i}^{j}\left(\mathbf{x}^2_k + (\bar{s}_{i,j})^2 - 2\mathbf{x}_k\bar{s}_{i,j}\right), \\
&= \frac{1}{j-i-1}\left(s^2_{i,j} + \frac{(s_{i,j})^2}{(j-i)} - 2\frac{(s_{i,j})^2}{(j-i)}\right), \\
&= \frac{1}{j-i-1}\left(s^2_{i,j} - \frac{(s_{i,j})^2}{(j-i)}\right).
\end{aligned} \tag{14}$$

Furthermore, Eq. 11 can now be simplified as follows:

$$\begin{aligned}
\Delta^t_{i,N} &= \sqrt{\frac{s^2_{i,j} - \frac{(s_{i,j})^2}{(j-i)} + s^2_{j,N} - \frac{(s_{j,N})^2}{(N-j)}}{N-i-2}\left(\frac{N-i}{(j-i)(N-j)}\right)}, \\
\Delta^t_{i,N} &= \sqrt{\frac{s^2_{i,N} - \frac{(s_{i,j})^2}{(j-i)} - \frac{(s_{j,N})^2}{(N-j)}}{N-i-2}\left(\frac{N-i}{(j-i)(N-j)}\right)}.
\end{aligned} \tag{15}$$

Thus, the figure of merit $\Upsilon^{\text{MB-TSTAT}}_t$ for an (i, j) split can be computed using the sum of the data in the sub-windows $(s_{i,j})$ and the sum of the squares of all the elements $(s^2_{i,N})$. Since these terms can be easily updated incrementally, the algorithm takes $\mathcal{O}(N^2)$ times using constant, *i.e.*, $\mathcal{O}(1)$ memory. For high-dimensional data, a $\Delta^t_{i,N}$ score is computed for each dimension, and the final figure of merit is quantified by the norm of the vector holding the scores for each dimension.

Note that our proposed algorithms work for any possible (i, j) split within the window. There has been some research work such as (Guha et al. 2006) where the use of fixed-size sub-windows has been proposed (*i.e.*, when $N - j = j - i$). Using fixed-size sub-windows

Algorithm 3 Compute $\Upsilon_t^{\text{MB-TSTAT}}$

```
Inputs:
    Sliding window size:   $N$
    Sliding window :       $\Gamma^t = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
Local variables:
    Partial-sum:           $s_{tot}$ /* For an $(i, j)$ split, $s_{tot} = \sum_{k=i}^{N} \mathbf{x}_k$ */
    Partial-sum:           $s_r$ /* For an $(i, j)$ split, $s_r = \sum_{k=j}^{N} \mathbf{x}_k$ */
    Partial-sum:           $s_l$ /* For an $(i, j)$ split, $s_l = \sum_{k=i}^{j-1} \mathbf{x}_k$ */
    Partial-sum:           $s_{tot}^2$ /* For an $(i, j)$ split, $s_{tot}^2 = \sum_{k=i}^{N} \mathbf{x}_k^2$ */
    Recurrent figure of merit:$\mathcal{T}_{\text{MB-TSTAT}}^t$   /* Max. figure of merit for an $(i, j)$ pair */
Outputs:
    Figure of merit:       $\Upsilon_t^{\text{MB-TSTAT}}$   /* Max. figure of merit for $\Gamma^t$ */
1:  $s_{tot} = \mathbf{x}_N$
2:  $s_{tot}^2 = \mathbf{x}_N^2$
3:  for $i = (N-1)$ to 1 do
4:      $s_{tot} = s_{tot} + \mathbf{x}_i$
5:      $s_{tot}^2 = s_{tot}^2 + \mathbf{x}_i^2$
6:      $s_r = 0$;
7:      for $j = N$ to $i-1$ do
8:          $s_r = s_r + \mathbf{x}_j$
9:          $s_l = s_{tot} - s_r$;
10:         $\mathcal{T} = \frac{|s_r/(N-j) - s_l/(j-i)|}{\sqrt{\frac{s_{tot}^2 - \frac{(s_l)^2}{(j-i)} - \frac{(s_r)^2}{(N-j)}}{N-i-2} (\frac{N-i}{(j-i)(N-j)})}}$
11:         $\Upsilon_t^{\text{MB-TSTAT}} = \max((\mathcal{T}, \Upsilon_t^{\text{MB-TSTAT}})$ /*Updating figure of merit*/
12:     end for
13: end for
14: return $\Upsilon_t^{\text{MB-CUSUM}}$
```

can only provide a reduction in the asymptotic complexity of $\mathcal{O}(N^2)$ by a constant. As we will show in our experimental results, using fixed-sized sub-windows can be a significant prohibition and can affect the change detection performance adversely in some cases.

5 Experimental analysis

To facilitate a better understanding of the results and to interpret the behavior of underlying algorithms, we conducted an extensive experimental analysis on both simulated and real data. The effectiveness of a change detection method can be analyzed based on the following two performance metrics:

- Receiver Operating Characteristic Curve (ROC)—this is a plot of the true-positive rate against the false-positive rate, constructed as described in (Provost and Fawcett 1997).
- Activity Monitor Operating Characteristic (AMOC)—this is a plot of the mean detection delay (also called the average run-length of the algorithm), against the false-alarm rate.

We are investigating the algorithms under the assumption of a single change hypothesis (SCH) which, as noted, is appropriate for destructive changes that persist indefinitely after they have occurred. Under this assumption, once the algorithm has decided that the change has occurred, it believes that this change persists for all subsequent time steps, until the end of the data stream. This entails the following definitions of when true (false) positives (negatives) occur:

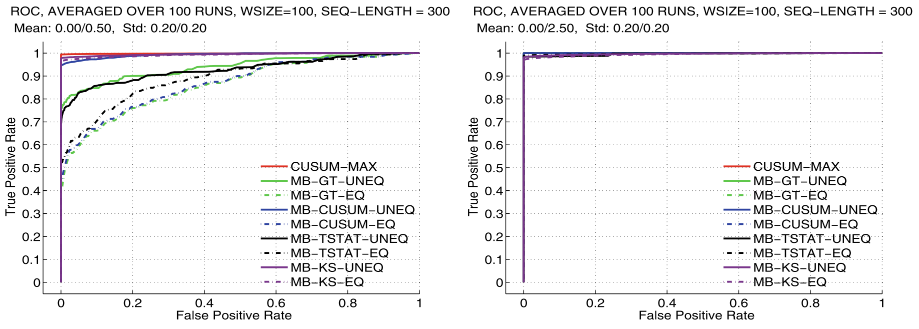
True positive (TP): Time step when the algorithm has declared that a change has occurred, and such a change has indeed already occurred in the data stream. Also known as a "hit".
False positive (FP): Time step when the algorithm thinks that a change has occurred, but it has not occurred yet in the data stream.
True negative (TN): Time step when the algorithm has not declared that a change has occurred, and such a step has indeed not occurred in the data stream.
False negative (FN): Time step when a change has in fact occurred, but the algorithm has failed to detect it yet. Also known as a "miss".

