Expert Systems With Applications 209 (2022) 118260

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Real-time Change-Point Detection: A deep neural network-based adaptive approach for detecting changes in multivariate time series data

Muktesh Gupta *, Rajesh Wadhvani, Akhtar Rasool

Department of CSE, Maulana Azad National Institute of Technology, Bhopal, M.P., India

ARTICLE INFO

Keywords:
Adaptive Change-Point Detection
Data normalization
Deep learning
Recursive singular spectrum analysis
Lagrangian multiplier
Overcomplete autoencoder

ABSTRACT

The behavior of a time series may be affected by various factors. Changes in mean, variance, frequency, and auto-correlation are the most common. Change-Point Detection (CPD) aims to track down abrupt statistical characteristic changes in time series that can benefit many applications in different domains. As demonstrated in recently introduced CPD methodologies, deep learning approaches have the potential to identify more subtle changes. However, due to improper handling of data and insufficient training, these methodologies generate more false alarms and are not efficient enough in detecting change-points. In real-time CPD algorithms, preprocessed data plays a vital role in increasing the algorithm's efficiency and minimizing false alarm rates. Therefore, preprocessing of data should be a part of the algorithm, but in the existing methods, preprocessing of data is done initially, and then the whole dataset is passed to the CPD algorithm. A new three-phase architecture is proposed to address this issue, in which all phases, from preprocessing to CPD, work in an adaptive manner. The phases are integrated into a pipeline, allowing the algorithm to work in real-time. Our proposed strategy performs optimally and consistently based on performance metrics resulting from experiments on real-world datasets and artifacts. This work effectively addresses the issue of non-stationary data normalization using deep learning approaches. To reduce noise and outliers from the data, a recursive version of singular spectrum analysis is introduced. It is demonstrated that the method's performance has significantly improved by combining adaptive preprocessing with deep learning CPD techniques.

1. Introduction

The Internet of Things (IoT) devices and sensors produce an endless stream of data. The fast proliferation of linked devices and networks generates a massive volume of data. IoT data is generally expressed as a time series, a sequence of observations over time describing system behavior. Dynamic change happens due to systemic change in internal or external events. Change-Point Detection (CPD) is the problem of discovering the sudden change in the data when the attributes of the time series change (Aminikhanghahi & Cook, 2017). If any change occurs in system behavior, these insights and the hidden structure help in identifying the change-points in the time series. The importance of the CPD in the time series can be observed in many applications, such as DNA sequencing, in which DNA structure is divided into segments based on different properties of interest (Braun et al., 2000). The CPD is also used in atmospheric climate change analysis for monitoring the possible occurrence of variation in climate and greenhouse gases (Reeves et al., 2007). In medical condition monitoring, CPD plays an important role in identifying irregularities in patient health (Reeves et al., 2007). The CPD is also used in segmentation and to recognize boundaries between words in speech recognition (Gupta, 2015). CPD has become extremely important for human activity analysis to detect changes in the behavior of human health in the smart home (Bermejo et al., 2021). CPD plays an essential role in financial data analysis and sensor network data (Xie & Siegmund, 2013). A change-point in a time series can be due to the jumping mean, where the mean of a time series fluctuates with constant variance. There is also a possibility that the mean of a time series remains constant while the change in variance occurs. Another scenario is when both the mean and the variance of a time series change. All the mentioned change-point types in time series are considered by the present state-of-the-art literature (Aminikhanghahi & Cook, 2017). Still, a few more complex types of changes present in time series are based on the change in frequency and changes in the auto-correlation of time. (Mastrantonio et al., 2022).

Some well-known algorithms described in the literature use the entire dataset to calculate the change-point, such as the method introduced by Kawahara et al. (2007) to detect change-points in time series data using subspace identification, the method introduced by Koepcke et al. (2016) to detect single and multiple change-points in spike trains

* Corresponding author.
E-mail addresses: mg.203112003@manit.ac.in (M. Gupta), rajeshwadhvani@manit.ac.in (R. Wadhvani), akhtarr@manit.ac.in (A. Rasool).

https://doi.org/10.1016/j.eswa.2022.118260
Received 15 February 2022; Received in revised form 23 June 2022; Accepted 20 July 2022
Available online 26 July 2022
0957-4174/© 2022 Elsevier Ltd. All rights reserved.

using CUmulative SUM (CUSUM), the method introduced by Darkhovsky and Piryatinska (2018) to detect multiple CPD in multidimensional time series using ϵ (epsilon) complexity of continues vector function and the recently introduced method by Katser et al. (2021) to detect change-point using ensemble technique. All these algorithms identify change-points only on a predetermined dataset; therefore, they are unsuitable for real-time applications. The primary requirement of some applications, such as medical monitoring, stock prediction, human activity analysis, and so on, is to detect change-points in real-time as data arrives. Hence, our objective is to come up with an effective way to find change-points that can be used in a real-time application. Many real-time CPD algorithms have been introduced that are based on deep learning techniques, such as the dual-LSTM framework combined with a CPD module to predict the remaining useful life of bearings in real-time (Shi & Chehade, 2021). Another method that detects online memory-free CPD is based on the LSTM Autoencoder proposed by Atashgahi et al. (2021). The method introduced by Yoo et al. (2021) predicts change-points in the stock market by adopting the Denoising LSTM Autoencoder model. For the CPD of weather forecasting, Selvi et al. (2021) has introduced Bi-LSTM and 1D CNN based methods. Although these approaches use deep learning techniques to detect change-points in real-time, there are some limitations to these algorithms. The CPD module in all these methods adaptively updates to detect change-points, but the preprocessed data is fed directly into the algorithm. As the deep learning-based CPD module has already been proposed to work in an adaptive manner, there is a requirement for the module that preprocesses the data in an adaptive manner as well. The preprocessed data cannot be overlooked because it improves the algorithm's efficiency and reduces false alarm rates. Therefore, our second objective is to make the preprocessing phase adaptable so that it can be linked to the CPD module and both preprocessing and CPD can happen in real-time. Many real-time algorithms are limited in their ability to detect a change-point as soon as it arrives. Methods introduced by Rezvani et al. (2019) calculate Mutual Information (MI) between the consecutive windows and then detect change-points if present. Another method based on the direct ratio estimation introduced by Kato et al. (2022) detects change-points based on the dissimilarities between two consecutive segments. These approaches have a delay in detecting the change-point since they wait for the whole segment data to arrive before calculating statistical comparisons between windows or segments, and so the algorithm's performance is dependent on the window size or segment length size assumption. There is a requirement to develop a methodology based on threshold so that the waiting time for a segment data to arrive can be reduced. Therefore, our third objective is to present a novel approach based on deep learning techniques that can detect complex change-points more quickly and can be used in real-time applications.

Deep learning models require properly normalized data for better training as well as adequate smoothing of data in order to acquire higher accuracy and a lower false alarm rate. Hence, preprocessing steps such as data normalization and data smoothing approaches that work in real-time are emphasized (Sola & Sevilla, 1997). The Min–max and Z-score are the most prevalent static normalization algorithms for time series data in neural networks (Eesa & Arabo, 2017). As the time series data may be non-stationary, typical static approaches cannot offer accurate results; hence, an adaptive solution is necessary to normalize the data. There is also a need for an algorithm that can do smoothing in real-time and eliminate noise from time series. Many methods based on an adaptive version of CPD with outliers have been proposed. The method introduced by Fearnhead and Rigaill (2019) is based on a penalizing cost approach on a loss function which is less sensitive to outliers. Another algorithm based on Adaptive Bayesian and Change-point Analysis and Local Outliers Scoring uses the state-space approach (Wu & Matteson, 2020). The Enhanced Annotation Framework for Activity Recognition through CPD based on deep learning by Dhekane et al. (2022) also works for outlier removal along with CPD. All these methods do not focus on adaptive normalization of data and directly use normalized data if required to detect change-points. Therefore, to address all these issues, a new three-phase architectural framework has been proposed. The first phase normalizes the data; the second reduces noise and outliers; and the third detects the change-point. All these phases are connected in such a way that the data flows in a pipeline throughout the process; i.e., when data arrives, the output of one phase is fed into the next, allowing the algorithm to run in real-time. In the first phase, adaptive normalization is carried out using data-driven Deep Adaptive Input Normalization (DAIN) (Passalis et al., 2019). It provides normalized data that adapts to the statistical features of recently processed data. In the second phase, after acquiring normalized output, a recursive version of the Singular Spectrum Analysis (SSA) approach is developed to obtain smoother and noise-free time series in real-time. The SSA decomposes data into several independent additive components based on the covariance matrix. It inherently calculates the singular value decomposition (SVD) and finds the most prominent principal component, which captures most of the variance of a projection with the help of eigenvalues and eigenvectors (Golyandina et al., 2018). In the third phase, an overcomplete autoencoder is used to detect change-points in time series data. An autoencoder is a sort of feed-forward network that primarily performs two functions: encoding and decoding. During the training phase, the autoencoder learns to rebuild the output vector with the least amount of reconstructed error. Because the change-point is unlikely to be foreseeable, the trained model will produce an unexpected reconstruction error above the threshold. The dynamic thresholding technique is used to detect numerous change-points in a time series.

The following are the major contributions of our proposed method:

- A novel three-phase CPD framework based on fully end-to-end adaptivity is proposed, which combines the preprocessing phase and the change-point detection module, allowing the algorithm to run in real-time with improved efficiency.
- The performance of the existing DAIN architecture is improved by optimizing the hidden layer parameters, which reduces the scaling and shifting errors.
- A recursive version of SSA is proposed. Perturbation-based recursive steps are introduced to find the most useful eigenvalue and eigenvector in SVD decomposition. The Recursive SSA (RSSA) will remove noise and outliers in real-time.
- A new method based on deep learning techniques has been presented to identify CPD. A change-point is detected by analyzing the reconstruction loss error using an autoencoder. The regularization of the model is focused so that it can train on complex input as well.
- All three phases, i.e., adaptive normalization, recursive singular spectrum analysis, and change-point detection phase, are interlinked in such a way that new data is processed in a pipeline from one phase to the next, allowing the algorithm to operate in real-time.

The organization of the rest of the paper is as follows: In Section 2, work done in literature is discussed. In Section 3, problem formulation for the change-points and RSSA has been listed. Section 4 discusses materials and methods; the details of the deep learning-based CPD approach; the proposed method's complexity analysis; performance measurements; and parameter setting. Section 5 is devoted to results and discussion. Section 6 is for the conclusion and future scope of the paper.

2. Related works

Many different criteria can be used to categorize CPD techniques. The distinction between the online CPD algorithm, which provides real-time detection with some delay, and offline CPD algorithms, which

provide more powerful detection but require whole data, is prevalent (Reeves et al., 2007). Both criteria include supervised and unsupervised methods (Aminikhanghahi & Cook, 2017). Many improvements and adaptations have been selected according to the desired outcome of an algorithm. Supervised approaches require Machine-Learning (ML) algorithms to find state boundaries based on the specified state given in the training phase (Wang, 2016). These approaches can have both binary and multi-class classifier ML algorithms. Some well-defined algorithms of multi-class classifiers are Naive Bayes (Reddy et al., 2010), Support Vector Machine (Jin et al., 2019), Nearest Neighbor (Zhou, 2019), Gaussian Mixture Model (GMM) (Saatçi et al., 2010), Bayesian Net (Mohammad-Djafari & Féron, 2006), Decision Tree (Auret & Aldrich, 2010) and Hidden Markov Model (Luong et al., 2012). Even though the supervised approach has a shorter training phase, enough diverse training data is required to represent all the classes (Keogh et al., 2004). On the other hand, the binary classifier treats CPD as a binary classification problem where one class represents all potential state transition (change-point) sequences, and the other class represents all within-state sequences. Examples are logistic regression (Desobry et al., 2005), Naive Bayes (Feuz et al., 2014) and Support Vector Machines (Camci, 2010).

