Time2State: An Unsupervised Framework for Inferring the Latent States in Time Series Data

CHENGYU WANG, National University of Defense Technology, China
KUI WU, University of Victoria, Canada
TONGQING ZHOU*, National University of Defense Technology, China
ZHIPING CAI*, National University of Defense Technology, China

Time series data from monitoring applications reflect the physical or logical states of the objects, which may produce time series of distinguishable characteristics in different states. Thus, time series data can usually be split into different segments, each reflecting a state of the objects. These states carry rich high-level semantic information, e.g., run, walk, or jump, which helps people better understand the behaviour of the monitored objects. Nevertheless, these states are latent and hard to discover, because the characteristic of time series is complicated and the computational cost is high. This paper develops an efficient and effective unsupervised approach for inferring the latent states of massive multivariate time data. To reduce the computational cost, we present Time2State, a scalable framework that utilizes a sliding window and an encoder to greatly reduce the length of raw time series. To train the encoder, we propose a novel unsupervised loss function, LSE-Loss. Extensive experiments show that compared to the state-of-the-art time series representation learning methods of the same kind, LSE-Loss brings a performance improvement of up to 15% in accuracy.

CCS Concepts: • **Information systems → Data mining**.

Additional Key Words and Phrases: Time series segmentation, time point clustering, time series representation learning

ACM Reference Format:
Chengyu Wang, Kui Wu, Tongqing Zhou, and Zhiping Cai. 2023. Time2State: An Unsupervised Framework for Inferring the Latent States in Time Series Data. *Proc. ACM Manag. Data* 1, 1, Article 17 (May 2023), 18 pages. https://doi.org/10.1145/3588697

1 INTRODUCTION

Many applications, ranging from autonomous driving to wearable sensing, constantly generate a large amount of time series data [16]. These time series carry rich information capturing the state changes of monitored/interested objects. It is thus critical to develop tools for segmenting a time series into different segments that reflect the object states. The exploitation of these latent states lays the foundation of many real-world tasks, e.g., pattern recognition, event detection, anomaly detection [16, 23].

*Corresponding author.

This work is supported by National Natural Science Foundation of China (62072465, 62102425), Science and Technology Innovation Program of Hunan Province (2022RC3061, 2021RC2071), and NUDT Research Grants (ZK19-38).
Authors' addresses: Chengyu Wang, chengyu@nudt.edu.cn, National University of Defense Technology, Changsha, China; Kui Wu, wkui@uvic.ca, University of Victoria, Victoria, Canada; Tongqing Zhou, zhoutongqing@nudt.edu.cn, National University of Defense Technology, Changsha, China; Zhiping Cai, zpcai@nudt.edu.cn, National University of Defense Technology, Changsha, China.

2836-6573/2023/5-ART17 $15.00
https://doi.org/10.1145/3588697

An object usually has multiple different states that exhibit quite different data patterns in practice. For instance, an automobile can take different actions, such as turning, driving straight, braking, and accelerating, each generating a distinguishable pattern in time series [9, 14, 16, 18, 23]. When the physical or logical state of the object transits, such transition can be reflected by the pattern change of the corresponding time series. As illustrated in Fig.1, the upper subfigure shows a 4-channel time series data corresponding to the left/right arms and left/right legs, from the MoCap [2] dataset. The lower subfigure shows the underlying state sequence of the monitored object. We can see that the state transitions are clearly reflected by the pattern changes in the raw time series. Our goal is to infer the latent state sequence from a given time series.

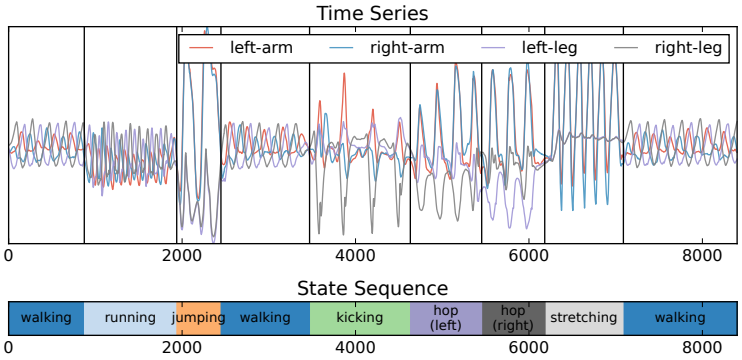


Fig. 1. Time series segmentation for state inference.

A practically meaningful solution to the above problem should be scalable in handling large time series because most data-intensive applications involve colossal data. In addition, the solution should be unsupervised because we usually do not know the number of classes in advance or do not have enough labelled data. Nevertheless, existing unsupervised methods generally suffer from low scalability. As we will see in § 5.3, for example, the fastest existing method takes about 3.18 minutes to segment a four-dimensional time series with a length of 200k. Such processing time is unacceptable in many data-intensive scenarios. This paper fills this gap by developing an unsupervised and scalable solution, called Time2State, for inferring the latent states in massive time series data. The major challenges that we are faced with include:

- Labeling time series is a labor-intensive task [41], and the time series in the same dataset can have diverse characteristics, e.g., different statistical moments and shapes. It is extremely challenging for researchers to label massive heterogeneous time series data and select a proper model for every time series. Thus the framework must be **unsupervised** and **general**.
- Without labelled data, it is usually hard to know the number of states in advance. Thus the framework must be **effective** even though the number of states is unavailable. In other words, it should be **adaptive** through self-learning the number of states from data.
- Typical IoT applications usually generate millions of multivariate time series, each of which has a considerable length (usually up to 100k). Thus the framework must be **scalable** and **efficient** enough to work on massive and long data.