The corresponding rate for each of these four events is equal to the number of such events divided by the overall number of positive (respectively, negative) events. Under the SCH, when computing these four rates, what really matters is only the time when the change occurs respective to the time when the algorithm declares that it has occurred. If we denote the length of the data stream by M, the time when the change occurs by τ, and the time when the algorithm declares the change has occurred by $\bar{\tau}$, the following equations hold true. (Here the letter "R" after an event abbreviation denotes the rate of that event.) When $\bar{\tau} < \tau$ (early detection), $FPR = (\tau - \bar{\tau})/\tau$, $TNR = \bar{\tau}/\tau$, $FNR = 0$, and $TPR = 1$. When, conversely, $\bar{\tau} > \tau$ (late detection), $FPR = 0$, $TNR = 1$, $FNR = (\bar{\tau} - \tau)/(M - \tau)$, and $TPR = (M - \bar{\tau})/(M - \tau)$. Only when $\bar{\tau} = \tau$ it holds that $TPR = TNR = 1$, $FPR = FNR = 0$. By definition, the ROC curve is the plot of *TP* versus *FP* rates.

It turns out that under the assumption of a single change hypothesis (SCH), the AMOC curve can be derived trivially from the ROC curve by a simple reflection along the vertical axis. The reason is that under the SCH, the figure of merit grows monotonically, and the delay in detecting an abrupt change, as displayed on the y-axis of the AMOC curve, is exactly equal to the rate of false negatives, or misses. When there is a delay, its actual duration is $\bar{\tau} - \tau$, which, normalized by the number of time steps when change existed, gives $(\bar{\tau} - \tau)/(M - \tau)$ for the normalized delay of detection that is plotted in the AMOC curve. However, as noted, this is precisely equal to *FNR* under the SCH. And, since $FNR = 1 - TPR$, in its turn, it follows that the AMOC curve is a mirror image of the ROC curve across the vertical direction. For this reason, we present ROC curves only.

5.1 Simulated data

For simulated data, we used the CUSUM method as a gold standard for comparison, since it is known to be optimal when the data distributions before and after the change are known. In this way, it establishes an upper limit on the possible performance of any learning algorithm for change detection. The ROC curves for the CUSUM method were generated by providing the true shape and parameter information of the data distributions to the CUSUM algorithm. Although our novel algorithms operate in a parameter-free setting, we first present simulation results using Gaussian distributions (characterized by their means and standard deviations). Keeping the characteristics of the underlying data distribution fixed, different testbeds were constructed to study the effect of one particular aspect on the tested algorithms. For each testbed, we averaged the results over 100 independent runs. When averaging them, we used the vertical averaging method, as opposed to threshold averaging (Provost and Fawcett 1997).



(a) $AC - mean = 0.50$ (b) $AC - mean = 2.50$

Fig. 2 ROC plots for Testbed 1 with ($N = 100$, $BC - mean = 0.0$, $BC - std = AC - std = 0.20$)

5.1.1 Testbed 1—performance against difference in means in Gaussian data

The experimental results for this testbed were conducted on a one-dimensional data stream with 300 elements with a window size of 100 (*i.e.*, $N = 100$) such that the *change-point* always occurred at the center of the sequence (*i.e.*, at time index $\tau = 150$). The before-change (*BC*) data is always zero-mean. We keep the standard deviation of the data fixed before and after the change. Each algorithm was tested in two modes:

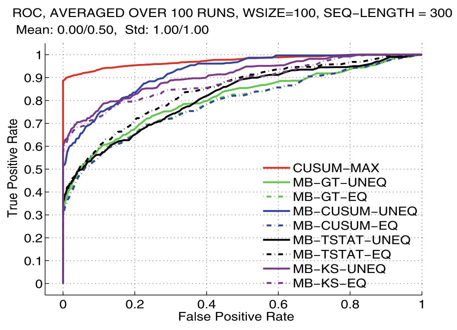
– *EQ*—Using only equally-sized sub-window splits, *i.e.*, $(N - j + 1 = j - i) \wedge (1 \le i < j < N)$.
– *UNEQ*—Using all possible window splits, *i.e.*, $1 \le i < j \le N$.

The objective of conducting experiments on Testbed 1 was to answer the following three questions:

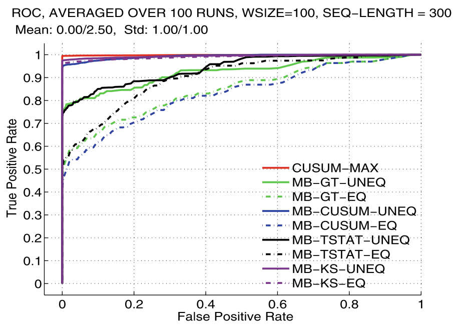
1. How does the performance change as we change the separation between the two distributions, as measured by the ratio between their means normalized by their standard deviations?
2. How well do the novel memory-based change detection schemes compare against the CUSUM method?
3. Is using all possible (i, j) splits instead of fixed-sized sub-windows providing any performance enhancement?

In Figs. 2–3, we show the ROC plots for different values of the standard deviation of the data. Since the standard deviation is fixed for a particular set of experiments, the ease of detection is directly under the influence of the change in the mean of the data. Hence the ease of detection increases from left to right in all the figures. As expected, all the algorithms show a consistent improvement in performance as we increase the *AC* mean. In terms of the *non-parametric* methods, the MB-KS emerges as the best method (only marginally inferior to CUSUM), and shows better robustness to data variance. However, this algorithm suffers from the prohibitive complexity of $\mathcal{O}(N^3 \log N)$. Furthermore, MB-CUSUM catches up quickly with MB-KS, and hence would be the *preferred* scheme, considering its low computational complexity. Hence, in the rest of our discussion, we will restrict ourselves to the other versions of the memory based algorithms.