CPD methods based on unsupervised learning algorithms are more robust, as prior training is not required for each behavior change in the system (Chalapathy & Chawla, 2019). Traditionally, control charts are used to detect change-points. These charts can easily detect major changes and isolated abnormal points, but fail to detect subtle changes. The Cumulative sum (CUSUM) was introduced to replace the control chart and is the most well-known CPD algorithm still employed in today's applications (Taylor, 2000). The CUSUM can detect change-points based on the cumulative sum chart along with the confidence level. In CUSUM, bootstrapping is used to calculate the confidence level of a detected change-point (Hinkley & Schechtman, 1987). Another non-parametric unsupervised approach that has been identified in the literature is based on Bayesian methods (Adams & MacKay, 2007). These methods apply the Bayes theorem to estimate the run length of a segment. Each time, this run length can be set to zero or increased by one, depending upon the change in the probability distribution of segments. The unsupervised learning algorithms generally segment the time series data based on statistical feature changes. The algorithm proposed by Liu et al. (2013) calculates the divergence estimation between two consecutive intervals of the data segment. The likelihood ratio is calculated instead of the density estimation of two segments. The Kullback–Leibler Importance Estimation Procedure (KLIEP) introduced by Liu et al. (2013) uses KL divergence to estimate Direct Ratio Estimation (DRE). Another method, known as unconstrained Least Square Importance Fitting (uLSIF) uses Pearson Divergence as a dissimilarity measure (Khan et al., 2019) to estimate DRE. The recently introduced improved version of uLSIF known as Relative uLSIF (RuLSIF) by Hushchyn and Ustyuzhanin (2021) removes the limitation of unbounded ratios by considering relative density ratios. A New Pattern Representation Method (NPR) introduced by Rezvani et al. (2019) uses entropy instead of probability distribution to detect change-points. This method also reduces the time series length using a piece-wise aggregated approximation. Finally, it detects change-points, but the CPD false rate depends on the window size and number of ranges assumed in the data stream.

The deep learning techniques are new ways to deal with the complex problem of CPD. The Ebrahimzadeh et al. (2019) proposed a Pyramid Recurrent Neural Network(PRNs) that uses CNN to recognize short-term patterns with trainable wavelet layers and PRNs on top to detect long-term patterns. PRNs are able to detect abrupt and gradual change-points in multivariate time series. The scale-invariant PRNs are compared to scale-sensitive Convolution Neural-based Networks(CNN) and Hierarchical Multi-scale Recurrent Neural Networks (HM-RNN) using synthetic datasets generated by mixing the Brownian process with white noise. The findings reveal that PRNs have a reduced false alarm rate and are more efficient in detecting changes. Compared to other approaches, PRNs have a larger Area Under Curve (AUC). However, PRNs have some shortcomings, such as the inability to handle noisy and missing labels. PRNs are based on supervised learning, which requires more training time and makes it nearly impossible to train a network for all types of changes (Lattari et al., 2022). An autoencoder-based technique that uses statistical data filtering to detect change-points is proposed by Maleki et al. (2021). The results derived using recall and precision matrices demonstrate that this method accurately detects all online and offline change-points. However, this method is limited to univariate time series, and data is preprocessed using static techniques that have been used for decades. Another methodology based on the deep learning technique uses an autoencoder that can detect all types of changes in time series (Deryck et al., 2021). This method focuses on CPD with a Time-Invariant Representation and successfully detects change-points on non-iid (independent and identically distributed) data by combining frequency and time domains. The performance is evaluated using AUC by setting an application-specific threshold parameter. With the autoencoder, we extended the methodology to work in online mode. The proposed work majorly focuses on resolving all the limitations of algorithms working in real-time.

3. Problem formulation

Consider N multivariate time series having d dimensions (features) of length L which is represented as $Y_L^{(i)} = (y_0^{(i)}, y_1^{(i)}, \ldots, y_L^{(i)})$ where $y_t^{(i)} \in \mathbb{R}^{d\times L}$ and i = (1,2,…,N). To refer nth time series having d feature at time instance t will be denoted as $y_t^{(n)} \in \mathbb{R}^d$. The Change-point in sub-sequences $\{y_t^{(i)}, y_{t+1}^{(i)}, y_{t+2}^{(i)} \ldots y_{t+n}^{(i)}\}$ in a time series can be defined as a hypothesis testing problem between two alternatives, the null hypothesis H_0 : "No Change-point occurs at time t", and alternate hypothesis H_A : "A change occurs to time t" represented as: H_0 : $\rho y_t^{(i)} = \cdots = \rho y_{t+\tau}^{(i)} = \cdots = \rho y_{t+n}^{(i)}$ H_A : There exists t< τ < n such that $\rho y_t^{(i)} = \cdots = \rho y_{t+\tau}^{(i)} \neq \rho y_{t+\tau+1}^{(i)} \ldots = \rho y_{t+n}^{(i)}$, where ρ is the threshold value of the reconstructed loss function in autoencoder and $y_t^{(i)}$ is the start and $y_{t+n}^{(i)}$ is the end point of time series and $y_{t+\tau}^{(i)}$ is the change-point. Multiple change-points are also present in time series which are represented as $\{y_{t+\tau 1}^{(i)}, y_{t+\tau 2}^{(i)}, \ldots, y_{t+\tau n}^{(i)}\}$. The proposed algorithm can handle multiple change-points. The target is to find the correct number of these change-points τ without having any prior knowledge of where and how many change-points are present in the input time series. SSA is used during preprocessing, and a portion of the method calculates SVD to get eigenvalues and eigenvectors of time series. The problem is to find the principal component that captures the maximum variance. If X is the n-dimensional random vector projected on W_j unit norm, then projection is given by $y_i = W_j^T X$. The solution to this optimization problem is to find the first principal component, which is given by:

$$W_1 = \arg\max_{W} (W)^T RW, \text{ subject to: } W^T W = 1$$

where R is the covariance matrix. The subsequent principal component is determined by the extra restrictions on the problem of orthogonality in the previously acquired component as:

$$W_j = \arg\max_{W} (W)^T RW$$

$$\text{subject to: } W^T W = 1, W^T W_1 = 0, 1 < j$$

The objective is to train a neural network based on autoencoder such that if a change in the behavior of a system occurs, reconstruction loss should cross the threshold and immediately algorithm flag the ongoing evaluation timestamp as a change-point.

4. Materials and methods

4.1. Datasets

In the following section, five artificial datasets are generated, and four real-word datasets are chosen to evaluate the proposed algorithm. For reference, all dataset summaries are listed in a Table 1.

4.1.1. Artificial dataset

A time series of 500 samples with 1000 observations is constructed using an auto-regressive model, which is given by the following equation:

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \epsilon_t \tag{1}$$

Here, α_1 and α_2 are the correlation coefficient of time series having lag 1 and lag 2 represented as y_{t-1} and y_{t-2}, respectively. The ϵ_t is the Gaussian noise at timestamp t with a mean μ and a standard deviation σ. An outlier is also added between regular intervals of every 100 timestamps. The outlier value with the random values is added. The interval between the two changes in the time series is also chosen randomly. The change-point position is calculated as an incremental multiple of λ using the equation:

$$\epsilon_t = \beta + \lambda \times i \tag{2}$$

Here, $\beta = 50$ are the buffer timestamps which the model requires for training before detection of any change-point, $\lambda = 25$. and i is the multiplicative factor. By changing the parameters of the auto-regressive model given in Eq. (1) many types of datasets are created (Hurvich & Tsai, 1989), which are described in further sections.

Dataset 1 (Change in Mean). An auto-regressive model given in Eq. (1) is used for the generation of time series observations having only drift in mean by assuming values of $\alpha_1 = 0.6$ and $\alpha_2 = 0.5$. The drifted mean value is randomly chosen from the range [a, b], in our case [2,8] with a step size k = 2. The interval between two change-points are calculated using Eq. (2). The values of y_1 and y_2 are initialized as ϵ_1 and ϵ_2 respectively, and standard deviation is taken as 0.5.

Dataset 2 (Change in Variance). For the generation of changing variance datasets, the auto-regressive model given in Eq. (1) is used. The value of μ is fixed to origin-centered, and the value of variations is selected between ranges [0.5, 1.5] with step size k = 0.2. The intervals between two changes in variance are randomly chosen using the same equation used for changing the mean.

Dataset 3 (Change in Mean and Variance). For the generation of dataset 3, the auto-regressive model given in Eq. (1) is used. Both the mean and the variance change randomly from the range [2,8] and [0.5,1.5] with a step size k = 2 and 0.2 after the change-point, respectively. The value of α_1 is taken as 0.6 and α_2 as 0.5. The interval between two changes is randomly chosen using the same equation used for changing the mean.

Dataset 4 (Change in Frequency). The following equation was used to generate the change in frequency dataset:

$$y_t = sin\{\omega t\} + \epsilon_t \tag{3}$$

where ϵ_t is the origin centered noise and the standard deviation $\sigma = 0.5$, ω is the angle provided in radians. The interval between two changes in frequency is also randomly using Eq. (2).

Dataset 5 (Change in Auto-correlation). For the generation of this dataset, the auto-regressive model is used as given in Eq. (1). The values of α_1 and α_2 are changed in the range [0-1]. To get highly correlated time series both α_1 and α_2 are chosen as 1 and for no correlation in time series α_1 and α_2 are set to zero. The interval between two changes is randomly chosen using Eq. (2).

4.1.2. Real-world dataset

Dataset 6 (Well log). Well logging is the process of analyzing the geological formulations and properties of the rock. This process is done with the help of real-time monitoring of data received during the penetration of a borehole. The data generated is in the form of time series. The changing mean represents the change in the rock stratification. The dataset is made up of a one-dimensional time series with 675 points that has some noise and outliers.

Dataset 7 (Apple Stock). This dataset consists of the daily open, high, low, close price, adj close, and volume of Apple stock from 1996 to 2004, comprising 1866 observations. The dataset is reduced by sampling every three observations to get a time series of length 622. The close price and volume are only considered to detect the change-point.

Dataset 8 (CO2 Emission). CO2 describes carbon dioxide emissions from the burning of fossil fuels in metric tonnes per person in Canada. This dataset consists of 214 observations of CO2 emissions from 1800 to 2004. The CO2 variation is observed from 0 to 17.5 metric tons in the respective years.

Dataset 9 (Room Occupancy Data). The dataset consists of observations of room occupancy recorded in a one-hour interval of a day, having 18 143 observations. The dataset also contains multiple variables such as temperature, humidity, light, and CO2 based on which room occupancy is dependent. The dataset is reduced by sampling every 16 observations to get the final time series of the length of 509 observations.

4.2. Deep learning-based CPD methodology

The proposed methodology is divided into three phases: adaptive normalization, recursive singular spectrum analysis, and change-point detection, respectively. An overview of the proposed algorithm has been illustrated using algorithm 1 for better understanding, and all of these phases are shown in the Fig. 1 along with the methodology. The detailed working of these phases is discussed in further sections.

4.2.1. Adaptive normalization of a time series

The non-adaptive ways of normalization usually perform Z-score and Min–max scalars on time series features. These techniques are based on fixing normalization statistics for both the training and inferences phase, which leads to sub-optimal results (Passalis et al., 2019). Recently, a new method known as DAIN, that is based on deep learning techniques has been used for the adaptive normalization of data. Similar to Z-score normalization, the method performs shifting followed by scaling, but the goal of the network is to perform both operations appropriately and is represented by:

$$\hat{y}_t^{(i)} = (y_t^{(i)} - \alpha^{(i)}) \oslash \beta^{(i)} \tag{4}$$

Here, $\alpha^{(i)} = W_a E^{(i)}, \beta^{(i)} = W_b Z^{(i)}$ and $W_a, W_b \in \mathbb{R}^{d \times d}$ is a weight matrix of shifting and scaling respectively and $\oslash$ is an entry wise Hadamard division operator. The aggregate summary of shifting (difference mean from observations) of time series sequence $E^{(i)}$ is represented as averaging all L measurements:

$$E^{(i)} = \frac{1}{L}\sum_{t=1}^{L} y_t^{(i)}, L \in \mathbb{R}^D \tag{5}$$

and the aggregate summary of scaling of time series sequence $Z^{(i)}$ will be represented as standard deviation of all L measurements:

$$Z^{(i)} = \sqrt{\frac{1}{L}\sum_{t=1}^{L}(x_{t,k}^{(i)} - E^{(i)})^2}, k \in \mathbb{R}^D \tag{6}$$

Gradient descent of all parameters after learning in the end-to-end fashion given as:

$$\Delta(W_a, W_b, D, W) = -\eta(\eta_a \frac{\partial L}{\partial W_a}, \eta_b \frac{\partial L}{\partial W_b}, \eta_c \frac{\partial L}{\partial D}, \frac{\partial L}{\partial W}) \tag{7}$$

Table 1
A summary of synthetic and real-world datasets.