To meet the first requirement, we propose a novel self-supervised loss function that uses a sliding window to automatically learn distinguishable representations (a.k.a., features or embeddings in literature) from general time series that may exhibit different characteristics over time. To meet the second requirement, we adopt non-parametric Bayesian methods to estimate the number of states from the learned representations. To meet the third requirement, we leverage the proposed

loss function to train an encoder, which maps raw time series into a low-dimensional embedding sequence. Each embedding in the sequence is generated based on the smaller data amount within the sliding window. In this way, we can significantly reduce the computation time.

Our contributions are summarized as follows:

- We propose a scalable method, **Time2State**[1], which flexibly combines temporal correlation, self-supervised representation learning, and clustering for inferring the latent states of time series data.
- We propose a novel loss function, **LSE-Loss**, which can effectively learn distinguishable representations with the time series data in a sliding window. With LSE-Loss, we can effectively learn a sequence of embeddings when the sliding window moves across the entire time series.
- We perform extensive experiments with real-world datasets to demonstrate the advantages of Time2State. It outperforms the baselines with respect to effectiveness, scalability, and generality.

2 RELATED WORK

2.1 Time Series Segmentation

While time series segmentation has been broadly studied, we mainly focus on time series segmentation for state detection. In this context, existing methods can be divided into supervised [26, 39] and unsupervised [16, 23, 25] methods, but we only focus on unsupervised methods due to the reasons in the introduction.

Unsupervised methods can be further divided into two categories: those requiring the number of classes as input and those without. As a representative work of the first category, TICC [16] is a Toeplitz inverse covariance-based method. Each cluster is defined by a correlation network, which characterizes the dependencies between different observations in a typical subsequence of that cluster. Despite the promising results, these methods rely heavily on prior knowledge. If the number of classes (denoted as "K" hereafter) is not specified in advance, these methods cannot even work. This is a serious problem because K is usually unavailable in practice. To the best of our knowledge, it is difficult to add patches for these methods to adapt K. On the contrary, the second category is more practical but is rarely studied. AutoPlait [23] uses the Minimum description length (MDL) [15] to estimate the number of classes. HVGH [25] combines Variational Auto Encoder (VAE) [19] and Hierarchical Dirichlet process (HDP) [32] to estimate the number of clusters. Although they do not require specifying K, AutoPlait requires multiple rounds of optimization to greedily optimize an MDL-based loss, which leads to unstable running time and lower scalability, and HVGH requires multiple epochs of mutual learning between the VAE and HDP, which greatly weakens its scalability. Time2State differs from these methods in that it directly estimates K from the learned representations, which is more effective and efficient.

There are also some segmentation methods, e.g., ClaSP [28], FLOSS [14], and so on [9, 20], that is designed to detect change points. Despite the promising results they achieved in detecting change points, they are still far away from state detection. As explained in [16], state detection is more difficult because it requires *simultaneous* segmentation and clustering. Simply clustering the unstated segmentation results, even after deliberate alignment for overcoming distinct segment lengths, would generally lead to poor results (see § 5.2.3 for ClaSP+deliberate clustering). This is because there are usually phase differences or amplitude (value) drifts between different segmentations of the same category after time series segmentation. Such differences would enlarge the inner distances of segmentations in one cluster, causing obscure margins between clusters and poor clustering performance. To summarize, it is non-trivial, if not impossible, to modify/tailor these

[1]Code is available at https://github.com/Lab-ANT/Time2State

methods via straightforward technique combinations for state detection. Time2State differs from these methods in that it requires and involves no segmentation phase and its clustering process takes the embeddings of sliding windows, instead of segments, as input. By inversely parsing latent states from clusters, it avoids the segmentation pitfall of large inner-distance for clustering.

2.2 Time Series Representation Learning

Representation learning uses encoders to learn representations from raw data and maps them into a low-dimensional embedding space. There are dozens of time series representation learning methods [10, 12, 13, 22, 31, 33, 40], but very few consider segmentation as a downstream task. Their target downstream tasks are either classification or prediction. Hence, they pay more attention to the *overall* similarity of data. In other words, they prefer to learn instance-level representations that capture the overall similarity, which makes them not suitable for segmentation. Among those methods that consider segmentation as a downstream task, Triplet-Loss [13] and TNC [33] are two state-of-the-art methods. They are both within the scope of contrastive learning, which learns representations through comparing positive and negative samples. Triplet-Loss trains the encoder by maximizing the distance between an anchor sample and negative samples and minimizing the distance between the anchor samples and positive samples. TNC trains the encoder by distinguishing neighbour and non-neighbour samples through an extra discriminator. The biggest pitfall of Triplet-Loss and TNC is that they only consider pair-wise distance, with which they tend to learn a malformed embedding space structure. LSE-Loss differs from them in that it considers both intra-state and inter-state distance, and thus it can learn more effective embedding space structure. We further elaborate on this advantage in § 5.4.