Relatively, the MB-CUSUM algorithm performs better than any of the other algorithms, especially when in the *UNEQ* mode. In absolute terms, its performance is comparable to that



(a) $AC - mean = 0.50$



(b) $AC - mean = 2.50$

Fig. 3 ROC plots for Testbed 1 with ($N = 100$, $BC - mean = 0.00$, $BC - std = AC - std = 1.00$)

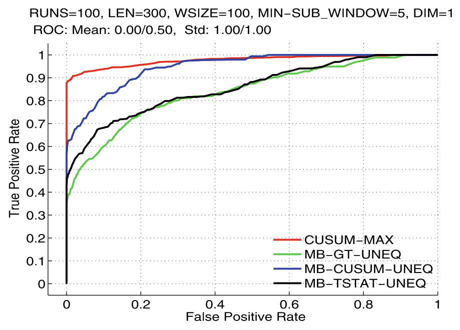
of CUSUM even when the change in the two distributions is subtle (as evident in Figs. 2(a) and 3(a)). When the change is significant, its performance is almost as good as that of CUSUM. This addresses Question 2 above. Furthermore, the performance of the MB-GT is pretty much similar to that of MB-TSTAT. However, note that the MB-TSTAT algorithm, operates under the assumption that the underlying data distributions are Gaussian, while the MB-GT algorithm does not make any such assumptions. Given the minimal memory requirements of the MB-TSTAT algorithm (note that the computational complexity of both algorithms is the same), one would prefer MB-TSTAT over MB-GT, if it is known *a priori* that the data distribution is Gaussian.

The answer to Question 3 is also evident from the figures. The *UNEQ* mode provides significant performance enhancement for all the proposed algorithms especially for the MB-CUSUM method. The *EQ* mode has negligible effect on the MB-TSTAT algorithm, however the MB-GT algorithm starts to show significant improvements in performance when the noise in data is high (*i.e.*, when the standard deviation is increased).

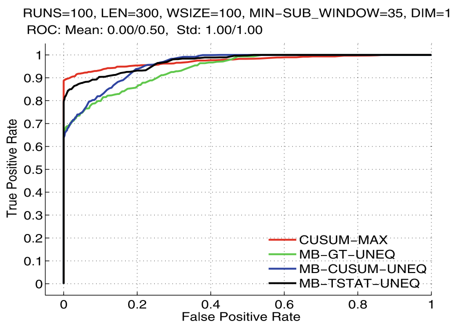
5.1.2 *Testbed 2—effect of the sliding window size N*

The experiments for this testbed were designed to analyze the effect of the sliding window size (N) on the performance of the algorithms. Having analyzed the results from Sect. 5.1.1, it is reasonable to assume that our memory based techniques provide best results when used in the *UNEQ* mode. Hence, to simplify the plots, we conducted experiments in this mode only.

We observed that for any fixed window size, the performance improved as we increased the data separability. Hence, the window size does not affect performance adversely. However, using too small windows can marginally lower the performance, especially when the data separability is not high. Whereas using very small windows can make the system susceptible to noise, using large windows can cause the system to lose sensitivity. Since MB-CUSUM operates using additive log-likelihood ratios, it is expected to be more robust to data noise than other algorithms. We observed that MB-CUSUM offered good performance when the window size was sufficiently large. Using very large windows can cause some degradation in the performance of the MB-GT and MB-TSTAT algorithms. However, overall, all the algorithms are robust to changes in the window size. Also, the relative performance of the algorithms does not change with the window size, and MB-CUSUM is the best performer in all cases.



(a) $MSW = 5$



(b) $MSW = 35$

Fig. 4 ROC plots for Testbed 3 with ($N = 100$, $BC - mean = 0.0$, $AC - mean = 0.50$, $BC - std = AC - std = 1$)

5.1.3 *Testbed 3—performance against the size of the smallest sub-window*

As we discussed in the earlier section, our memory based techniques compute the figure of merit by analyzing sub-windows of many possible sizes. Although this helps in making our methods more general (as seen in Sect. 5.1.1), it could introduce arbitrary bias in the performance results. For example, consider an (i, j) split on sliding window of size 300, with $i = 1$, and $j = N$. The sample sizes in question are 299 and one, respectively. Regardless of the algorithm used, estimates computed over a sub-window of one element only have no statistical significance. Hence, the objective of constructing this testbed is to verify if having a *lower bound* on the size of a sub-window (*MSW*) can affect detection accuracy. In Figs. 4–5, we show the ROC curves obtained as we increased this lower bound on the size of the smallest sub-window, for different separability of the data. Note that the data separability in the figures doesn't change in the horizontal direction.

One direct observation from the figures is that the MB-CUSUM algorithm is not very sensitive to the *MSW* bound, and does not exhibit a significant change in the performance as we increased the lower bound. But generally, all the algorithms show improvements in performance as the lower bound was raised. Algorithms working on the Euclidean space distances are more sensitive to the lower bound and show significant performance enhancements using larger sized sub-windows. MB-TSTAT is most sensitive to this lower bound, matches CUSUM when $MSW = 35$, and outperforms MB-CUSUM by a significant margin.

5.1.4 *Testbed 4—effect of data dimensionality*

While the MB-CUSUM and the MB-GT algorithms naturally generalize to high dimensional data, we compute the figure of merit for the MB-TSTAT algorithm by first computing the t-statistic for each dimension individually, and then taking the norm over the computed values. We generated high dimensional data such that data on each dimension was *i.i.d.*, and there was no covariance between data from different dimensions. The results suggested that all three schemes do not exhibit any performance degradation when increasing the data dimensionality; just on the contrary, the higher the dimensionality of the data, the closer the curve for the $MB - CUSUM$ algorithm is to that of the $CUSUM$ algorithm that uses the true data distributions. One possible explanation for this effect is that each of the dimensions of the data adds more evidence that a change has indeed occurred.

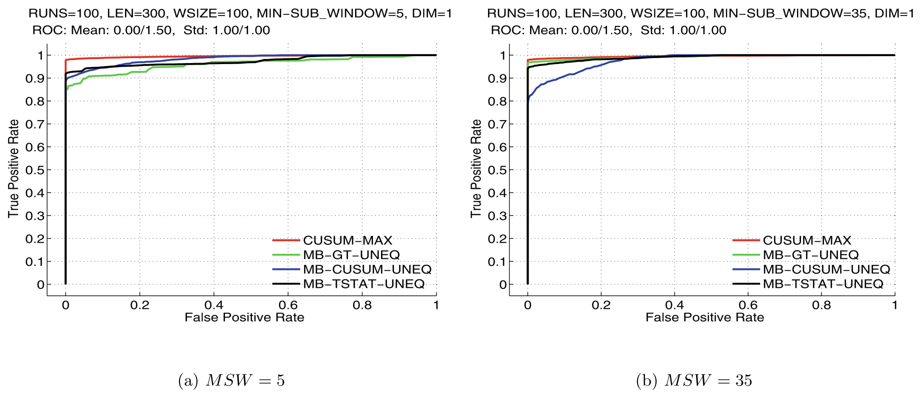


Fig. 5 ROC plots for Testbed 3 with ($N = 100$, $BC - mean = 0.0$, $AC - mean = 1.5$, $BC - std = AC - std = 1$)

5.1.5 *Testbed 5—performance on other distributions*

For the sake of a comprehensive experimental analysis, we tested the performance of our methods on other distributions, as well. As we have already seen in Sect. 5.1.3, the size of the smallest sub-window has a significant effect on the relative performance of our memory-based techniques. Hence, in this section, as we vary the distribution separation, we also vary the *MSW* factor, in order to verify if a similar trend of relative change in performance is prevalent.