S. No.	Dataset	Length	No. of series	No. of features	No. of CP	No. of Outliers
1	Changing mean	1000	500	1	6	9
2	Changing variance	1000	500	1	6	9
3	Changing mean and variance	1000	500	1	6	9
4	Changing frequency	1000	500	1	6	9
5	Changing auto-correlation	1000	500	1	6	9
6	Well Log	675	1	1	10	8
7	Apple stock	622	1	2	8	5
8	CO2 emission	214	1	1	7	7
9	Room occupancy	509	1	4	11	6

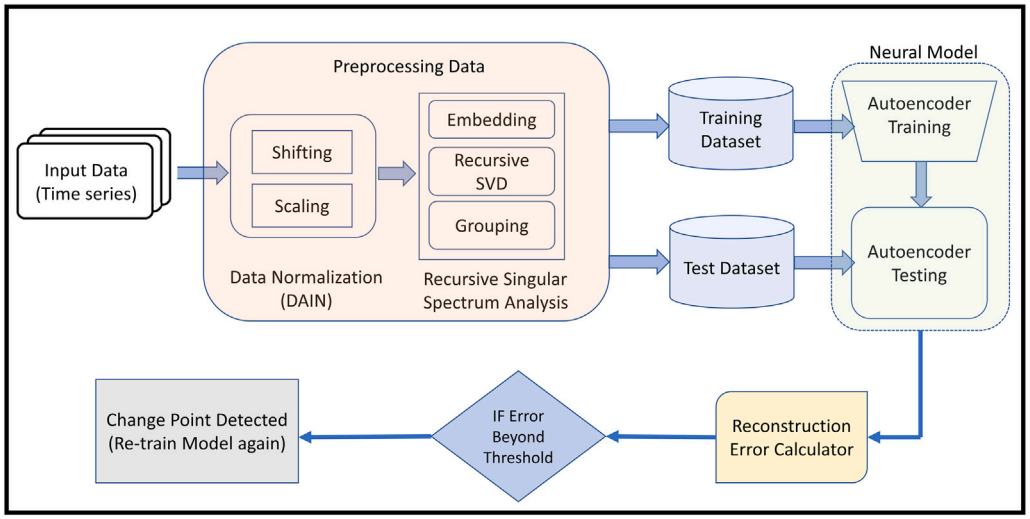

Fig. 1. The proposed CPD based deep learning techniques. The figure illustrates the data flow from the integrated preprocessing module with the CPD module.

Here, η_a, η_b, and η_c are parameters of learning rates and ∂L is the loss function used for training, and W are the weights of the neural network. The DAIN model employs certain timestamps at the start of proper training in real-time and can also function with multivariate time series. Once trained, the model can able to normalize any newly arrived datasets adaptively (Nalmpantis et al., 2021).

Algorithm 1 Change-Point Detection Algorithm Overview

INPUT:REAL-TIME APPLICATION DATA

OUTPUT:CHANGE-POINTS WHEN CHANGE IN APPLICATION BEHAVIOR

while (Data arrives in real-time application) **do**
  if (Initial N Timestamps arrived in buffer) **then**
    Initialize Deep Adaptive Input Normalization
    Initialize Singular Spectrum Analysis
    Initialize Change-Point Detection model
  else if (Newly arrived data after initialization) **then**
    Adaptive Normalization using **DAIN**→ Normalized output
    Recursive SSA→Noise and outlier free output
    Autoencoder output→ Reconstruction Error
    if (Reconstruction Error>Dynamic Threshold) **then**
      Change-Point Detected
    end if
  else
    continue
  end if
end while

4.2.2. Recursive singular spectrum analysis

After getting a normalized time series, the SSA method does two operations: decomposition and reconstruction of the time series (Claessen & Groth, 2002). The SSA separates the original time series into independent additive components based on their characteristics. The main purpose of using SSA is to make any time series smoother by eliminating outliers and noise. The SSA steps are discussed in greater detail below.

Decomposition. The SSA first maps time series $Y_L(i)$ to a multidimensional vector lag sequence to form a trajectory matrix, where L is the length of some ith time series, and then decomposes the trajectory matrix. All the steps are discussed in the following sections:

Embedding. Embedding is the process of obtaining a multi-dimensional matrix from a one-dimensional time series (Golyandina et al., 2018). It can be seen as a vector of delay coordinates $[\mathbf{x}(\mathbf{t}), \mathbf{x}(\mathbf{t}+\mathbf{1}), \ldots, \mathbf{x}(\mathbf{t}+\mathbf{K})]$ of a single time series, where K = L − W + 1, and W is the window length. The time series $Y_L^{(i)}$ is applied to embedded matrix $\mathbf{X}^{(i)} = [\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \ldots, \mathbf{x}_K^{(i)}]$ with vector $\mathbf{x}_j^{(i)} = (y_j^{(i)}, \ldots, y_{j+W_i+1}^{(i)})^T \in \mathbb{R}^{2<W_i<L}$. After mapping the multidimensional matrix obtained, also known as the Hankel matrix, which is represented as:

$$\mathbf{X}^{(i)} = [\mathbf{X}_1^{(i)}, \ldots, \mathbf{X}_{Ki}^{(i)}] = (y_{mn})_{m=0,\ n=0}^{Wi-1\ \ ki-1} \tag{8}$$

For N time series $\mathbf{X}^{(i)}$ (i = 1,2...N) a block of Hankel matrix will be obtained which is represented as:

$$\mathbf{X}_\delta = \begin{bmatrix} \mathbf{X}^{(1)} \\ \mathbf{X}^{(2)} \\ \vdots \\ \mathbf{X}^{(N)} \end{bmatrix}$$

Singular value decomposition. The factorization of the Hankel matrix $\mathbf{X}_\delta$ is performed in SVD. The Eigenvalues of covariance matrix (cov) obtained are denoted as $\lambda_{\delta 1}, \lambda_{\delta 2}, \ldots, \lambda_{\delta L_{SUM}}$, which is matrix multiplication of $X_\delta X_\delta^T$. The Unitary matrix $U_{v1}, U_{v2}, \ldots U_{vL_{SUM}}$ are the

corresponding eigenvectors, where $L_{SUM} = \sum_{i=1}^{N} L_i$. The structure of the matrix is shown below:

$$\mathbf{X}_\delta \mathbf{X}_\delta^T = \begin{bmatrix} \mathbf{X}^{(1)}\mathbf{X}^{(1)T} & \mathbf{X}^{(1)}\mathbf{X}^{(2)T} & \dots & \mathbf{X}^{(1)}\mathbf{X}^{(N)T} \\ \mathbf{X}^{(2)}\mathbf{X}^{(1)T} & \mathbf{X}^{(2)}\mathbf{X}^{(2)T} & \dots & \mathbf{X}^{(2)}\mathbf{X}^{(N)T} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{X}^{(N)}\mathbf{X}^{(1)T} & \mathbf{X}^{(N)}\mathbf{X}^{(2)T} & \dots & \mathbf{X}^{(N)}\mathbf{X}^{(N)T} \end{bmatrix}$$

$$cov_K = \frac{1}{K}\sum_{i=1}^{K} \mathbf{X}_\delta \mathbf{X}_\delta^T \tag{9}$$

Here, cov_K is also known as a variance–covariance matrix that includes all auto and cross-covariance functions of $\mathbf{X}$ (Hassani & Mahmoudvand, 2013). The SVD of $\mathbf{X}_\delta$ can be written as $\mathbf{X}_V = \mathbf{X}_{V1} + \mathbf{X}_{V2} + \cdots \mathbf{X}_{VL_{SUM}}$ where $\mathbf{X}\delta_i = \sqrt{\lambda_i} U_{vi} V_{Vi}^T$ and $V_{vi} = \mathbf{X}_V^T U_{Vi}/\sqrt{\lambda} Vi$

The SSA algorithm must be recursive and capable of handling real-time applications. By assuming only one time series represented as $Y_L = (y_0, y_1, \dots, y_L)$, we presented an recursive version of our SSA. The lagged vector representation of time series Y_L taking window size as W will be given by $\mathbf{X} = \{\mathbf{X}_t, \mathbf{X}_{t+1}, \dots \mathbf{X}_{t+K-1}\}^T$ where $K = N - W + 1$. After embedding resulting $K \times W$ trajectory matrix is obtained, which is also a Hankel matrix and is represented as:

$$\mathbf{X} = \begin{bmatrix} y_0 & y_1 & \cdots & y_{L-W} \\ y_1 & y_2 & \cdots & y_{L-W+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{W-1} & y_W & \cdots & y_{L-1} \end{bmatrix}$$

As our time series is already having L element, a newly arrived data y_{L+1} will increase the length of time series by one, the trajectory matrix dimension $K \times W$ changes to $(K+1) \times W$. After embedding of $\mathbf{X}$ newly arrived data is represented as:

$$\mathbf{X} = \begin{bmatrix} y_0 & y_1 & \cdots & y_{L-W} \\ \vdots & \vdots & \ddots & \vdots \\ y_{W-1} & y_W & \cdots & y_{L-1} \\ y_W & y_{W+1} & \cdots & y_{L+1} \end{bmatrix}$$

The next step is to calculate the covariance matrix of the trajectory matrix X. To make the covariance matrix recursive, the last iteration of covariance estimation is combined with a new vector. Let K be a new vector, then the covariance estimation will be given as:

$$\begin{aligned} cov_k &= \frac{1}{k}\sum_{i=1}^{k} X_i X_i^T \\ &= \frac{k-1}{k} cov_{k-1} + \frac{1}{k} X_k X_k^T \end{aligned} \tag{10}$$

As our covariance matrix is symmetric, it has precisely n real eigenvalues and n linearly independent eigenvectors. These vectors are orthogonal when they correspond to distinct eigenvalues (Abdi, 2007). The Eigenvalue decomposition of covariance matrix (cov) is given as $cov_k = P_k Q_k P_k^{-1}$, where P holds eigenvectors of cov_k and Q holds the eigenvalues of cov_k in the diagonal. The columns of P matrix are orthogonal if all eigenvalues are distinct, and if they are not orthogonal, we can choose orthonormal and normalize them with the unit length. This makes the matrix P orthogonal, and the inverse of the orthogonal matrix will be its transpose. The eigendecomposition of cov_k matrix if P is orthonormal eigenvector and Q is diagonal eigen matrices is given by

$$\begin{aligned} cov_k &= P_k Q_k P_k^T \\ cov_{k-1} &= P_{k-1} Q_{k-1} P_{k-1}^T \end{aligned} \tag{11}$$

Putting values of Eq. (11) and assuming $\alpha_K = P_{k-1}^T X_k$ Eq. (10) can be written as:

$$P_k(kQ_k)P_k^T = P_{k-1}[(k-1)Q_{k-1} + \alpha_k \alpha_k^T]P_{k-1}^T \tag{12}$$

If we assume R_k as orthonormal eigenvector and S_k as diagonal eigenvalue matrices of $(k-1)Q_{k-1} + \alpha_k \alpha_k^T$, then eigen value decomposition can be written :

$$(k-1)Q_{k-1} + \alpha_k \alpha_k^T = R_K S_K R_K^T \tag{13}$$

With the help of Eq. (10), we get the reduced form of Eq. (9) as:

$$P_k(kQ_k)P_k^T = P_{k-1} R_K S_K R_K^T P_{k-1}^T \tag{14}$$

Recursive updates of eigenvalues and eigenvectors can be obtained by comparing both sides of Eq. (14) as:

$$\begin{aligned} P_k &= P_{k-1} R_K \\ Q_k &= \frac{S_K}{K} \end{aligned} \tag{15}$$

Now the problem of finding eigenvalue and eigenvector of correlation matrix cov_K in a recursive manner is reduced to the problem of eigen decomposition of matrix$(k-1)Q_{k-1} + \alpha_k \alpha_k^T$ and from Eq. (15) this eigen decomposition can be solved by finding values of R_K and S_K. The solution of Eq. (12) is determined employing the concept of perturbation analysis of the first order matrix (Liu et al., 2008). When value of K is large matrix$(k-1)Q_{k-1} + \alpha_k \alpha_k^T$ is a diagonal matrix, S_K will be equal to $(k-1)Q_{k-1}$, R_K is Identity matrix I and $\alpha_k \alpha_k^T$ is known as perturb of diagonal matrix $(k-1)Q_{k-1}$. As a result, R_K and S_K can be written as:

$$\begin{aligned} R_k &= (k-1)Q_{k-1} + P_\gamma \\ S_k &= I + P_\lambda \end{aligned} \tag{16}$$

Here, P_γ and P_λ are small perturbation matrices. Finding of P_γ and P_λ will give us the solution of eigen decomposition problem of Eq. (13). Let $Q = (K-1)Q_{k-1}$ then $R_K S_K R_K^T$ can be expended as:

$$\begin{aligned} R_K S_K R_K^T &= (I + P_\lambda)(Q + P_\gamma)(I + P_\lambda)^T \\ &= (Q + P_\gamma + P_\lambda Q + P_\lambda P_\gamma)(I + P_\lambda)^T \\ &= Q + P_\gamma + P_\lambda Q + P_\Lambda P_\gamma + Q P_\gamma^T + \\ &P_\lambda^T P_\gamma + P_\lambda^T P_\lambda Q + P_\lambda^T P_\lambda P_\gamma \\ &= Q + P_\gamma + P_\lambda(Q + P_\gamma) + P_\lambda^T(Q + P_\gamma) + \\ &P_\lambda^T P_\lambda Q + P_\lambda^T P_\lambda P_\gamma \\ &= Q + P_\gamma + P_\lambda R_K + P_\lambda^T R_K + P_\lambda^T P_\lambda Q + \\ &P_\lambda^T P_\lambda P_\gamma \end{aligned} \tag{17}$$

Substituting these value in Eq. (13) and neglecting small values such as $P_\lambda^T P_\lambda Q$ and $P_\lambda^T P_\lambda P_\gamma$ we get:

$$\alpha_k \alpha_k^T \equiv P_\gamma + P_\lambda R_K + P_\lambda^T R_K \tag{18}$$

Another characterized equation of P_λ which make $P_\lambda = -P_\lambda^T$ is derived for orthonormality of V i.e. $VV^T = I$. By substituting $V = I + P_\lambda$ assuming $P_\lambda P_\lambda^T \approx 0$ and combining all the facts i.e., P_λ is anti-symmetric and P_γ, S_K are diagonal we get solution as:

$$\begin{aligned} \alpha_K \alpha_K^T &= (i,i)^{th} \quad \text{element of } P\gamma \\ \frac{\alpha_i \alpha_j}{\rho_j + \alpha_j^2 - \rho_i^2 - \alpha_i^2} &= (i,j)^{th} \quad \text{element of } P\lambda, i \neq j \\ 0 &= (i,i)^{th} \quad \text{element of } P\lambda \end{aligned} \tag{19}$$

Here, ρ_i, ρ_j are the eigenvalues of the matrix $Q = (K-1)Q_{k-1}$. The temporal covariance of time series can be explained by eigenvectors measured at different time lags. The Principal Components (PC) are computed by projecting the embedded time series onto the eigenvectors. The PC measures the variation of time series in decreasing order. Thus, the first PC will capture the majority of the variance, followed by the second PC for the remainder, and so on.