3 PROBLEM STATEMENT

We first introduce necessary definitions and notations.

Definition 1. *(Time Series) Multivariate time series is a sequence of multivariate observations:*

$$\boldsymbol{x} = \{x_i\}_{i=1}^{T}, x_i \in \mathbb{R}^d \tag{1}$$

where $x_i \in \mathbb{R}^d$ is the i-th multivariate observation, T is the length of the time series, d is the number of channels.

We will use $\mathbf{x}_{a:b}$ to denote a subsequence of $\mathbf{x}$, which starts from time a and ends at time b. If necessary, an extra subscript j can be added to indicate the channel, e.g., $x_{i,j}$ denotes the j-th variable in multivariate observation x_i.

To make our paper broadly applicable, we extend the traditional meaning of states by considering not only the low-level states of individual channels in the multivariate time series but also the high-level states composed of repetitive low-level states. For example, the low-level states may capture the movement of the left foot and the movement of the right foot separately. We can then define the high-level state "running" as alternating the feet quickly and the high-level state "walking" as alternating the feet slowly. In other words, we can define states as the state combination of different channels in multivariate time series to elaborate on an object's interesting physical behaviour. To this end, state is an abstract and encompassing concept related to context interests, which cannot be defined with explicit rules in advance and identified with hand-crafted features. So we only give an abstract definition by characterizing its temporal behavior.

Definition 2. *(State) A state is a subsequence with potential semantic value in real-world, which generally shows high internal similarity, such as repetitive patterns and similar statistical characteristics.*

Note that within a state duration of a multivariate time series, there is only one state in each channel since otherwise, we can split this state of the multivariate time series into multiple states. Fig. 1 shows an example of states in a multivariate time series. By indexing the states of a multivariate time series, we define the concept of state sequence:

Definition 3. *(State Sequence) A state sequence of a multivariate time series is a sequence of index numbers that indicate which state a time point x_i belongs to. A state sequence can be denoted as:*

$$\boldsymbol{s} = \{s_i \mid i \in \mathbb{N}^+, i \leq N, s_i \in \mathcal{S}\} \tag{2}$$

where $\mathcal{S}$ is the set of indexes $\{0, 1, \ldots\}$. Note that a state sequence and its corresponding raw time series have the same length.

It has been observed that the physical or logical process of the monitored target usually contains multiple discrete states, and the time series generated at different states are different w.r.t shape or statistical characteristics [14, 16, 23, 25, 38]. Even if some systems are not naturally in discrete states, it is often helpful to discretize their states for ease analysis [14]. We call a time series segmentable if it satisfies the above observation.

The problem that we study in this paper is: Given a segmentable multivariate time series $\mathbf{x}$, how to determine the number of states and which state each time point belongs to, i.e., finding the corresponding state sequence $\mathbf{s}$ of the given time series $\mathbf{x}$.

4 DESIGN

4.1 Core Ideas

Time2State contains two phases, the training phase and the detection phase, respectively. In the training phase, we train an encoder to learn distinguishable representations from the raw time series. This encoder will be used later in the detection phase shown in Fig. 3. To train an effective encoder, we utilize two observations: 1) As illustrated in Fig. 2, if a time window is randomly placed on a time series, the windows consecutive/close to it (windows of the same color) should contain samples of the same state with high probability. Note that these windows might span multiple states, but even so, their composition should be similar. In other words, *if each window were given a state*, the N consecutive windows form a sequence of states of length N, where these N states are likely to be the same. We call these N states intra-state samples. The representations of intra-state samples should be close in the embedding space. 2) Multiple non-consecutive windows (denoted as windows of different colors in Fig. 2) are more likely to contain data samples belonging to different states. *If each window were given a state*, these states are likely to be different. We call these states inter-state samples. Based on the above two observations, we design a novel loss function, Latent State Encoding Loss (**LSE-loss**), which encourages the encoder to map the intra-state samples close to each other and the inter-state samples farther away from each other in the embedding space. More details are disclosed in § 4.2.

4.2 Latent State Encoding Loss (LSE-Loss)

4.2.1 The Design. Recall that our goal is to approximate an encoder that can learn distinguishable representations, so as to map a time series into a low-dimensional embedded sequence (in a sliding window manner) and identify the cluster structure in these representations from the low-dimensional embedding space. The encoder can be formulated as $f_\theta : \mathbb{R}^{w*d} \mapsto \mathbb{R}^z$, where w, d, z is the window size, channel number, and dimension of embedding space, respectively. It is desired that windowed data from the same state are closer and windowed data from different states are far away from each others in the embedding space, because this is more favorable for clustering.