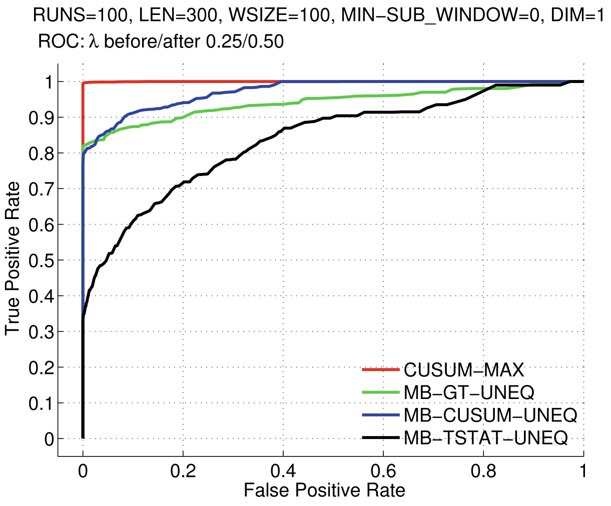
We focused the experiments in this testbed on exponential distributions, since their shape is very different from that of Gaussian distributions. While keeping the before-change distribution parameter λ fixed, we increased the after-change distribution parameter. An interesting observation is that detecting change is significantly easier when the *AC* distribution is *wider-spread* than the *BC* distribution. That is, the case when $\lambda_{BC} > \lambda_{AC}$ is much harder than the opposite case. Furthermore, when the above condition holds true, the MB-TSTAT algorithm can significantly improve its performance when we increase the *MSW*. However, in the opposite case, MB-CUSUM is the best performer. Also, MB-GT's performance degrades significantly for *harder* cases.

One hard representative case, when the two means are fairly close ($\lambda_0 = 0.25$, $\lambda_1 = 0.5$), is shown in Fig. 6. It can be seen that detecting the change is trivial for the omniscient CUSUM algorithm, which is supplied with the exact distributions p_0 and p_1. The two algorithms MB-CUSUM and MB-GT perform relatively well, while the Gaussian assumption built into the MB-TSTAT algorithm results in much worse performance.

5.2 Experiments with data acquired from a physical system

In this section, we present the performance results of the proposed memory-based change detection algorithms on data collected from a real physical system. The change-detection task at hand was to detect the state transition of an air conditioner's compressor (from off to on) based on temperature observations of the air blowing out of the air conditioner's vent. The entire air conditioner was indoors, i.e., both its evaporator (internal heat exchanger) and condenser (external heat exchanger) were in contact with room air. The ground-truth (the actual state of the compressor) was obtained by monitoring the power consumption of the

Fig. 6 For exponential distributions, the relative performance of the algorithms remains the same



air conditioner at a constant fan speed. The air conditioner always drew less than 100 Watts of power when the compressor was 'off', and more than 400 Watts when 'on'.

To introduce variability in the data, we collected temperature observations at different times of the day while operating the air-conditioner under different set-point requirements that were varied between 19°C and 22°C. We used two temperature sensors (accuracy $\pm 0.1^{\circ}$C) to record the mean air-temperature of the air coming out of the vent of the air conditioner, and an energy meter to monitor the power consumption of a 5,000 BTU/h window air-conditioner. Both the temperature and the power data were sampled uniformly at one-second intervals. We collected over five hours of data; during that time the compressor switched its state from off to on 36 times.

In Fig. 7, we have shown a snapshot of the temperature time-series obtained using the described procedure. In the figure, the solid line represents the temperature observations obtained when the compressor was on and the broken line when it was off. The temperature falls steadily when the compressor is on, but rises gradually otherwise.

The raw time series of vent air temperature does not conform to the assumptions of the change detection algorithms described above, since there is no discontinuous jump in air temperature due to the constraints of the physical system, and even more importantly, the data before and after the change (compressor switch) do not come from stationary distributions. Rather, when the compressor is off, the temperature rises gradually and steadily, and when it is on, it declines gradually and steadily. However, it might be expected that the *rate of change* of temperature would be much more stationary: positive before the compressor is turned on, and negative after that. Furthermore, this change from positive to negative will indeed be abrupt.

This reasoning suggests that pre-processing the time series by differencing it would bring the data sequence closer to the assumptions of the change detection algorithms. To this end, we performed change detection experiments on two types of time series:

– zeroth-order: the original time series of vent air temperatures;
– first-order: the first difference of the vent air temperatures.

Since our change-detection algorithms are tailored for single-change hypothesis, we processed the recorded data further to obtain shorter data-streams such that each individual