Reconstruction.

Grouping. Reconstruction of a time series which is associated with eigenvectors can be obtained by combining it with the associated principal component (Golyandina et al., 2018). The SVD of X_δ written in form $X_V = X_{V1}+X_{V2}+\cdots X_{VL_{SUM}}$ is split into several disjoint groups and then summation of matrices is performed within each group. The $I_1,\ldots,I_m$ are the split of the set of indices $1,\ldots,L_{sum}$ corresponds to the representation $X_v = X_{i1}+\cdots+X_{in}$. The share of corresponding eigenvalues determines the component X_{V_I} contribution to a particular group I and is given by:

$$G = \frac{\sum_{i\in I}\lambda_{V_i}}{\sum_{i=1}^{d_V}\lambda_{V_i}} E_I = \sum_{i\in I}\lambda_{V_i}/\sum_{i=1}^{d_V}\lambda_{V_i} \tag{20}$$

where I is set of split indices $\{1,2,\ldots,L_{sum}\}$ and d_V is the rank of matrix X_V. In an ideal situation, all the components of a time series are summed up to form the original time series represented as $X = \sum_j X^{(j)}$ where j is the components of our time series that are grouped according to the elementary matrices.

Algorithm 2 Recursive Singular Spectrum Analysis

1: **Initialize** $W \leftarrow$ Dimension of Embedded Vector
2: **Construct** Trajectory Matrix
$\mathbf{X} = [ssa1, ssa2, \ldots ssa_w] \leftarrow$ **SSA**$(Lags)$
3: **Initialize** U_0 Σ_0 and **calculate** SVD of X
4: **Compute** Principal Components as:
$PC^{(i)}(t) = \sum_{j=1}^{M}\sum_{l=1}^{L} X_l(t+j-1)\rho_l^K(j)$
5: **Reconstruct** time series by computing:
$R_l^K(t) = \frac{1}{M_t}\sum_{j=L}^{U} PC^k(t-j+1)\rho_l^k(j)$
6: **for** new arrival of data at step k **do**
7: **Update** Trajectory Matrix
8: **Calculate** new Embedded Vector α^T
9: **Calculate** $x_K = \alpha^T P_{k-1}^T$
10: **Find** $P\lambda, P\gamma$ Perturbation matrix corresponding to $(k-1)Q_{k-1} + \alpha_k\alpha_k^T$
11: **Update** eigenvalue and eigenvector matrices as
$\hat{Q}_k = Q_{k-1}(I+P\gamma), \hat{\Sigma}_k = (\Sigma_{k-1}+P\lambda)$
12: **Compute** principal components PC recursively
13: **Reconstruct** components and original time series RC
14: **end for**

Hankelization. To represent time series in the original coordinate system, the principal components are projected back to eigenvectors, also known as reconstructed components of time series (Golyandina et al., 2018). After getting these components, diagonal averaging needs to be done so that the off-diagonal element will have the same average value. Reconstructed time series $X^{(j)}$ is a average of corresponding off-diagonals of matrix $E^{(j)}$, Hankelization operator $\hat{H}$, which represents this operation mathematical is given as $\hat{X}^{(j)} = \hat{H}X^{(j)}$ where $X^{(j)}$ is $L\times K$ matrix and $\hat{X}^{(j)}$ is a Hankel Matrix. The Element $\hat{i}_{p,q}$ in matrix $\hat{X}^{(j)}$ for $r = p+q$ is given by:

$$\hat{i}_{p,q} = \begin{cases} \frac{1}{r+1}\sum_{l=0}^{r}\hat{i}_{l,r-1} & \text{if } 0 \le r \le L-1 \\ \frac{1}{L+1}\sum_{l=0}^{L-1}\hat{i}_{l,r-1} & \text{if } L \le r \le K-1 \\ \frac{1}{K+L-r-1}\sum_{l=r-K+1}^{L}\hat{i}_{l,r-1} & \text{if } K \le r \le K+L-2 \end{cases}$$

After Hankelization, additive components of time series based on the eigenvectors are obtained. A smoother version of the reconstructed time series is extracted by including these additive components, starting from the highest eigenvalue, the next highest, and so on. The cumulative contribution of additive components is calculated, and the number of components reconstructing 99% of the original time series is included. The rest of the reconstructed time series with 1% of components is treated as noise. Finally, a smoother version without outliers is constructed and fed into the CPD Algorithm. The working of RSSA is also demonstrated step-by-step in Algorithm 2 for better understanding.

4.2.3. Change-point detection using overcomplete autoencoder

The data received may be generated from processes that follow an unknown distribution. A particular strategy is required that is not dependent on the prerequisite distribution assumptions. The main advantage of using deep learning is that it can solve complex problems through unsupervised learning and can identify intricate relationships between a large number of interdependent variables (Chalapathy & Chawla, 2019). Deep learning algorithms can build more efficient decision rules by learning hidden patterns from the data. Moreover, the CPD techniques presented in the literature are based on the calculation of the statistical difference between two consecutive segments. This results in a delay in recognizing change-points and is not suitable for real-time applications. The delay is due to the waiting time for the segment data to arrive completely before comparing the statistical difference between the two segments. For instance, in the NPR approach introduced by Rezvani et al. (2019), the time series is divided into segments. Based on the mutual information of three adjacent segments, an approximate change-point is recognized. If there exists a triangle fluctuation, i.e., downward tread followed by upward or vice versa among the three MI of adjacent segments, then middle segment elements are marked as change-points. As a result, the change-points are determined only after calculating the mutual information of three adjacent rows. This causes some delay, which is proportional to the segment size. Hence, finding change-points using segmentation will not be beneficial for applications working in real-time. The deep learning technique is a new way to deal with this problem and can detect change-points without using segmentation (Lattari et al., 2022).

The deep learning-based autoencoders aim at learning input data representation efficiently. Autoencoders are based on unsupervised learning technique. The main objective of the autoencoder is to compress input fed into the network and reconstruct it back by minimizing the reconstruction error as a part of its training. The autoencoder learns to reconstruct the original input using compressed knowledge representation learned during the time of training, such as critical hidden relationships and complicated characteristics of data (Atashgahi et al., 2021). The autoencoder hyper parameters can also be tuned easily so that the complicated input can be reconstructed with minimum error. Furthermore, model over-fitting of autoencoder can be addressed with the help of regularization (Steck & Garcia, 2021). The change-point can easily be detected using the autoencoder by evaluating the magnitude of the reconstruction error. The complete work of the autoencoder is discussed that are divided into two phases, encoding and decoding. Some issues, such as autoencoder training and learning parameter optimization, are also addressed in following sections.

Encoder. The input data **X** of length m having n features is learned and compressed by the encoder. It consists of the input and hidden layer h, also known as the bottleneck, as shown in the Fig. 2. The dimension of the input vector is nxm. An autoencoder's primary function is to compress the input present in the training data. Due to the limitations of the latent space dimension, the network only learns the most significant input features and ignores the rest. If the input features have complex characteristics, the network is unable to capture all the important characteristics. This leads to lossy compression and high reconstruction error (Thies & Alimohammad, 2019). Therefore, in the proposed method, an over-complete autoencoder with a hidden layer dimension larger than the input is used to overcome the issue of lossy compression. The input **X** is stored in the hidden representation

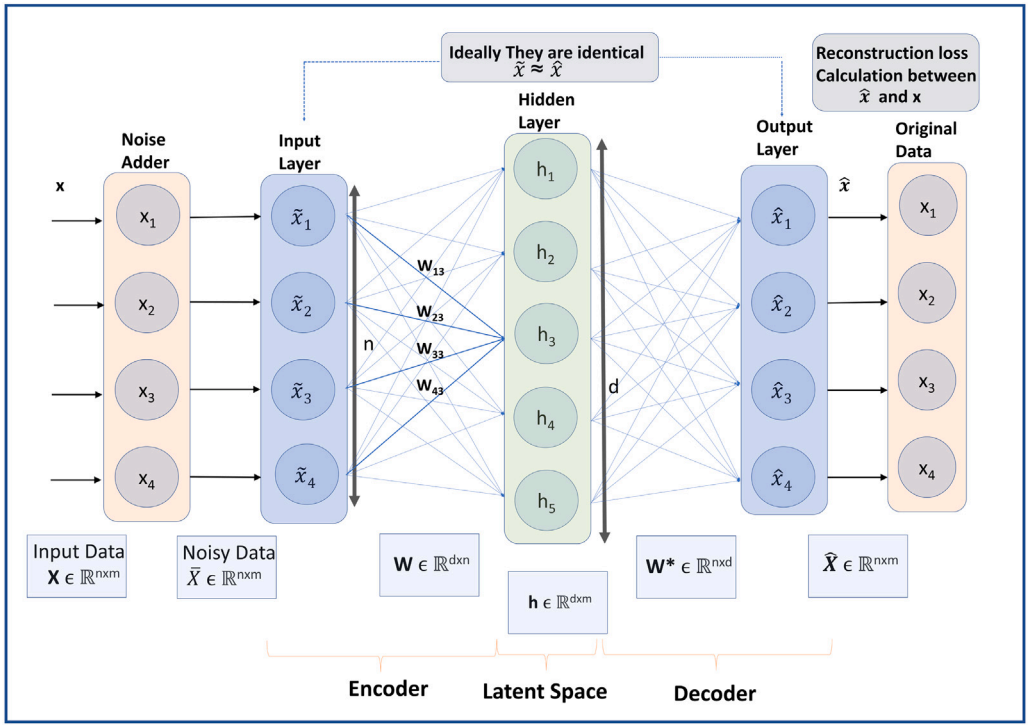

Fig. 2. Autoencoder with noise added to input data, latent space with d neurons, and weights of input layer to hidden layer neuron h3 shown for calculating excitation.

$h = g(WX + b)$, where W is the weight matrix of dimension $d \times n$ and X is the input vector of $n \times m$, b is the bias, and g is the non-linearity function applied get final matrix of dimension $d \times m$. All layers, as well as the output dimensions, are depicted in the Fig. 2.

Decoder. The hidden layer extracts some meaningful information from the input feature and encodes it into latent space. The model attempts to reconstruct the data from the encoded representation based on the learning during the model's training. The calculation of reconstruction loss is the process through which the decoder's performance is evaluated, and the closeness to the original input is determined, as shown in Fig. 2. The model tries to minimize the reconstruction loss error by adjusting weights at the time of back-propagation (Meng et al., 2017). Reconstructing input back from the hidden representation is given by $\hat{X} = f(W^*h + b)$ where $\hat{X}$ is the reconstructed output of dimension $n \times m$, which is same as the input dimension, W^* is the weight learned by the decoder having dimension $n \times d$, and f is the non-linearity function applied to weights and hidden layer to get the final reconstructed output.

The dimension of latent space is larger than that of input space, so the encoder may learn and store more information in order to regenerate a complex output. However, there is a good possibility that the complex input gets transferred directly to the output layer without any learning of the function's inherent characteristics. Some procedures are required to ensure that the overcomplete autoencoder correctly executes the learning task of disentangling a complex function's feature. To address this issue, regularization is performed in the network using the $L2$ optimization framework.