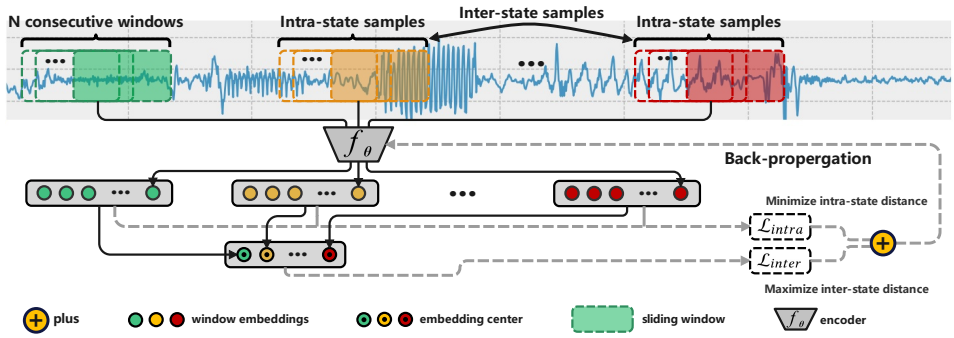


Fig. 2. Training procedure of Time2State. In each round of training, Time2State randomly samples N consecutive windows for M times. Then it optimizes the encoder by jointly minimizing the total distance of intra-state samples and maximizing the total distance of inter-state samples.

To achieve this goal, we propose a novel loss function called Latent State Encoding Loss **(LSE-Loss)**. As illustrated in Fig. 2, in each round of training, there is a two-stage sampling process in LSE-Loss. To obtain inter-state samples, the first sampling stage (inter-state sampling hereinafter) randomly places M non-consecutive windows on the given time series $\mathbf{x}$. The positions of these windows, denoted as $\{t_k\}_{k=1}^{M}$, are randomly drawn from a uniform distribution, i.e., $t_k \sim U(0, T - w - N)$, because this enables LSE-Loss to evenly learn the features of the entire time series. Each window is assigned a state (i.e., a sample), and thus corresponding to all the windows, we obtain M inter-state samples, denoted as $\mathbf{x}_{t_k:t_k+w}$, $t_k \sim U(0, T - w - N)$. To obtain intra-state samples, the second sampling stage (intra-state sampling hereinafter) slides every window placed by the first sampling stage forward for $N - 1$ times in a step size 1. Assuming a window falls in position t, its subsequent $N - 1$ windows and itself can be denoted as $\{\mathbf{x}_{t+i:t+w+i}\}_{i=1}^{N}$. Assigning each window a state (i.e., a sample), we obtain all the intra-state samples. Through the two-stage sampling process, we get M groups of intra-state samples.

The LSE-Loss contains two parts:

$$\mathcal{L}_{LSE} = \mathcal{L}_{intra} + \mathcal{L}_{inter}. \tag{3}$$

The intra-state part $\mathcal{L}_{intra}$ encourages the encoder to maximize the similarity between the representations of intra-state samples, so as to push them close in the embedding space:

$$\mathcal{L}_{intra} = \alpha_1 \sum_{k=1}^{M} \sum_{i=1}^{N} \sum_{j=1, j<i}^{N} -\log\left(\sigma\left(f_\theta(\mathbf{o}_i^k)^\intercal f_\theta(\mathbf{o}_j^k)\right)\right) \tag{4}$$

where $\alpha_1 = \frac{2}{M*N*(N-1)}$ is used to average the similarity, σ is the sigmoid function, f_θ is the encoder parameterized by θ. $\mathbf{o}_i^k = \mathbf{x}_{t_k+i:t_k+i+w}$ denotes the i-th window in the k-th group, where $t_k \sim U(0, T - w - N), i \leq N, k \leq M$. $f_\theta(x_i)^\intercal f_\theta(x_j)$ is the similarity, which uses the widely-adopted dot product to measure the similarity [13, 17, 24, 35].

The inter-state part $\mathcal{L}_{inter}$ encourages the encoder to minimize the similarity between the representations of inter-state samples, so as to separate them farther away in the embedding space:

$$\mathcal{L}_{inter} = \alpha_2 \sum_{k=1}^{M} \sum_{l=1, l<k}^{M} -\log\left(\sigma\left(-\mathbf{c}_k^\intercal \mathbf{c}_l\right)\right) \tag{5}$$

where $\alpha_2 = \frac{2}{M*(M-1)}$ is used to average the similarity. Note that we put the $-$ sign before $\mathbf{c}_k$ so that minimizing $\mathcal{L}_{inter}$ is equivalent to minimizing the embedding similarity between all inter-state samples. Also note that since minimizing the embedding similarity between all inter-state samples is computational expensive, we instead use the similarity between the embedding centers of different groups. Embedding center is calculated by $\mathbf{c}_k = \frac{1}{N}\sum_{i=1}^{N}\left(f_\theta(\mathbf{o}_i^k)\right)$.