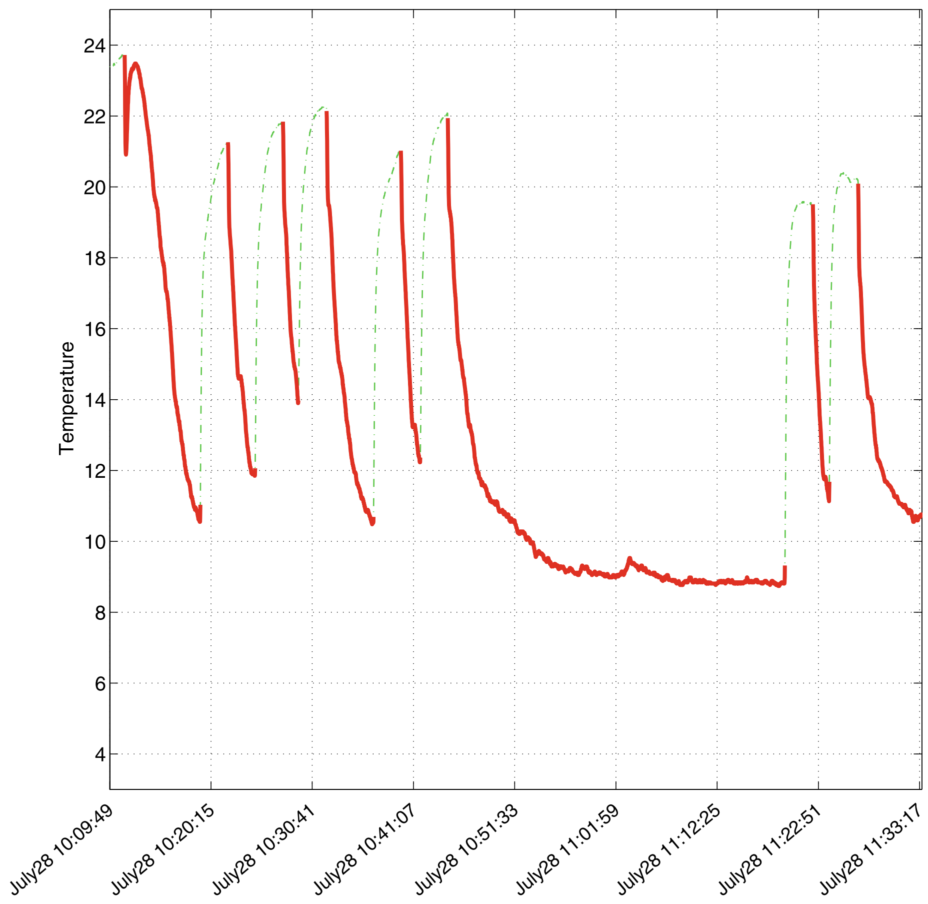


Fig. 7 Time series from the vent of an air conditioner. The *solid line* corresponds to operation of the fan when the compressor is switched on; the *broken line* shows measured air temperature when the compressor is off

stream had only one 'off' to 'on' transition or change-point in it. We randomly generated 100 such data streams of 420 observations (*i.e.*, 7 minutes) with the change-point positioned randomly within the stream. Since there are only 36 change-points in the entire dataset, the 100 data-streams generated had repetitions; however the change-point position was at different position in different runs. We report performance using ROC curves (aggregated using vertical averaging) over the 100 runs.

For the CUSUM algorithm, we model both the before-change and after-change temperatures by normal distributions. Note that unlike the testbed with simulated data reported above, we cannot provide it with the true data distributions before and after the change. Rather, we provide it with estimates of the parameters of these distributions from data.

We used two different ways of estimating these parameters: one where the distribution parameters were obtained once from the entire 5 hour data, and another where the parameters were obtained for *each* individual run (we called this approach CUSUM-LOCAL). If the streaming data distribution is truly normal, then the CUSUM-LOCAL algorithm is expected to offer the best performance.

We also tested a domain-specific change detector that is based on physical understanding of the process of heat exchange in air. Heating by convection is based on Newton's law of cooling, where the exchanged heat is proportional to the difference in temperatures of the two bodies that participate in the exchange. In this case, these two bodies are the evaporator of the air compressor and the air going through it. When the compressor is off, the temperature of the room air goes up, and the temperature of the evaporator and the refrigerant in it goes up, too. When the compressor is turned on, the relatively warm refrigerant is removed from the evaporator and replaced with refrigerant that has low heat content and low temperature. At this point the difference between the temperatures of the evaporator and the room air entering it is the largest, and hence the rate of heat exchange is also the largest. This corresponds to the fastest change in the temperature of air coming out of the vent.

Based on this reasoning, we can build a domain-specific change detector that would signal a change when the first difference of air temperature is higher than a certain threshold. However, this threshold would be different under different operating conditions, user settings, and air conditioner models, so the accuracy of detection will vary. We report its ROC curve along with that of the other window-based change detectors. (In order to generate this ROC curve, we use the *negative* first difference as the figure of merit, so as to keep consistency in ROC curve generation policy, where higher figure-of-merit indicates higher probability of the sample being an *after-change* sample.) Similarly, the actual temperature value could also be used as a figure-of-merit, although its performance is expected to be much lower.

In Figs. 8–11, we report ROC curves for different window sizes ($N = 10, 25, 50, 100$) for all change-detection algorithms when operating on absolute temperature data (i.e., 0th-order). We also report the ROC curves for the two domain-specific change-detectors VELOCITY (one that uses the negative rate of change, i.e. first difference of temperature as the figure of merit) and DATA (one that uses the data itself as a figure of merit). Note that it is not possible to capture the distribution of the 0th-order data reliably since it is not stationary and thus the probabilistic change-detection algorithms are not expected to work well.

As can be seen in Figs. 8–11, the DATA algorithm comes out as a top-performer followed by CUSUM-LOCAL. However, the MB-CUSUM algorithm still offers the best performance for a non-parametric and non-domain specific change detector. It is also the best algorithm for $N = 50$ and $N = 100$. Even the MB-GT algorithm performs better than the CUSUM and the MB-STAT algorithms. We further observe that increasing the window size improves the performance of MB-CUSUM, and it performs as well as the DATA algorithm with $N = 50$. Larger windows allow for better approximation of the ever changing data distribution. While the CUSUM algorithm exhibits poor performance when the parameters are obtained from the entire dataset, the CUSUM-LOCAL algorithm works well since it always has a more accurate estimate of the distribution parameters. However, since the data is not likely to be truly normal, MB-CUSUM offers better performance since it does not make any underlying assumptions on the data distribution.[1] While working with the 0th-order data, one can expect the DATA algorithm to work reasonably well, since the temperature in the compressor "on" state is highly likely to be lower than for the compressor "off" state. Theoretically, it is always possible to find a temperature threshold that would detect the change with a reasonable accuracy. However, learning this threshold is not trivial as it will be sensitive to the ambient conditions and user set-points. Other the other hand, probabilistic algorithms mea-

[1]The optimal kernel bandwidth depends on the window size, however in all our experimental analysis we used a fixed kernel bandwidth value of unity.