Regularization of autoencoder with L2 norm. Regularization by adding noise is applied to the over-complete autoencoder. Regularization avoids the direct copying of inputs to the outputs without extracting meaningful structure from the hidden layers. The input data is corrupted by using a stochastic process represented as $p(\tilde{x}|x)$ before feeding to the network. Here $\tilde{x}$ is going to be a output when the input x is passed through a noise process. The loss function L of the overcomplete autoencoder can be represented as:

$$L(x, g(f(\tilde{x}))) \tag{21}$$

Here, loss function L penalizes $g(f(\tilde{x}))$ on the basis of L2 norm difference, i.e. mean squared error, if $g(f(\tilde{x}))$ differs from input x. Prior to regularization, the hidden layers relied on one another to grasp the high-level features of input data. However, following regularization, these layers become self-sufficient and can learn to recognize more prominent features straight from the input layer. Because noise in the input effects the learning of a specific hidden layer node during regularization, a complete examination of neural nodes in relation to noisy input is essential for proper network training. The hidden neuron h3 is considered for the analysis of learning, which relies on the impact of noise, as shown in Fig. 2. The hidden layer neuron gets activated for a particular input configuration when the sigmoidal activation function crosses the threshold, which is given by $A_3^{[1]} = g((W^{[1]})^T(\tilde{x}) + B^{[1]})$, where $A_3^{[1]}$ is the activation function for the hidden neuron h_3, g is the non-linearity, $(W^{[1]})^T$ are the weights associated with the hidden layer and the input layer, $\tilde{x}$ is the input added with noise, and $B^{[1]}$ is the biases. Now the activation function $A_3^{[1]}$ will be maximally activated when the value of $(W^{[1]})^T(\tilde{x})$ is maximum. The $\tilde{x}$ value that maximally activates the $A_3^{[1]}$ is calculated as follows:

$$\hat{\tilde{x}} = \arg\max_{\tilde{x}} (W_3^{[1]})^T \tilde{x}$$
$$\text{subject to } \tilde{x}^T\tilde{x} = \|\tilde{x}\|^2 = 1 \tag{22}$$

Here, the objective function is $(W_3^{[1]})^T\tilde{x}$ that is to be maximized by the value of $\tilde{x}$ having a constraint $\tilde{x}^T\tilde{x} = \|\tilde{x}\|^2 = 1$, which is a normalized version of input having unit norm one. Using Lagrange Multiplier function's local maximum and minimum can be determined, provided one or more equations must be exactly fulfilled with the selected values of the variables, Eq. (22) can be written in terms of Lagrangian function as:

$$L(\tilde{x}, \lambda) = (W_3^{[1]})^T\tilde{x} - \lambda\{\tilde{x}^T\tilde{x} - 1\} \tag{23}$$

Here, λ is the Lagrange variable and $(W_3^{[1]})^T\tilde{x}$ is the objective function which is to be maximized. As Lagrangian multiplier is a function of $\tilde{x}$ and λ, to find extrema of L derivative with respect to both $\tilde{x}$ and λ is

calculated as:

$$\frac{\partial L(\tilde{x},\lambda)}{\partial \tilde{x}} = (W_1^{[1]}) - \lambda\tilde{x} = 0$$
$$\tilde{x} = \frac{W_1^{[1]}}{\lambda} \tag{24}$$

and,

$$\frac{\partial L(\tilde{x},\lambda)}{\partial \lambda} = \tilde{x}^T\tilde{x} - 1 = 0$$
$$\tilde{x}^T\tilde{x} = 1 \tag{25}$$

The Lagrangian multiplier can be obtained by plugging $\tilde{x}$ from Eq. (24) into Eq. (25) as:

$$\frac{(W_3^{[1]})^T}{\lambda^T}\frac{W_3^{[1]}}{\lambda} = 1$$
$$\lambda^T\lambda = (W_3^{[1]})^T(W_3^{[1]}) \tag{26}$$
$$\lambda = \sqrt{(W_3^{[1]})^T(W_3^{[1]})}$$

Putting the value of λ in Eq. (24) we get value of $\tilde{x}$ which will excite a hidden neuron h3 which is represented as:

$$\hat{\tilde{x}} = \frac{W_3^{[1]}}{\sqrt{(W_3^{[1]})^T W_3^{[1]}}} \tag{27}$$

Generalizing Eq. (27) for all the neuron present in hidden layer we get:

$$\hat{\tilde{x}}_1 = \frac{W_1^{[1]}}{\sqrt{(W_1^{[1]})^T W_1^{[1]}}}, \hat{\tilde{x}}_2 = \frac{W_2^{[1]}}{\sqrt{(W_2^{[1]})^T W_2^{[1]}}}, \ldots,$$
$$\hat{\tilde{x}}_d = \frac{W_d^{[1]}}{\sqrt{(W_d^{[1]})^T W_d^{[1]}}} \tag{28}$$

Here, $W_1^{[1]}, W_2^{[1]}, \ldots, W_d^{[1]}$ corresponds to a particular d neuron present in hidden layer, that are the vectors of $n \times 1$ where n is the number of features present in input, and $\hat{\tilde{x}}_1, \hat{\tilde{x}}_2, \ldots, \hat{\tilde{x}}_d$ are the particular values of excitation of d neurons in the hidden layer. The Vincent et al. (2008) shows that during learning, feature exaction by the hidden neuron on the MNIST dataset is not significant when input is directly fed without adding noise. As noise increases, more features become informative and dominant. In our proposed methodology also input fed in auto encoder is affected by 50 percent noise so that network training can be done efficiently on the complex inputs.

4.2.4. Change-point detection based on threshold

Initially, some input data n is fed into the autoencoder for training. The models learn to rebuild the output during this process. The neurons in the encoder's hidden layer extract the components, and the mean absolute error between the noise-free input (before regulation) and the reconstructed output is calculated, and the threshold value is chosen using the following equation:

$$T_{val} = \frac{\rho}{100} \times \frac{\|\hat{x}_i - x_i\|}{n} \tag{29}$$

Here, ρ is the tunable parameter that decides the percentage of error taken into consideration and can be changed based on the application to which CPD is applied, and n is the number of features for an input. When the new data points arrive in online mode, the mean absolute error is calculated, and if it exceeds the threshold value, the data point is considered a change-point. Let S be a continuous variable, containing values in the interval [a, b]. If the following rules are satisfied, represented by $R_{T_{val}}$, there will be a change-point in the variable S_i with the threshold T_{val}:

$R_{T_{val}}$: $if\ S(i) \geq T_{val}$, then i is considered as change-point.
$if\ S(i) < T_{val}$, then i is not considered as change-point.

If the value of S(i) crosses threshold values T_{val}, the model starts updating the weights on the following n input data again to detect the next change-point in the time series. The working of our proposed approach has been described using an algorithm 3, which shows all three phases of setup and training step by step.

Algorithm 3 Change-Point Detection Algorithm using deep learning technique

INPUT: TIMESTAMP OF DATA IN ONLINE MODE
OUTPUT: ALL POSSIBLE CHANGE-POINT

Initialize List_ts ← List of timestamps
buffer_ts ← Number of timestamps for initialization
while (Timestamp arrival ! = NULL) **do**
  Append.List_ts (ts_new)
  if (List_ts.**count** = buffer_ts) **then**
    Initialize DAIN, **Train DAIN**← Training Dataset (List_ts)
    List_ts_next ← **Predict DAIN** (List_ts)
    Initialize SSA, Window w ← buffer_ts
    Construct Trajectory Matrix for List_ts
    Compute SVD of Trajectory Matrix to find U_0 Σ_0
    Find the principal components PCs
    Reconstruct Time series Ts , **Update** List_ts
    Initialize Change-Point Detection model
    Noisy_List_ts ← List_ts + 0.5×**Gaussian white noise**
    Train Autoencoder← Training Dataset (Noisy_List_ts)
    Calculate Mean Absolute Error ← (**Predict Autoencoder** (Noisy_List_ts -List_ts)
    Calculate Threshold t ← 0.9 × MAE
  else if (List_ts.**count** > buffer_ts) **then**
    List_ts_next ← **Predict DAIN** (List_ts[-buffer_ts:])
    Calculate Threshold t ← 0.9 × MAE, **Update** Trajectory Matrix
    Calculate PCs using recursive SVD , **Reconstruct** ts
    Calculate Mean Absolute Error ← (**Predict Autoencoder** (ts)
    if (MAE>Threshold t) **then**
      Append.change_point_list ← index of ts as change-point
      List_ts ← 0
    end if
  else
    Continue
  end if
end while

4.3. Complexity analysis of proposed algorithm

The complexity of our proposed algorithm majorly depends on the deep learning algorithm used for normalization, change-point detection, and also calculation of RSSA. The SVD is inherently calculated in the RSSA algorithm, which dominates the time complexity of SSA. Considering N timestamps to initialize or to update weights of the model after CPD detection as buffer timestamps. Training instances t will be N − W + 1, where W is the sliding window size. Step-by-step calculations of computation complexity are as follows:

1. For DAIN having two hidden layer architecture
   (a) In general during Feed-forwarding complexity calculated as $\mathcal{O}((i_n \times h_1) + (i_o \times h_n) + \sum_{i=1}^{n-1} h_i h_{i+1})$ where i_n, i_o, h_n are the number of neuron inputs, outputs and hidden layer respectively. The complexity of feed-forwarding in DAIN network having two hidden layers will be $\mathcal{O}(t \times ((h_1 \times i_n) + (o_n \times h_2) + (h_2 \times h_1)))$.
   (b) For backward propagation, delta weights are computed by multiplying the error matrix with the transpose of activation output, and the complexity will be the same as in the case of feed-forwarding. Total complexity of DAIN

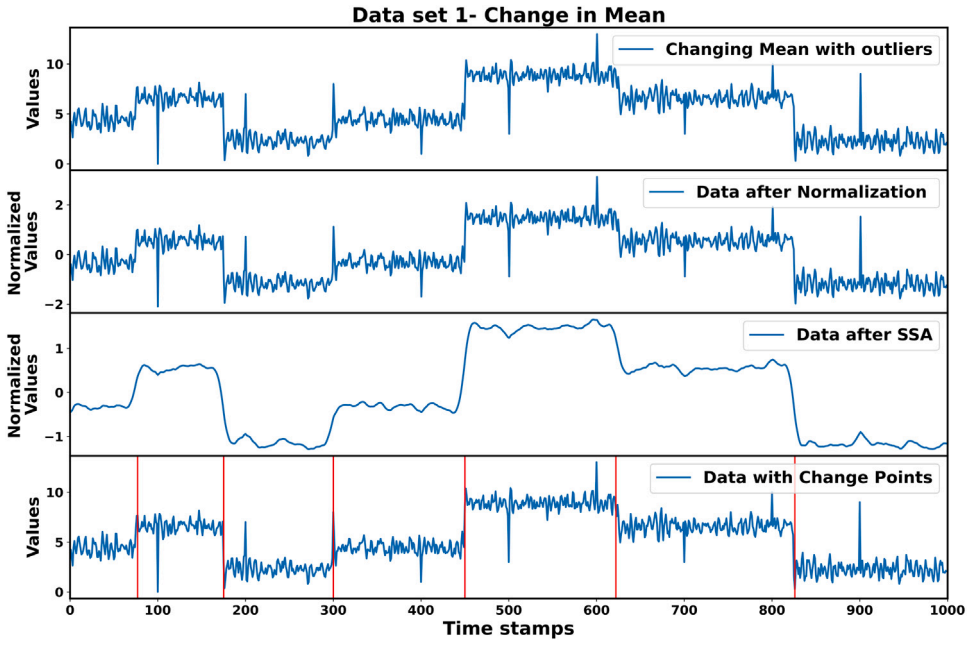

Fig. 3. Row 1 depicts changing mean time series with outliers. Row 2 depicts data after normalization with the normalized value on the y-axis. Row 3 shows data after SSA with outliers removed. Row 4 shows change-points identified by the algorithm.

Table 2
Results of proposed method for real-world datasets.

Performance measure	Formula	Description
Recall	$REC = \frac{TP}{TP+FN}$	Recall measures the algorithm's true positive rate. It is the fraction of the actual change-points that are predicted.
Precision	$PREC = \frac{TP}{TP+FP}$	Precision is the fraction of predicted change-points that are correct.
FP Rate	$FPR = \frac{FP}{FP+TN}$	FP Rate refers to the ratio of data points that are not change-points, but wrongly labeled as change-points to the total number of non change-points. FP Rate also known as false alarm rate.
Accuracy	$ACC = \frac{TP+TN}{TP+FP+TN+FN}$	The accuracy measure indicates the percentage of positive and negative change-points correctly categorized by the algorithm.
F1 Score	$F1 = 2 \times \frac{Precision \times recall}{Precision+recall}$	The F1 score is the harmonic mean of precision and recall and provides a better approximation than accuracy for recognizing incorrectly classified cases when class distribution is uneven.
Timestamp delay	$T_d = \frac{\sum_{i \in TP \ \ class} \lvert \hat{CP}_i - CP_i \rvert}{TP}$	T_d is an expectation of dispersion of the correctly predicted CP from its actual position. It is calculated by taking the weighted sum of the absolute difference between predicted and actual change-point timestamps that belong to the TP Class.