*4.2.2 Rationale of **LSE-Loss**.* The samples generated with our sampling method can be divided into four groups: true/false intra-state samples and true/false inter-state samples. As discussed in § 4.1, even though two subsequent windows both span multiple states, their state composition are still similar with the same principle states. In this way, these windows contain mostly true intra-state samples, with very few exceptions on false intra-state samples. Inter-state sampling, however, may generate samples that are in the same state (i.e., false inter-state) or in different states (i.e., true inter-state). Using $\mathcal{L}_{inter}$ to push true inter-state samples away will not increase $\mathcal{L}_{intra}$, but using it to separate false inter-state samples would increase $\mathcal{L}_{intra}$. Hence, minimizing $\mathcal{L}_{intra}$ becomes the counter-force against the latter case and enforces $\mathcal{L}_{inter}$ to focus more on those true inter-state samples.

4.3 The Encoder

We highlight that the implementation of the encoder is a design choice, and LSE-Loss is encoder-extensible, meaning that we can adopt other encoder architectures in Time2State. In this paper, we implement the encoder as a causal convolution network [13] because it has been proved to be efficient for time series data, and can alleviate the disadvantages of recurrent neural networks (e.g., LSTM, GRU) [13].

4.4 Detection Phase of Time2State

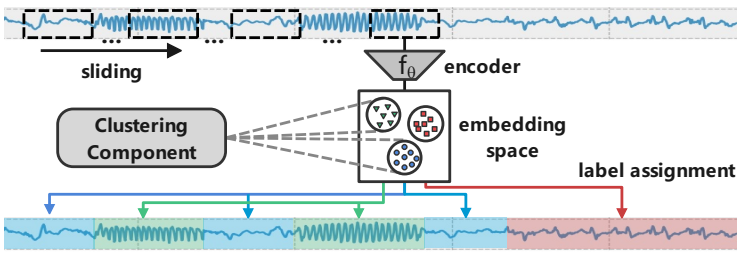


Fig. 3. Detection phase of Time2State.

Once the encoder is trained, we use a sliding window in the detection phase to slide across the entire time series at a step size s, and calculate the representations for the data in each window, as illustrated in Fig. 3. In this way, a long time series with multiple channels can be reduced into a shorter low-dimensional embedded sequence. Finally, we identify the cluster structure in these representations from the low-dimensional embedding space and assign the clustering labels to the raw time point. Note that as the sliding window moves, at a step size s, across the time series, each time point (except the first and last few points) is covered by the window multiple times. In other words, each time point will receive multiple labels. The final label assigned to each time point is based on majority voting of the labels, which also helps eliminate a small number of potentially incorrect point labels. In addition, we do not use any label smoothing algorithm to maintain a clear state transition, because some data may not have clear semantics. For example, if the whole time series consists of random noise, Time2State would not be able to find clear state switches.

4.5 The Clustering Component

After transforming a time series into an embedding sequence, we cluster the embeddings and assign labels for each cluster. Accordingly, the raw time points are also labelled. Unlike usual scenarios, the state number K is unknown in advance; thus the selection of clustering methods is quite limited. Existing methods like K-means [30], spectral clustering [36], and other clustering methods require the number of clusters as input. Although density-based clustering algorithms like DBSCAN [11], Density-peak clustering [27] do not require to specify the number of clusters, they need to specify EPS and threshold to divide clusters, which are also difficult to obtain as a prior knowledge. Even their variants, such as OPTICS [6], still require human intervention to select the threshold. Moreover, when the dataset is heterogeneous, it is costly to set parameters for every time series. To avoid all the above difficulties, we use a nonparametric Bayesian method to cluster low-dimensional embeddings. In detail, we apply the Dirichlet Process Gaussian Mixture Model (DPGMM) [8] to cluster the embeddings, which can automatically estimate the number of clusters.

DPGMM is an infinite mixture model with the Dirichlet Process (DP) as a prior distribution on the number of clusters. It uses the DP model to adapt the number of states automatically. It assumes that the observed data are generated from a Gaussian mixture model, and the parameters of the distribution come from a discrete random measure G (G is drawn from a DP), i.e.,

$$e_i|\eta_i \sim F(e_i|\eta_i),\ \eta_i|G \sim G,\ G \sim DP(\alpha, H) \tag{6}$$

There is no direct way to compute the posterior distribution under a DP mixture prior. Approximate inference methods are required for DP mixtures, and variational inference provides a deterministic methodology for approximating likelihoods and posteriors. The DPGMM can be approximated by maximizing the Evidence Lower Bound. We follow the implementation described in [8].

4.6 Selecting Parameters for Time2State

The window size w and the step size s are two important factors in Time2State. Overall, w should be selected properly such that it is long enough to contain information about the underlying state but not too long to span over multiple states. s should also be carefully selected such that it is large enough to speed up the detection process but not too large to miss important information. As a general guideline, w can be set according to the state level of interest, which relies on particular applications. For instance, in power systems, states are usually at the level of a day or a week; in ECG signals analysis, states are usually at the level of seconds [33]. Regarding the step size s, our later empirical results suggest that it should be set in proportion to w, generally 20% $\sim$ 40%, to maintain a good balance between detection speed and detection accuracy.

Moreover, M, N may also affect the performance of Time2State, although later revealed to be slightly. We will investigate the effect of these parameters in § 5.5.