Fig. 8 $N = 10$, 0th-order

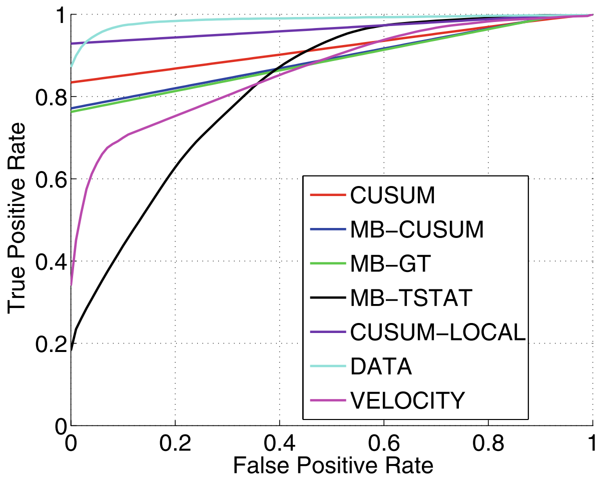
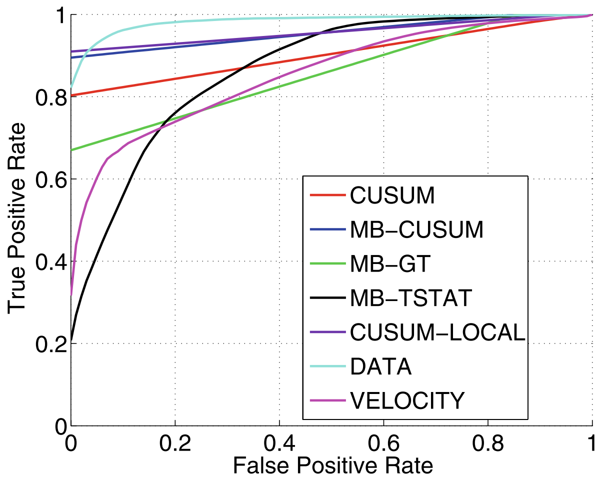


Fig. 9 $N = 25$, 0th-order



sure the change in the data distribution which is likely to be consistent across user settings simplifying the task of threshold selection.

Our change detection algorithms (including CUSUM) are best-suited to detect change in two *stationary* data distributions. For this dataset, the distribution of the rate of change of the temperature (i.e. its first-order derivative) is likely to be much more stationary than the data itself. This derivative can be approximated conveniently by simply computing the difference between two consecutive temperature values. In Figs. 12–15 we show experimental results with first-order differences as we change the window size. Since this is a much better representation of the data, we observe that both versions of the CUSUM algorithm offer close to 100% detection accuracy. Furthermore, our proposed algorithms outperformed the DATA algorithm by significant margins when used with small sized windows. Hence, our proposed methods match CUSUM's performance, without making any underlying assumptions about the distribution of the data. We observe that in this case, increasing the window size af-

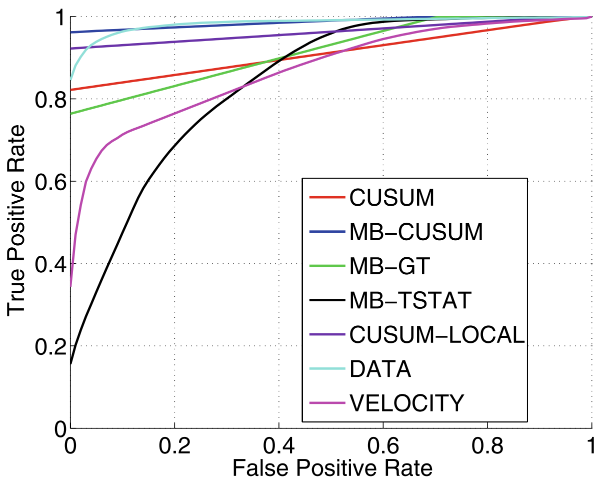


Fig. 10 $N = 50$, 0th-order

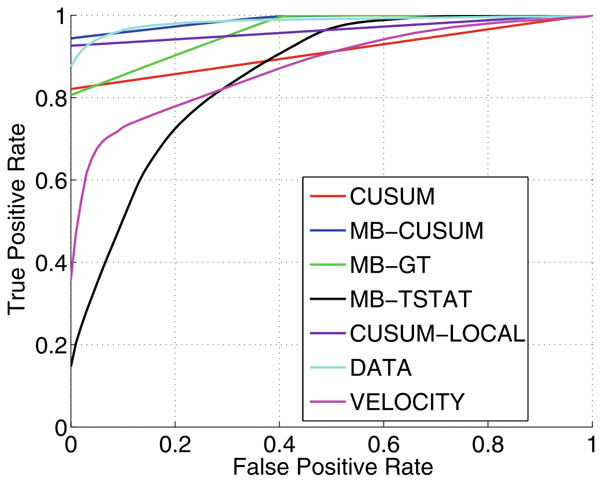


Fig. 11 $N = 100$, 0th-order

fects the performance of our algorithms adversely, which means that relatively small-sized windows are better suited to this application.

5.3 Experimental results on dynamical systems with dependent observations

One of the fundamental assumptions of the CUSUM algorithm is that the observations in the time series are independent and identically distributed. This assumption is violated when the observations are produced by an underlying dynamical system, since the dynamics of that system correlate consecutive observations, thus creating statistical dependencies among them. In particular, the joint likelihood of the data points in a window cannot be factored as in Eq. 4.

The traditional method to deal with dynamical systems is to continuously estimate models of these systems, and execute a change detection algorithm on the *parameters* of the model, rather than on the original observations. This method works especially well for linear

Fig. 12 $N = 10$, 1st-order

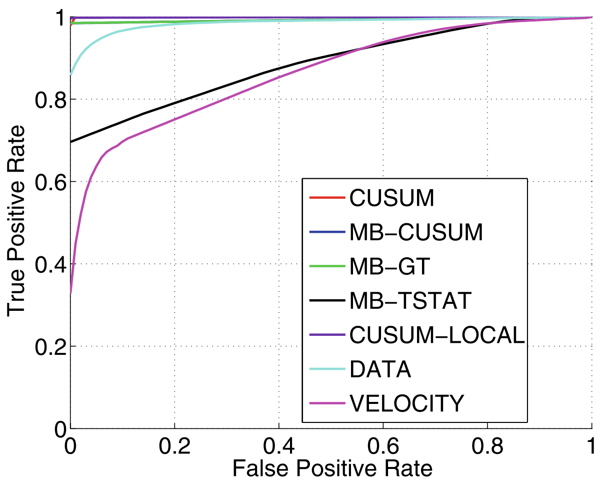
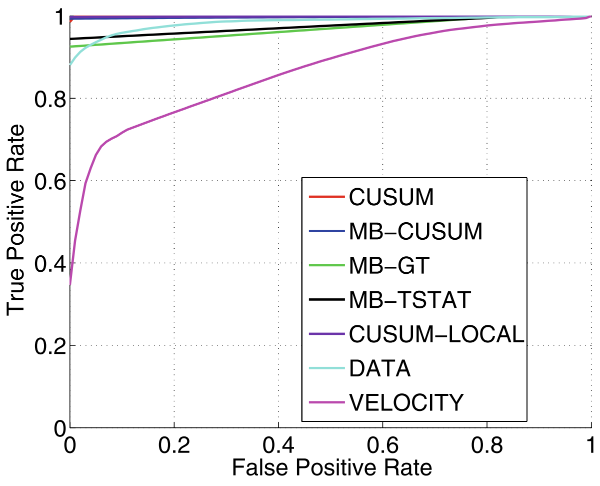


Fig. 13 $N = 25$, 1st-order



systems: in such cases, linear ARMA models are fit to windows of the observed data, and the estimated vector of model parameters can be monitored directly by a change detection algorithm such as CUSUM (Brodsky and Darkhovsky 1993). Since the parameter estimates for linear systems are linear combinations of time-lagged windows of observations, Gaussian noise in the observations translates directly into Gaussian noise in the estimates of model parameters. This facilitates significantly the specification of the correct distributions before and after the change that are needed for the proper operation of the CUSUM algorithm.