Table 3
Scoring Matrix based on negative root mean square error for hyper parameter tuning using Grid Search.

Dropout rate	Number of Epochs			
	10	20	30	40
0.0	−0.049	−0.065	−0.060	−0.053
0.2	−0.044	−0.059	−0.020	−0.014
0.4	−0.118	−0.083	−0.092	−0.062
0.6	−0.043	**−0.009**	−0.019	−0.045
0.8	−0.700	−0.324	−0.225	−0.142

will be $\mathcal{O}(e \times t \times ((h_1 \times i_n) + (o_n \times h_2) + (h_2 \times h_1)))$. Where e is the number of epochs to train network.

2. Recursive SSA is implemented using Rssa package in R based on Fast Fourier transform that uses Lanczos partial SVD having time complexity of $\mathcal{O}(k \times t \times log(t))$ where k is the components of SVD (Golyandina et al., 2018).
3. Complexity of Autoencoder based on one hidden layer can be calculated in similar way as in DAIN which will be $\mathcal{O}(e \times t \times ((h \times i_a) + (o_a \times h)))$, where i_a, o_a, h are the input, output and hidden layer neurons.

In general $\mathcal{O}(e \times t \times ((h \times i_a) + (o_a \times h)))$ is a dominating term that is nothing but matrix multiplication. According to naive matrix multiplication asymptotic run-time will be given as $\mathcal{O}(t^3)$ where t is the size of training instances (Thottethodi et al., 1998). During testing, the sum of products is done between the hidden layers in the neural network. Therefore, the time complexity will be $\mathcal{O}(h)$ where h is the number of hidden layers in the network. Again the time complexity of SSA Algorithm to calculate SVD on newly arrival data and to reconstruct back to original coordinates will be $\mathcal{O}(k \times t \times log(t) + k^2 t)$. Therefore the total complexity of the CPD Algorithm during the testing phase will be $\mathcal{O}(h + k \times t \times log(t) + k^2 t + h)$ which can be written as $\mathcal{O}(k^2)$. Hence, the time complexity of our CPD Algorithm during training is $\mathcal{O}(t^3)$ and during testing it is $\mathcal{O}(k^2)$.

4.4. Performance measures

A variety of measures are frequently employed to evaluate the performance of CPD methodologies. The performance measures

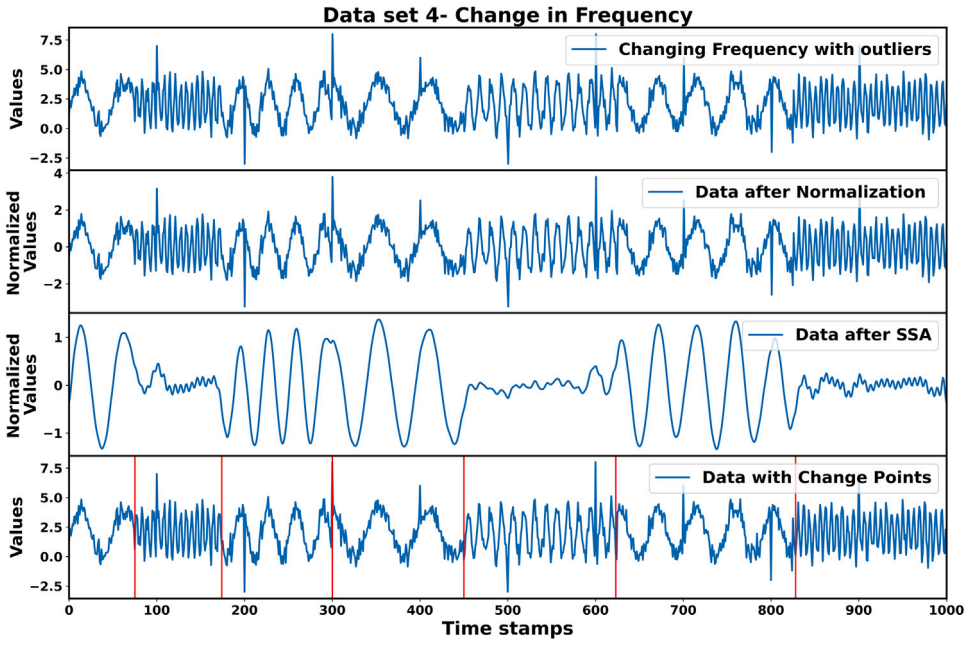

Fig. 4. Row 1 depicts varying frequency time series with outliers. Row 2 depicts data after normalization with the normalized value on the y-axis. Row 3 shows data after SSA with outliers removed. Row 4 shows change-points identified by the algorithm.

Table 4
Results of CUSUM, Bayesian, DRE, NPR and the proposed methods for change in mean, change in variance, and change in mean–variance datasets.

Dataset	Methods	Performance measures					
		REC	PREC	FPR	ACC	F1	T_d
Dataset 1 Changing mean	CUSUM	78.17%	82.17%	0.10%	99.77%	80.12%	0.61 ts
	KLIEP	66.90%	64.83%	0.22%	99.58%	65.85%	3.15 ts
	uLSIF	71.13%	67.62%	0.21%	99.62%	69.33%	3.56 ts
	RuLSIF	75.20%	69.33%	0.20%	99.65%	72.15%	3.01 ts
	Bayesian	71.60%	81.80%	0.10%	99.76%	76.36%	0.24 ts
	NPR	73.63%	33.21%	0.89%	98.95%	45.78%	–
	Proposed method	86.97%	87.73%	0.07%	99.85%	87.35%	1.91 ts
Dataset 2 Changing variance	CUSUM	62.23%	63.16%	0.22%	99.56%	62.69%	0.68 ts
	KLIEP	63.27%	63.52%	0.22%	99.56%	63.39%	2.52 ts
	uLSIF	66.90%	60.05%	0.27%	99.53%	63.29%	3.32 ts
	RuLSIF	70.40%	65.79%	0.22%	99.60%	68.02%	2.80 ts
	Bayesian	66.00%	80.29%	0.10%	99.70%	72.45%	0.33 ts
	NPR	70.27%	32.23%	0.89%	98.94%	44.19%	–
	Proposed method	74.47%	84.53%	0.08%	99.77%	79.18%	1.75 ts
Dataset 3 Changing mean–variance	CUSUM	76.57%	78.93%	0.12%	99.74%	77.73%	0.58 ts
	KLIEP	66.33%	66.51%	0.20%	99.60%	66.42%	3.17 ts
	uLSIF	70.03%	67.75%	0.20%	99.62%	68.87%	3.34 ts
	RuLSIF	73.23%	68.25%	0.21%	99.64%	70.65%	2.92 ts
	Bayesian	63.60%	80.92%	0.09%	99.69%	71.22%	0.39 ts
	NPR	71.83%	32.93%	0.88%	98.95%	45.15%	–
	Proposed method	85.20%	87.24%	0.08%	99.84%	86.21%	1.88 ts

introduced by Keogh et al. (2004) were used to evaluate the proposed algorithm's ability, as shown in the Table 2. The number of correctly labeled change-points is represented by True Positive (TP). The number of change-points that are incorrectly categorized as non-change-points is represented by False Negative (FN). The number of non-change-points that were incorrectly classified as change-points is represented by False Positive (FP) and the number of correctly identified non-change-points is represented by True Negative (TN). The performance measure recall gives the fraction of actual change-points retrieved from the total number of change-points, whereas precision gives the fraction of correctly identified change-points among the total predicted change-points. The Precision and recall are mutually exclusive. If greater coverage is attempted, the precision will be reduced. Similarly, if an attempt is made to improve accuracy, the recall will be reduced. Therefore, the F1 score is chosen, which combines precision and recall into a single metric for better algorithm comparisons. For the real-time application, if the actual change-point is at timestamp $(t + \tau)$ and the detected change-point is within the range of $(t + \tau \pm \lambda)$, then it is considered to be a correctly detected change-point and the value of (λ) varies with the application. For example, in medical monitoring of ECG signals, if the predicted change-point is within the range of $\lambda = \pm 10$ timestamps, it can be treated as a change-point because the intervals between timestamps are in milliseconds, but the same value of $\lambda = \pm 10$ will not be applicable for finding the change in stock price as the timestamp intervals are in days. As a result, Timestamp Delay (T_d) is calculated only for the predicted change-point, which gives a measure of how well predicted change-points align with actual change-points. A lower value indicates that more predicted change-points are aligned with the actual change-point.

Table 5
Results of ratio density estimation based methods and a proposed method for Change in frequency and change in auto-correlation datasets.

Dataset	Methods	Performance measures					
		REC	PREC	FPR	ACC	F1	T_d
Dataset 4 Changing frequency	KLIEP	50.07%	48.77%	0.32%	99.38%	49.41%	2.50 ts
	uLSIF	52.63%	50.30%	0.31%	99.40%	51.44%	2.51 ts
	RuLSIF	63.47%	57.11%	0.29%	99.49%	60.12%	2.12 ts
	Proposed method	62.23%	69.77%	0.16%	99.61%	65.79%	0.99 ts
Dataset 5 Changing auto-correlation	KLIEP	49.93%	47.53%	0.33%	99.37%	48.70%	2.43 ts
	uLSIF	53.37%	52.94%	0.29%	99.44%	53.15%	2.57 ts
	RuLSIF	61.30%	53.99%	0.32%	99.45%	57.41%	2.15 ts
	Proposed method	60.30%	80.58%	0.09%	99.67%	68.98%	0.96 ts

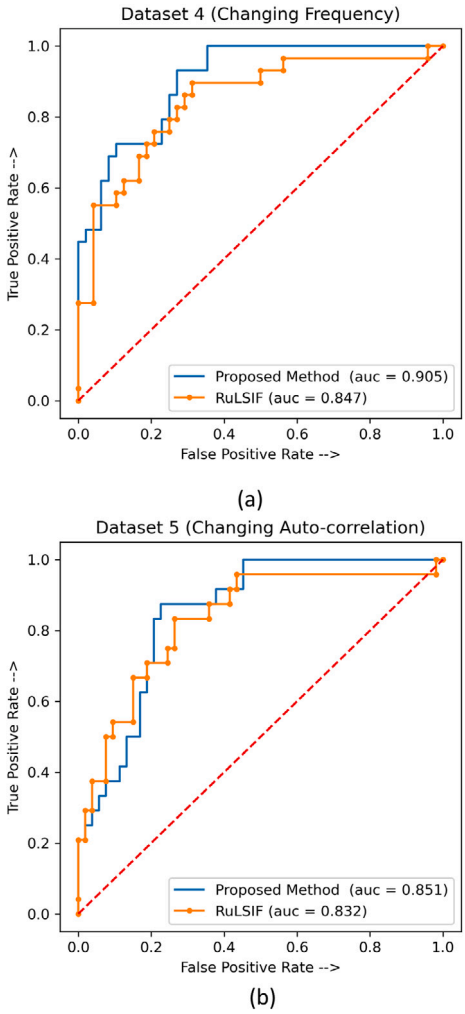

Fig. 5. Part A of the figure shows the AUC of the proposed and RuLSIF methods for changing frequency datasets, and Part B shows AUC of changing auto-correlation datasets.

4.5. Parameter settings

Various parameters tuning are required to obtain optimal efficiency in the CPD algorithms. The DAIN Multi Layer Perceptron (MLP) (Nousi et al., 2019) architecture is chosen for the normalization, which works in an anchored evaluation setup (Tomasini & Jaekle, 2011). To determine the best parameter for our model, hyperparameter tuning is performed on the training set. The GridSearchCV class from the model selection module of sk-learn is used, and the optimal parameters are selected based on the negative root mean squared error scoring matrix (Lafuente et al., 2021). The search space or dictionary is declared with a set of values. In order to determine the ideal hyperparameter values, the grid search is run on all combinations of the declared key values. The values or keys used are the number of epochs, the dropout rate, activation, optimizer, learning rate, and the number of neurons. After grid search, the best parameters are chosen. A fully connected hidden layer of 512 neurons is opted with a REctified Linear Activation Unit (ReLU). A dropout rate of 0.6% is applied after the hidden layer and Adam optimizer with epochs of 20. The Table 3 shows the scoring matrix of dropout rate versus the number of neurons in a network. The Learning Hyper-parameters mentioned in Eq. (7) set to $\eta = 10^{-4}$ (base learning rate) and $\eta_a = 10^{-5}$, $\eta_b = 10^{-3}$, $\eta_c = 10$ and initial weights are set to identity matrix i.e., $W_a = W_b = I_{d\times d}$. For RSSA, the output of DAIN is used to construct trajectory matrix X. For every new arrival of data after buffer timestamps, matrix X gets updated. Memory depth parameter $\lambda_k = 1/k$ is set to 0.2 and initial 50 timestamps are used to initialize P_0 and Q_0 to obtain eigendecomposition of covariance matrix X_0. An overcomplete autoencoder is considered with a latent space of 200 LSTM units fully interconnected and a used batch size of 32 with a validation split of 0.1 to train the model. The learning rate of the hyper-parameter η set to 10^4. The threshold value described in Eq. (29) is set to 90% of MAE, which is obtained after training the model.