4.7 Discussion and Analysis

4.7.1 Working Mode of Time2State. Time2State works offline as a statistical and unsupervised model for analyzing a given time series. In this context, there is no "future data" for online inference – all time points are provided to the model. Similar to the methods mentioned in § 2, the acquired model is directly used on the given time series, and there is no need to distinguish training data, validation data, and test data. In this context, there is no information leakage.

4.7.2 Regularization and Convergence. We do not use explicit regularization terms to mitigate overfitting. Instead, dropout is added to the network used by Time2State to alleviate overfitting, which is considered a regularization method for neural network models [29]. Regarding convergence,

similar to statistical methods like HDP-HSMM [32], Time2State contains a maximum iteration limit to balance computation time and performance.

4.7.3 Computational Complexity Analysis. The computational cost in each training epoch is $O\big(M \cdot N \cdot c_{f_\theta}(w)\big)$, where $c_{f_\theta}(w)$ is the cost of encoding and back-propagating through f_θ on a time window of size w. To speed up the detection process, we can adopt a step size $s(>1)$ and move the sliding window at the step size s. The computational cost in the detection phase is $O\big(\lceil (n-w)/s \rceil \cdot \hat{c}_{f_\theta}(w)\big)$, whose upper bound is $O\big(n \cdot \hat{c}_{f_\theta}(w)\big)$, where w, s are the window size and step size, respectively, $\hat{c}_{f_\theta}(w)$ is the cost of encoding through f_θ (without propagation) on a time window of size w.

5 EVALUATION

We design experiments to answer three questions:

(Q1) Effectiveness of Time2State: Is Time2State effective in inferring the latent state sequence of time series?

(Q2) Scalability of Time2State: How does Time2State scale in terms of computational time?

(Q3) Effectiveness of LSE-Loss: Is LSE-Loss effective in learning distinguishable representations?

5.1 Setup

5.1.1 Datasets. We use five real-world datasets and a synthetic dataset to validate the effectiveness of Time2State. The statistics of these datasets are summarized in Table 1. We choose the datasets to cover diverse application scenarios such as human activity, medical context, and insect monitoring. Note that some datasets used in time series segmentation, e.g., those in [28], are not suitable for evaluating state detection as they lack necessary out-of-band information to label the states.

MoCap [2]: This dataset is from the CMU motion capture dataset. Following the setting in [23], we choose four dimensions corresponding to left/right arms and legs.

USC-HAD [5]: This dataset includes 12 human activities (e.g., jumping, running) that were recorded separately for 14 people. Each human wore a 3-axis accelerometer and a 3-axis gyrometer attached to the front of the right hip and sampled at 100Hz.

PAMAP2 [3]: This dataset includes 24 low-level (e.g., walking and sitting) and high-level (e.g., playing soccer, which consists of two or more low-level) human activities undertaken by 8 people. We choose 9 dimensions corresponding to x, y, and z-axis acceleration data of hand, chest, and ankle, respectively.

ActRecTut [1]: This dataset contains acceleration data collected from sports activities. In the experiment, we use a subset of the data related to people's walking. Note that the data were for mining frequent patterns rather than latent state inference, and thus the dataset does not include enough labelled data for other activities.

UCR-SEG [14]: A collection of benchmark datasets, each corresponds to a univariate time series taken from the medical, insects, robots, electrical power demand, etc.

Synthetic: We use a tool called TSAGen [37] to generate synthetic time series, each containing 5 states. The shape characteristics of segments reflect the states, each lasting for a random period.

5.1.2 Baselines. We compare Time2State with multiple time series segmentation methods, including:

AutoPlait [23]: A fully automatic time series segmentation algorithm, which requires no parameter tuning and user intervention.

HVGH [25]: A hybrid method that combines VAE, HDP, and GP, and automatically estimates the number of states through HDP.

Table 1. Statistical information of the datasets.

Datasets	# states	# channels	length (k)	# time series	# segments	state duration (k)
Synthetic	5	4	9.3∼23.7	100	18∼28	0.1∼3.9
MoCap	5∼8	4	4.6∼10.6	9	6∼11	0.4∼2.0
ActRecTut	6	10	31.4∼32.6	2	42	0.02∼5.1
PAMAP2	11	9	253∼408	10	18∼25	2.0∼40.3
USC-HAD	12	6	25.4∼56.3	70	12	0.6∼13.5
UCR-SEG	2∼3	1	2∼40	32	2∼3	1∼25

states, # channels, # segments, state duration and length are for each time series.

TICC [16]: A Toeplitz inverse covariance-based time series segmentation method. This is the only baseline used in our experiments that needs to specify the number of states.

HDP-HSMM [32]: A Bayesian non-parametric method that extends the traditional Hidden Semi-Markov Model. The number of states is estimated by HDP.