However, this approach works only for linear systems whose order p is known, and even in such cases, it is fairly expensive computationally. For each candidate split (i, j) of the memory buffer, the algorithm would have to estimate two linear models of order p. This operation is linear in the size of the buffer N, but generally cubic in the order of the system p, resulting in computational complexity of $O(N^3 p^3)$ when considering all possible splits (i, j). Reusing partial results of the ARMA parameter estimation across different pairs

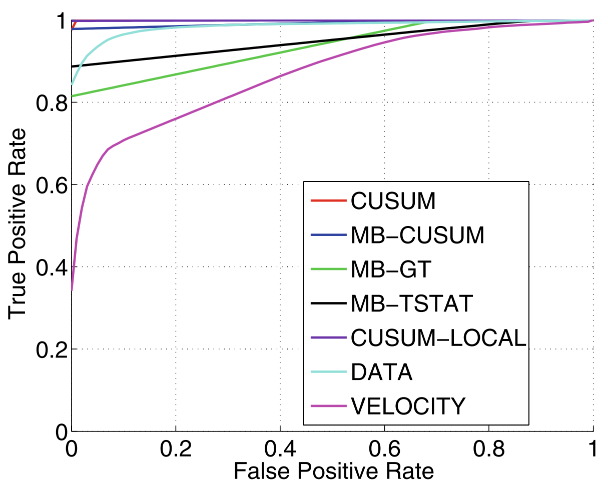


Fig. 14 $N = 50$, 1st-order

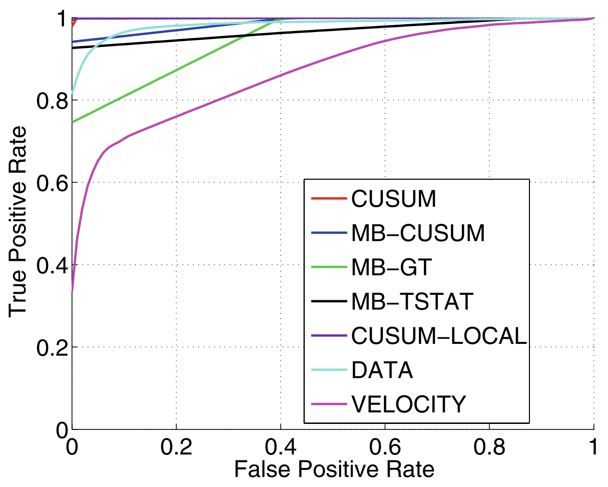


Fig. 15 $N = 100$, 1st-order

(i, j) does not seem possible, since ARMA estimation for each pair involves the inversion of a matrix that incorporates only the observations for this particular split.

Given the high computational cost of this approach, we decided to investigate the performance of the two algorithms proposed by us, when applied *directly* to the observations in a time series generated by a dynamical system. One particularly challenging detection problem for this approach is to detect changes in a harmonic oscillator whose frequency only changes, but whose amplitude remains the same.

The differential equation that governs a harmonic oscillator is $\ddot{x} = -\omega^2 x$, where ω is the frequency of the oscillator. Its solution $x = \sin(\omega t)$ is a sine wave. Very clearly, the observations in the time series $x(t)$ are not independent. The probability distribution of these observations is not Gaussian—rather, its two peaks are at the minimum and maximum of the range. Moreover, this distribution *does not* depend on the frequency ω, so if the only change in the system is a change in frequency, but not in amplitude, the data distributions

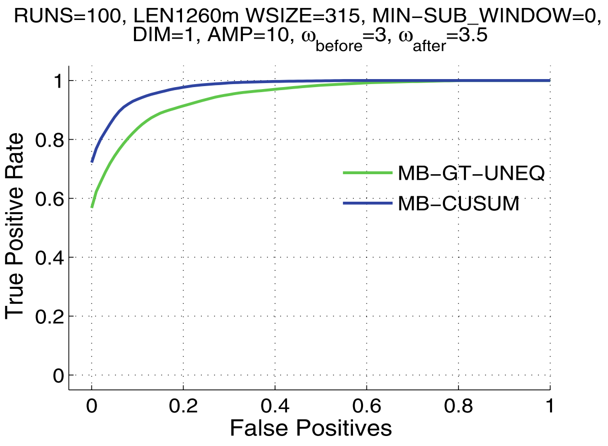


Fig. 16 Change detection in a harmonic oscillator whose frequency increases from $\omega_{before} = 3.0$ Hz to $\omega_{after} = 3.5$ Hz

before and after the change would be completely identical, and traditional CUSUM executed on these observation would have no way of detecting the change.

However, surprisingly, the two algorithms proposed in this paper performed very well in this setting. In our experiments, we used a discrete-time version of the harmonic oscillator with added Gaussian noise in the dynamics. In discrete-time state-space form, the system can be represented as:

$$\begin{aligned} \dot{x}^t &= \dot{x}^{(t-1)} - \omega^2 x \Delta t, \\ x^t &= x^{(t-1)} + \dot{x}^t \Delta t + \mathcal{N}(0, 1). \end{aligned} \tag{16}$$

In our experiments, we used a sampling frequency of 50 Hz (sampling interval $\Delta t = 0.02$) , and the initial conditions were set to $x^0 = 10$ and $\dot{x}^0 = 0$, corresponding to a cosine wave. The datasets always had 630 observations such that for a unit frequency value ($\omega = 1$), one sinusoidal cycle had 157 observations. Similarly to previous experiments, the change point is exactly in the middle of the time series.

In order to preserve the amplitude of the signal when changing its frequency, the change must occur when the velocity of the oscillator is zero, i.e., when $x(t)$ is at one of the two extremes of its range. The physical rationale for this is that a harmonic oscillator corresponds to an undamped mass-spring system, and the frequency of oscillation is determined by the spring constant and the mass of the system. Changing the mass when the velocity is not zero would change the kinetic energy of the system, which would result in a different amplitude of motion. Because of this, in our experiments, the number of samples in the first half of the time series was always an exact multiple of the period of the wave before the change; the frequency after the change can have an arbitrary value.