5. Results and discussion

The performance of the proposed CPD method is compared with well-known and widespread baseline models such as the CUmulative SUM method (CUSUM) (Otto & Breitung, 2020), Bayesian methods (Agudelo-España et al., 2020), Direct-Ratio Estimation such as KLIEP (Liu et al., 2013), uLSIF (Khan et al., 2019), RuLSIF (Hushchyn & Ustyuzhanin, 2021) and also the recently proposed NPR CPD algorithm based on mutual information (Rezvani et al., 2019). To compare our proposed method to offline methods such as CUSUM and NPR algorithms, complete data is first preprocessed and then provided to the CPD module for results. The results for the synthetic data, which are changing mean (Dataset 1), changing variance (Dataset 2), and changing mean and variance (Dataset 3), are shown in the Table 4. All datasets consist of 500 samples of time series with 6 change-points in each sample. Therefore, a total of 3000 change-points are present in each dataset. Our proposed approach outperforms in all datasets with the highest precision in detecting change-points. The proposed algorithm detected 2609 (true positive cases) for changing the mean, 2234 for changing variance, and 2556 for changing mean–variance. Our algorithm achieved the highest recall for dataset 1 and dataset 3. For dataset 2, an algorithm based on RuLSIF has a slightly greater recall, but if we compare other performance metrics, our algorithm is better than RuLSIF. One sample of the changing mean with change-point detection has been shown in the Fig. 3. The false-positive rate of the NPR methods increases drastically due to the assumption that all the points in a segment are change-points, as previously described in Section 4.2.3. This increases the recall of the algorithm but decreases the precision. The false-positive cases of our algorithm are 365, 409, and 374 for datasets 1, 2, and 3, respectively. To identify the change-point, the CUSUM algorithm iteratively computes a cumulative sum

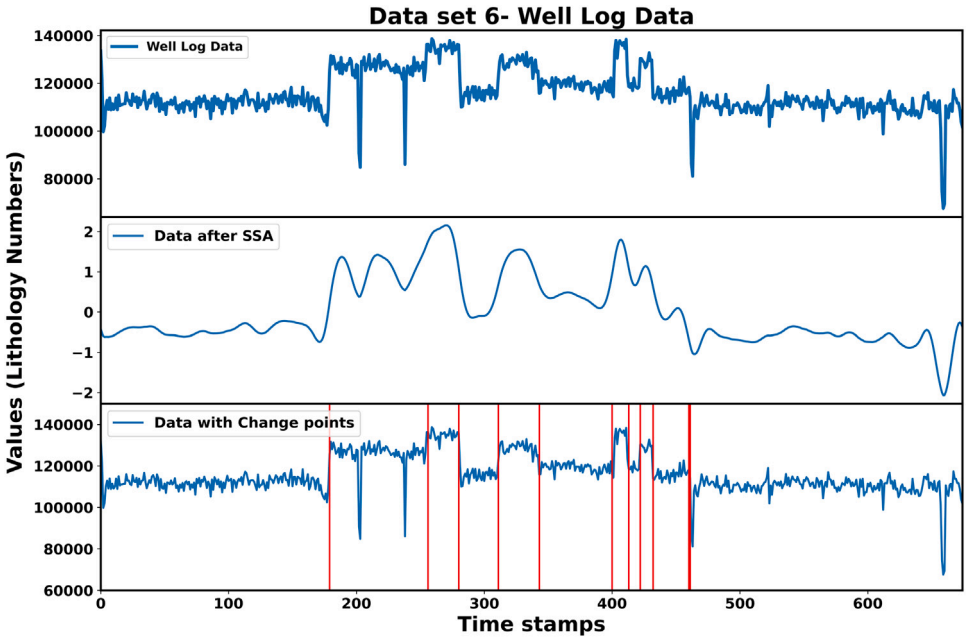

Fig. 6. Row 1 contains real-world well log data time series with outliers. Row 2 depicts data after normalization and SSA with the normalized value on the y-axis and outliers eliminated. Row 3 shows change-points found by the algorithm.

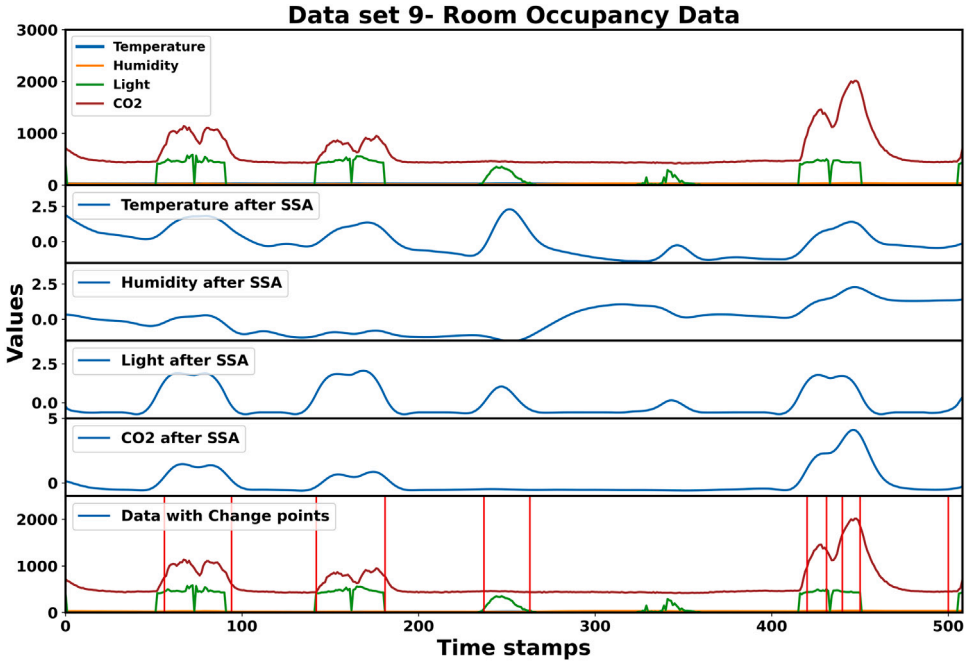

Fig. 7. Row 1 contains multivariate real-world room occupancy data having four variables. Rows 2–5 depict data after normalization and SSA of all four variables with the normalized value on the y-axis and outliers eliminated. Row 6 shows change-points identified by the algorithm.

chart and performs bootstrapping on the entire data set. Similarly, the Bayesian algorithm calculates the predictive probability of each possible previous run length to determine whether the new timestamp is a change-point. Due to the repetitive calculations on entire datasets, the predicted change-points of these algorithms generally align (close in terms of timestamps) with the actual change-point. As the Timestamp Delay (T_d) measures how close the predicted change-point is to the actual change-point, these methods have less T_d as compared to the proposed method. However, both the CUSUM and Bayesian algorithms have quadratic time complexity (Romano et al., 2021), making them substantially more computationally expensive than the proposed approach. Moreover, overall performance to detect change-points can be calculated using the F1 Score, which gives a measure of correct prediction. Our algorithm has the highest F1 score as compared to all the state-of-the-art algorithms. One of the objectives of our algorithm is to decrease the FP Rate, which has been achieved successfully, thus minimizing the Type I error.

Since CUSUM, Bayesian, and NPR approaches do not address frequency and auto-correlation changes in time series and hence fail to detect these complicated changes, the proposed methodology is compared to KLIEP, uLSIF, and RuLSIF ; the performance metrics for datasets 4 (Changing Frequency) and 5 (Changing Auto-correlation) are listed separately in the Table 5. The proposed algorithm detected 1867

Table 6
Results of proposed method for real-world datasets.

Dataset	Methods	Performance measures					
		REC	PREC	FPR	ACC	F1	T_d
Dataset 6 Well Log	CUSUM	60.00%	75.00%	0.30%	99.11%	66.67%	1.90 ts
	RuLSIF	70.00%	77.78%	0.30%	99.26%	73.68%	2.46 ts
	Bayesian	90.00%	90.00%	0.15%	99.70%	90.00%	1.73 ts
	Proposed method	100.00%	100.00%	0.00%	100.00%	100.00%	2.03 ts
Dataset 7 Apple stock	CUSUM	87.50%	70.00%	0.49%	99.36%	77.78%	0.94 ts
	RuLSIF	75.00%	60.00%	0.65%	99.04%	66.67%	1.37 ts
	Bayesian	87.50%	70.00%	0.49%	99.36%	77.78%	0.89 ts
	Proposed method	87.50%	77.78%	0.33%	99.52%	82.35%	1.03 ts
Dataset 8 CO2 emission	CUSUM	71.43%	62.50%	1.45%	97.66%	66.67%	1.56 ts
	RuLSIF	71.43%	55.56%	1.93%	97.20%	62.50%	2.88 ts
	Bayesian	85.71%	60.00%	1.93%	97.66%	70.59%	1.27 ts
	Proposed method	100.00%	87.50%	0.48%	99.53%	93.33%	1.71 ts
Dataset 9 Room occupancy	CUSUM	72.73%	66.67%	0.80%	98.62%	69.57%	0.83 ts
	RuLSIF	81.82%	75.00%	0.60%	99.02%	78.26%	1.37 ts
	Bayesian	90.91%	83.33%	0.40%	99.41%	86.96%	0.82 ts
	Proposed method	100.00%	84.62%	0.40%	99.61%	91.67%	1.09 ts

and 1809 True Positive cases out of 3000 in the frequency and auto-correlation datasets, respectively. A sample of time series with changing frequency is shown in the Fig. 4. The RuLSIF has higher recall than the proposed methods, but poorer precision in both datasets, hence its F1 score is lower than the proposed method. Furthermore, the Timestamp Delay (T_d) of RuLSIF is much greater as compared with the proposed method. The Area Under Curve (AUC) is calculated, which represents two levels of separability, as an extra measure to compare our method to RuLSIF's. The results show (refer to Fig. 5) that our algorithm has more AUC for both datasets. Therefore, our model has high efficiency in detecting change-points with good precision and less Timestamp Delay (T_d) in the case of auto-correlation data compared to frequency change data.

The performance metrics of all real-world datasets are presented in the Table 6. The proposed method is compared to CUSUM, RuLSIF, and Bayesian as these algorithms are capable of finding change-points in a multivariate time series. The KLIEP, uLSIF, and NPR techniques only find changes in the univariate time series. The recall of our algorithm is highest in all cases. In the well log dataset (dataset 6), our algorithm detected all the change-points correctly with an FP rate of zero, as shown in Fig. 6. The F1 score of our algorithm is also the highest as compared to all other algorithms. The Timestamp Delay (T_d) of our algorithm is slightly higher as compared with CUSUM and Bayesian but lower than RuLSIF. The false positive rate of the proposed algorithm is the lowest as compared with other algorithms. Finally, our algorithm outperforms for the room occupancy dataset by correctly detecting 11 out of 11 change-points shown in the Fig. 7 and has a recall of 100%.

6. Conclusion and future perspectives

This paper introduces a fully end-to-end adaptive method to detect a time series change-point that can be used for various real-time applications across many domains. In this paper, the accuracy of change-point detection algorithms is shown to be dependent on the data provided in the detector module. Therefore, the need for data preprocessing is necessary for CPD methods that are based on deep learning techniques. This work primarily focuses on the smooth working of algorithms for real-time applications. Various measures have been implemented to ensure that data must be preprocessed simultaneously and delivered to the CPD module for detection as soon as possible. The well-known existing methodologies such as direct density ratio estimation or Bayesian statistics-based change-point detection are present in the literature. These methodologies can detect change-points but are less efficient because, in these methods, preprocessing of data is not a part of the CPD module. The data is preprocessed first, and then the entire data is passed to the CPD model, resulting in delays in detecting change-points, making the algorithm inefficient for real-time scenarios. In this paper, all these issues are resolved by introducing a novel three-phase architecture that links preprocessing phases to the change-point phase. In preprocessing, adaptive normalization is employed in the first phase for better training of the CPD module based on deep learning techniques. In the second phase, a recursive version of singular spectrum analysis is developed to feed a smoother version of noisy-free data to the CPD module. The third phase is to detect the change-point on the fly. Our CPD module consists of an autoencoder neural net structure that efficiently learns input data characteristics during training. The CPD module successfully detects any kind of change-points in a time series based on the change in the mean, variances, frequency, or auto-correlation. The proposed algorithm outperforms the existing method in the experimental validation of the synthetic and real-world dataset. Change-points for synthetic datasets are also detected, having a change in frequency, and auto-correlation, which are entirely ignored by the existing state-of-the-art algorithms. The results of these datasets show that our algorithm can detect these complex changes successfully.