ClaSP+TSKMeans: We construct this baseline by combining the cut point detection method ClaSP [28] with a time series clustering method (Time Series KMeans [4]) for state detection. Note that ClaSP is the newest time series segmentation method, but for univariate time series only. We put a great deal of effort to make ClaSP work for multivariate time series segmentation by following the method suggested (but not implemented) in [14], i.e., we average the Classification Score Profile (CSP) curves (intermediate products extracted by ClaSP) of every channel to obtain the CSP curve of the whole multivariate time series and then apply ClaSP on this overall CSP for segmentation. For simplicity, we use ClaSP to denote this ClaSP+TSKMeans baseline in the rest of the paper.

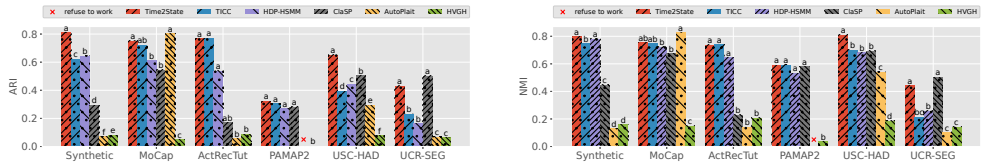
5.1.3 Metrics. Following the convention in state detection evaluation [7, 16], we test the detection performance using the clustering measures. The predicted state sequence is compared to the ground truth under two clustering metrics, Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI). These metrics intuitively show the matching degree and state assignment performance.

5.1.4 Implementation details. We conducted all experiments on a machine equipped with an Intel(R) i7-7800X CPU @ 3.50GHz and 64 GB RAM, running Ubuntu 20.04 LTS with access to an NVIDIA 1080Ti GPU.

5.1.5 Hyperparameter Optimization. For fair comparison, we optimize the parameters for all baselines except AutoPlait, as AutoPlait requires no user intervention and no parameter tuning [23]. Note that parameter tuning upon datasets is feasible, but parameter tuning for every time series is prohibitive. For all methods, we refer to the strategies and recommendations given in their original papers to select their hyperparameters. If there is no recommendation, we perform grid search in an appropriate hyperparameter space (according to the settings in their evaluation code). For example, for TICC, the λ (regularization parameter that determines the sparsity level in the MRFs characterizing each cluster), β (smoothness penalty that encourages adjacent subsequences to be assigned to the same cluster), and δ (threshold) parameters are optimized in $\{i * 500 | i \leq 6, i \in \mathbb{N}^+\}$, $\{1e-2, 1e-3, 1e-4\}$, and $\{1e-2, 1e-3, 1e-4\}$, respectively. The parameter optimization for Time2State is carried out in a much more sparser space. We only jointly optimize the window size and step size in {128, 256, 512} and {50, 100}, respectively. Regarding M and N, we do not deliberately optimize them, but simply set them to 10 and 4. We can see from the following evaluation results that Time2State outperforms the counterparts even with parameters settings optimized within a small set of values.

5.2 Effectiveness of Time2State

Quantitative results are shown in Fig. 4, wherein significance markers are gained from *t-test*. Totally different markers indicate there is a significant difference between two methods on the dataset with confidence level 95%. For example, "a", "ab", "bc" means that there is no significant difference between "a" and "ab", but "a" and "bc" are significantly different. Fig. 5 is the Critical Difference (CD) diagram on all time series data, where methods that are not connected by a bold line are significantly different in their average ranks. This validates that Time2State significantly outperforms other methods in average rank. Strictly speaking, only AutoPlait, HDP-HSMM and HVGH can be regarded as the counterparts of Time2State, because they do not require the number of states in advance, while TICC and ClaSP+TSKmeans require the number of states in advance, which undoubtedly contributes to the performance. Thus we compare Time2State with its counterparts (AutoPlait, HDP-HSMM and HVGH) and the others (TICC and ClaSP), separately.



(a) ARI (b) NMI

Fig. 4. Performances of Time2State and baselines.

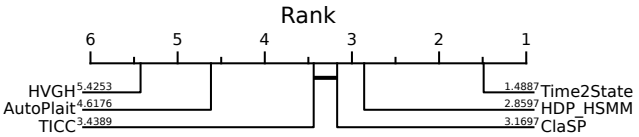


Fig. 5. Critical Difference Diagram on all time series data for Time2State and baselines with a confidence level of 95%. A smaller rank means better performance.