The two algorithms performed surprisingly well for various frequency changes, both when the frequency after the change was increased and when it was decreased with respect to that before the change. Figures 16 and 17 show the ROC curves for two fairly difficult cases, when two close frequencies (3 Hz and 3.5 Hz) were used. (Results are again averaged over 100 runs.)

We attempted to analyze the surprisingly good performance of the two algorithms at detecting the change point, and believe that it is due to the fact that they always compare two contiguous and continuous portions of the relatively slowly varying signal. (The relatively slow variation is ensured by the typically high sampling rates in modern monitoring

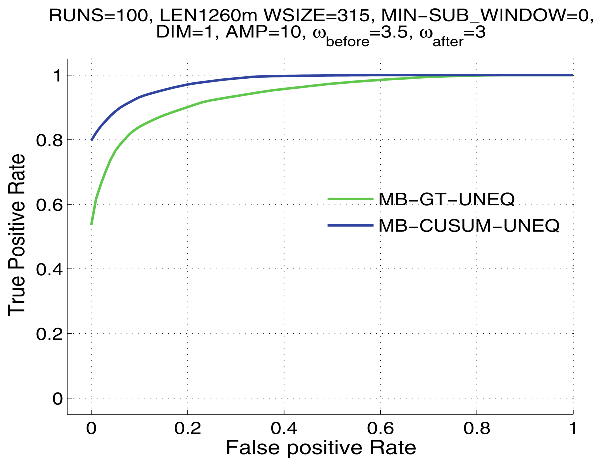


Fig. 17 Change detection in a harmonic oscillator whose frequency decreases from $\omega_{before} = 3.5$ Hz to $\omega_{after} = 3.0$ Hz

systems—the sampling rate is always designed to be above the Nyquist rate, or twice the frequency of the fastest-varying periodic component of the sampled signal.)

We experimentally observed that regardless of the size N of the memory buffer allowed for analysis, the combined size $N - i$ of the two windows that resulted in maximum difference C_{ij} (respectively, S_{ij}) always spanned less than one full cycle of the sine wave. This can be explained by the fact that adding more cycles to either window would actually decrease the difference between the two windows, since the average value of the samples in a complete cycle is zero. This suggests that our algorithms are in fact computing distances that correlate well with the first derivative of the slowly varying signal, and since this derivative changes when the frequency of oscillation changes, the algorithms are able to detect the change correctly.

6 Conclusion

Two novel fast memory-based algorithms for abrupt change detection were proposed, and their performance versus known methods for change detection was tested. Experimental verification under various conditions such as type of distribution, magnitude of change, data dimensionality, and memory buffer size confirmed their good performance. Between the two, the memory-based extension to CUSUM, MB-CUSUM, outperformed MB-GT in practically all cases, which can be attributed to its probabilistic foundation and the optimality guarantees for its non-learning predecessor, the original CUSUM algorithm. The only algorithm that outperformed MB-CUSUM was the one using the Kolmogorov-Smirnov statistic; however, this algorithm is very expensive computationally ($O(N^3 \log N)$), and inherently limited to one- or at most two-dimensional data. In contrast, MB-CUSUM is fast ($O(N^2)$), and easily handles multivariate data; its accuracy even increases on higher-dimensional data, at least when the dimensions are uncorrelated. In addition, MB-CUSUM is not too sensitive to the size of the memory-buffer. Furthermore, MB-CUSUM and MB-GT are non-parametric memory-based algorithms that can work on all kinds of distributions, whereas using t-tests is limited to those that do not depart too much from Gaussian distributions. The performance of the proposed algorithms were also verified on a data stream from a real physical system, achieving near perfect accuracy of change detection.

Furthermore, the two algorithms performed surprisingly well on change detection in dynamical systems whose amplitude remains constant, while only their frequency changes. Whereas such a change cannot be detected by the traditional CUSUM algorithm, and instead computationally expensive system models have to be fit to data, both MB-CUSUM and MB-GT were able to reliably detect even relatively small changes in frequency. This is likely due to their direct comparisons between sets of samples, rather than reliance on pre-specified data distributions before and after the change.

The proposed algorithmic solutions reduce the complexity of computing their respective figures of merit from $O(N^4)$ to $O(N^2)$; the usefulness of these solutions is much reinforced by the experimental verification that considering all possible splits of the memory buffer does make a big difference when compared to the case when only equal splits are used. This is a substantial improvement over the current practice in the field, where typically only one single split into two windows of size $N/2$ is considered. Imposing a lower limit on the size of either window can easily be implemented, and often results in better detection accuracy.

As to whether further improvements in computational complexity are possible, one direction is to try to amortize computation across time periods. Currently, computation for each time period t starts from scratch, and one might imagine an alternative scheme where the statistics $C_{i,j}$, respectively $S_{i,j}$, are computed from their counterpart values in previous time slices. However, since these statistics are placed in a tableau of size $O(N^2)$, retaining these tableaux in their entirety would destroy the $O(N)$ memory property of the algorithms.

Another direction to consider is the use of more advanced memory-based learning algorithms, for example ones that rely on data structures such as kd-trees, adaptive rectangular trees, ball-trees, etc. (Atkeson et al. 1997). These methods have been used very effectively in memory-based learning, especially for lower-dimensional data, and are suitable candidates for future use in abrupt change detection algorithms.

References

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, *11*(1–5), 11–73.

Basseville, M., & Nikiforov, I. V. (1993). *Detection of abrupt changes: theory and application*. Englewood Cliffs: Prentice Hall.

Brodsky, B. E., & Darkhovsky, B. S. (1993). *Nonparametric methods in change-point problems*. Dordrecht: Kluwer.

Fawcett, T., & Provost, F. (1999). Activity monitoring: noticing interesting changes in behavior. In *Proceedings of the fifth international conference on knowledge discovery and data mining (KDD-99)* (pp. 53–62).

Guha, S., McGregor, A., & Venkatasubramanian, S. (2006). Streaming and sublinear approximation of entropy and information distances. In *Proceedings of SODA'06* (pp. 733–742). New York: ACM Press.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning*. Berlin: Springer.

Jain, A., & Nikovski, D. (2007). *Memory-based change detection algorithms for sensor streams* (Technical Report TR2007-079). Mitsubishi Electric Research Laboratories. http://www.merl.com/publications/TR2007-079.

Massey, F. J., Jr. (1951). The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, *46*(253), 68–78.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*, 100–115.

Provost, F. J., & Fawcett, T. (1997). Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In *Proceedings of KDD'97* (pp. 43–48). Menlo Park: AAAI Press.