Even if the given technique has been shown to be beneficial in most situations, its effectiveness may increase further. The higher the Timestamp Delay (T_d), the higher the false alarm rate, which can be reduced by efficient training of the model so that change-points can be recognized within the acceptable range of Timestamp Delay (T_d). The performance of CPD for data having changes in frequency and auto-correlation can be improved by increasing recall and precision. More improvements can be made by lowering the number of buffer timestamps required to initialize the method, allowing the model to be trained with fewer timestamps. Another key area of future research is to increase the algorithm's computing efficiency. If the cost of computing goes down, adaptive preprocessing will be easier to use in real-world applications.

CRediT authorship contribution statement

Muktesh Gupta: Investigation, Methodology, Validation, Writing – original draft. **Rajesh Wadhvani:** Conceptualization, Writing – review & editing, Supervision, Validation. **Akhtar Rasool:** Conceptualization, Writing – review & editing, Supervision, Validation, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We would like to thank the Alan Turing Institute, founded by the Government of the United Kingdom, for open sourcing their work on Turing Change-Point. The dataset in the repository was explicitly collected for the evaluation of change-point detection algorithms on real-world data.

References

Abdi, H. (2007). Singular value decomposition (SVD) and generalized singular value decomposition. *Encyclopedia of Measurement and Statistics*, 907–912.

Adams, R. P., & MacKay, D. J. (2007). Bayesian online changepoint detection. arXiv preprint arXiv:0710.3742.

Agudelo-España, D., Gomez-Gonzalez, S., Bauer, S., Schölkopf, B., & Peters, J. (2020). Bayesian online prediction of change points. In *Conference on uncertainty in artificial intelligence* (pp. 320–329). PMLR.

Aminikhanghahi, S., & Cook, D. J. (2017). A survey of methods for time series change point detection. *Knowledge and Information Systems*, *51*(2), 339–367.

Atashgahi, Z., Mocanu, D. C., Veldhuis, R., & Pechenizkiy, M. (2021). Unsupervised online memory-free change-point detection using an ensemble of LSTM-autoencoder-based neural networks. In *8th ACM celebration of women in computing womencourage*.

Auret, L., & Aldrich, C. (2010). Change point detection in time series data with random forests. *Control Engineering Practice*, *18*, 990–1002. http://dx.doi.org/10.1016/j.conengprac.2010.04.005.

Bermejo, U., Almeida, A., Bilbao-Jayo, A., & Azkune, G. (2021). Embedding-based real-time change point detection with application to activity segmentation in smart home time series data. *Expert Systems with Applications*, *185*, Article 115641.

Braun, J. V., Braun, R., & Müller, H.-G. (2000). Multiple changepoint fitting via quasilikelihood, with application to DNA sequence segmentation. *Biometrika*, *87*(2), 301–314.

Camci, F. (2010). Change point detection in time series data using support vectors.. *International Journal of Pattern Recognition and Artificial Intelligence*, *24*, 73–95. http://dx.doi.org/10.1142/S0218001410007865.

Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407.

Claessen, D., & Groth, A. (2002). A beginner's guide to SSA. In *CERES*. Paris, France: Ecole Normale Supérieure.

Darkhovsky, B., & Piryatinska, A. (2018). Model-free offline change-point detection in multidimensional time series of arbitrary nature via -complexity: Simulations and applications. *Applied Stochastic Models in Business and Industry*, *34*(5), 633–644.

Deryck, T., De Vos, M., & Bertrand, A. (2021). Change point detection in time series data using autoencoders with a time-invariant representation. *IEEE Transactions on Signal Processing*.

Desobry, F., Davy, M., & Doncarli, C. (2005). An online kernel change detection algorithm. *IEEE Transactions on Signal Processing*, *53*(8), 2961–2974.

Dhekane, S. G., Tiwari, S., Sharma, M., & Banerjee, D. S. (2022). Enhanced annotation framework for activity recognition through change point detection. In *2022 14th international conference on communication systems networks (COMSNETS)* (pp. 397–405). http://dx.doi.org/10.1109/COMSNETS53615.2022.9668475.

Ebrahimzadeh, Z., Zheng, M., Karakas, S., & Kleinberg, S. (2019). Deep learning for multi-scale changepoint detection in multivariate time series. arXiv preprint arXiv:1905.06913.

Eesa, A. S., & Arabo, W. K. (2017). A normalization methods for backpropagation: a comparative study. *Science Journal of University of Zakho*, *5*(4), 319–323.

Fearnhead, P., & Rigaill, G. (2019). Changepoint detection in the presence of outliers. *Journal of the American Statistical Association*, *114*(525), 169–183.

Feuz, K. D., Cook, D. J., Rosasco, C., Robertson, K., & Schmitter-Edgecombe, M. (2014). Automated detection of activity transitions for prompting. *IEEE Transactions on Human-Machine Systems*, *45*(5), 575–585.

Golyandina, N., Korobeynikov, A., & Zhigljavsky, A. (2018). *Singular spectrum analysis with R*. Springer.

Gupta, V. (2015). Speaker change point detection using deep neural nets. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 4420–4424). IEEE.

Hassani, H., & Mahmoudvand, R. (2013). Multivariate singular spectrum analysis: A general view and new vector forecasting approach. *International Journal of Energy and Statistics*, *1*(01), 55–83.

Hinkley, D., & Schechtman, E. (1987). Conditional bootstrap methods in the mean-shift model. *Biometrika*, *74*(1), 85–93.

Hurvich, C. M., & Tsai, C.-L. (1989). Regression and time series model selection in small samples. *Biometrika*, *76*(2), 297–307.

Hushchyn, M., & Ustyuzhanin, A. (2021). Generalization of change-point detection in time series data based on direct density ratio estimation. *Journal of Computer Science*, *53*, Article 101385.

Jin, B., Chen, Y., Li, D., Poolla, K., & Sangiovanni-Vincentelli, A. (2019). A one-class support vector machine calibration method for time series change point detection. arXiv:1902.06361.

Kato, M., Imaizumi, M., & Minami, K. (2022). Unified perspective on probability divergence via maximum likelihood density ratio estimation: Bridging KL-divergence and integral probability metrics. arXiv preprint arXiv:2201.13127.

Katser, I., Kozitsin, V., Lobachev, V., & Maksimov, I. (2021). Unsupervised offline changepoint detection ensembles. *Applied Sciences*, *11*(9), 4280.

Kawahara, Y., Yairi, T., & Machida, K. (2007). Change-point detection in time-series data based on subspace identification. In *Seventh IEEE international conference on data mining (ICDM 2007)* (pp. 559–564). IEEE.

Keogh, E., Chu, S., Hart, D., & Pazzani, M. (2004). Segmenting time series: A survey and novel approach. In *Data mining in time series databases* (pp. 1–21). World Scientific.

Khan, H., Marcuse, L., & Yener, B. (2019). Deep density ratio estimation for change point detection. arXiv preprint arXiv:1905.09876.

Koepcke, L., Ashida, G., & Kretzberg, J. (2016). Single and multiple change point detection in spike trains: comparison of different CUSUM methods. *Frontiers in Systems Neuroscience*, *10*, 51.

Lafuente, D., Cohen, B., Fiorini, G., García, A. A., Bringas, M., Morzan, E., & Onna, D. (2021). A gentle introduction to machine learning for chemists: an undergraduate workshop using python notebooks for visualization, data processing, analysis, and modeling. *Journal of Chemical Education*, *98*(9), 2892–2898.

Lattari, F., Rucci, A., & Matteucci, M. (2022). A deep learning approach for change points detection in InSAR time series. *IEEE Transactions on Geoscience and Remote Sensing*, 1. http://dx.doi.org/10.1109/TGRS.2022.3155969.

Liu, J., Liu, X., & Ma, X. (2008). First-order perturbation analysis of singular vectors in singular value decomposition. *IEEE Transactions on Signal Processing*, *56*(7), 3044–3049.

Liu, S., Yamada, M., Collier, N., & Sugiyama, M. (2013). Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, *43*, 72–83.

Luong, T. M., Perduca, V., & Nuel, G. (2012). Hidden Markov model applications in change-point analysis. arXiv:1212.1778.

Maleki, S., Maleki, S., & Jennings, N. R. (2021). Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering. *Applied Soft Computing*, *108*, Article 107443.

Mastrantonio, G., Jona Lasinio, G., Pollice, A., Teodonio, L., & Capotorti, G. (2022). A Dirichlet process model for change-point detection with multivariate bioclimatic data. *Environmetrics*, *33*(1), Article e2699.

Meng, Q., Catchpoole, D., Skillicom, D., & Kennedy, P. J. (2017). Relational autoencoder for feature extraction. In *2017 international joint conference on neural networks (IJCNN)* (pp. 364–371). IEEE.

Mohammad-Djafari, A., & Féron, O. (2006). A Bayesian approach to change points detection in time series. *International Journal of Imaging Systems and Technology*, *16*, 215–221. http://dx.doi.org/10.1002/ima.20080.

Nalmpantis, A., Passalis, N., Tsantekidis, A., & Tefas, A. (2021). Deep adaptive group-based input normalization for financial trading. *Pattern Recognition Letters*, *152*, 413–419.

Nousi, P., Tsantekidis, A., Passalis, N., Ntakaris, A., Kanniainen, J., Tefas, A., Gabbouj, M., & Iosifidis, A. (2019). Machine learning for forecasting mid-price movements using limit order book data. *IEEE Access*, *7*, 64722–64736.

Otto, S., & Breitung, J. (2020). Backward CUSUM for testing and monitoring structural change. arXiv preprint arXiv:2003.02682.

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2019). Deep adaptive input normalization for time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(9), 3760–3765.

Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., & Srivastava, M. (2010). Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks*, *6*(2), 1–27.

Reeves, J., Chen, J., Wang, X. L., Lund, R., & Lu, Q. Q. (2007). A review and comparison of changepoint detection techniques for climate data. *Journal of Applied Meteorology and Climatology*, *46*(6), 900–915.

Rezvani, R., Barnaghi, P., & Enshaeifar, S. (2019). A new pattern representation method for time-series data. *IEEE Transactions on Knowledge and Data Engineering*.

Romano, G., Eckley, I., Fearnhead, P., & Rigaill, G. (2021). Fast online changepoint detection via functional pruning CUSUM statistics. arXiv preprint arXiv:2110.08205.

Saatçi, Y., Turner, R. D., & Rasmussen, C. E. (2010). Gaussian process change point models. In *ICML*.

Selvi, S., Bala Nivetha, V., Divya, R., Gayathri, S., & Pavithra, G. (2021). Change point detection technique for weather forecasting using Bi-LSTM and 1D-CNN algorithm. In *Next generation of internet of things* (pp. 109–119). Springer.

Shi, Z., & Chehade, A. (2021). A dual-LSTM framework combining change point detection and remaining useful life prediction. *Reliability Engineering & System Safety*, *205*, Article 107257.

Sola, J., & Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, *44*(3), 1464–1468.

Steck, H., & Garcia, D. G. (2021). On the regularization of autoencoders. arXiv preprint arXiv:2110.11402.

Taylor, W. A. (2000). Change-point analysis: a powerful new tool for detecting changes.

Thies, J., & Alimohammad, A. (2019). Compact and low-power neural spike compression using undercomplete autoencoders. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *27*(8), 1529–1538.

Thottethodi, M., Chatterjee, S., & Lebeck, A. R. (1998). Tuning strassen's matrix multiplication for memory efficiency. In *SC'98: Proceedings of the 1998 ACM/IEEE conference on supercomputing* (p. 36). IEEE.

Tomasini, E., & Jaekle, U. (2011). *Trading systems*. Harriman House Limited.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Wang, J. (2016). A supervised topic model based approach for change point detection in review data.

Wu, H., & Matteson, D. S. (2020). Adaptive Bayesian changepoint analysis and local outlier scoring. arXiv preprint arXiv:2011.09437.

Xie, Y., & Siegmund, D. (2013). Sequential multi-sensor change-point detection. In *2013 information theory and applications workshop (ITA)* (pp. 1–20). IEEE.

Yoo, S., Jeon, S., Jeong, S., Lee, H., Ryou, H., Park, T., Choi, Y., & Oh, K. (2021). Prediction of the change points in stock markets using DAE-LSTM. *Sustainability*, *13*(21), 11822.

Zhou, Y. (2019). Change point detection based on directed K-nearest neighbour graph.