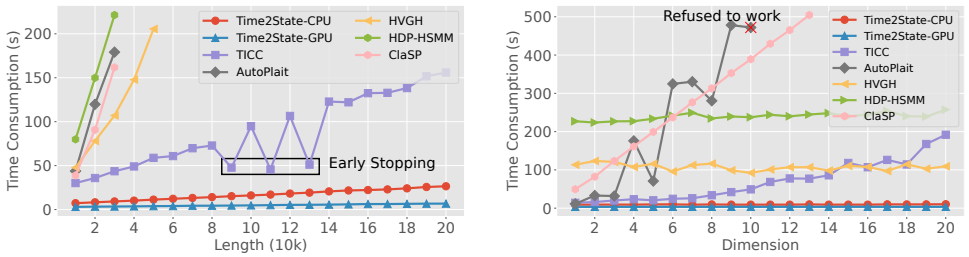
5.2.1 Time2State versus Counterparts. As shown in Fig. 4, Time2State significantly exceeds its counterparts on all datasets, but does not perform as well as AutoPlait on the MoCap dataset. However, AutoPlait performs poorly on the other datasets. Inspecting the original paper of AutoPlait, we found that although AutoPlait is claimed to be a general method, its evaluation is mainly carried out on the MoCap dataset, suggesting that it might overfit this dataset. Another noteworthy phenomenon is that HVGH almost fails to return meaningful results on all datasets. Since its code is available but the evaluation code is not publicly available, we tried to tune the hyperparameters and report the best results of HVGH. As a counterpart, HDP-HSMM also achieves good results. But such a performance is obtained at the cost of a sharp increase in computing time. As we will see in § 5.3, its computation time increases sharply with the increment of time series length, which greatly limits its application in large-scale data. The above result shows that Time2State has a significant advantage when dealing with long time series, which demonstrates its generality. Unlike AutoPlait and HVGH, Time2State shows stable performance on all datasets.

5.2.2 Time2State versus TICC. As illustrated in Fig. 4, Time2State outperforms TICC on all datasets except for ActRecTut. TICC slightly outperforms Time2State on ActRecTut. However, considering that TICC requires to specify the number of states, the performance of Time2State is very competitive. This further illustrates the effectiveness of Time2State. It fills the gap between methods that require the number of states as input and methods that do not.

5.2.3 Time2State versus ClaSP. ClaSP achieves good performance on datasets where the state duration is long and the states do not switch frequently (e.g., UCR-SEG, PAMAP2), but its performance on other datasets where the state duration is relatively short and the states switch frequently (e.g., ActRecTut, Synthetic) is relatively poor. A possible reason is that the latter datasets contain more segments and the phase difference is larger, in which cases clustering suffers from obscure margins. This result indicates that ClaSP is good at finding change points but is not suitable for state detection, as also revealed in § 2.

Another important problem is that change point detection methods like ClaSP and FLOSS must specify the number of segments, which is difficult in the context of state detection. Note that the number of segments is more difficult to obtain than the number of states as prior knowledge, because the number of segments may also be related to the length of monitoring duration, which is usually not fixed. In summary, these methods do not consider state detection as a downstream task and are not suitable for state detection.

5.3 Scalability of Time2State



(a) Effect of length

(b) Effect of length

Fig. 6. Computational time of Time2State and baselines.

We also compare Time2State and baselines in terms of scalability via variable-control experiments. We generate time series of different lengths and dimensions (i.e., number of channels) to test their scalability. To validate the effect of length, we fix the dimension to 4 and vary the length from 10k to 200k; To validate the effect of dimension, we fix the length to 30k and vary the dimension from 1 to 20. Because the training and encoding process of the encoder in Time2State can be accelerated by GPU, we conducted the experiment twice for Time2State with and without GPU, respectively. Note that the VAE in HVGH always uses GPU. In this experiment, we adopted the same setting for the effectiveness experiment. For ClaSP, we do not calculate the time used for clustering to show its fastest possible performance. For Time2State, we used the slowest setting in the effectiveness experiment (window size = 512, step size = 50) to reflect the worst-case running speed of Time2State. Also, note that the time consumption of Time2State contains the training time, i.e., the dataset is heterogeneous, and in this case, Time2State needs to be trained on each time series, respectively.

Fig. 6 compares the computation time of Time2State and baselines over time series of varying lengths and dimensions. Note that the computation time of AutoPlait, HVGH and HDP-HSMM

increases too fast with the increase of the length of the time series. Thus we only plot their first few points.

5.3.1 Effect of Length. As shown in Fig 6(a), with the increment of time series length, the computation time of Time2State increases linearly and increases much slower than baselines. The computation time of baselines also shows a roughly linear growth rate (except HVGH), but the growth rate is much larger. The curve of TICC slightly fluctuates because TICC has an early stop mechanism and it may converge early in some cases. But even in these cases, it is still much slower than Time2State. With the increment of length, the advantage of Time2State becomes more pronounced. When the length reaches 200k (which is quite common in practice), the computation time of Time2State-CPU and Time2State-GPU is about 26.48 s and 6.75 s, respectively, which are about 5.9 times and 23.1 times faster than the fastest baseline (TICC, 155.94s), respectively.

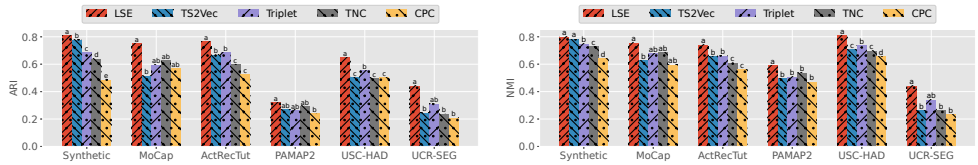
5.3.2 Effect of Dimension. As shown in Fig 6(b), with the increment of dimension, the computation time of Time2State, HDP-HSMM and HVGH does not change significantly. This is because these methods do not consider the correlation between different channels. For example, HDP-HSMM considers each multivariate observation as a high-dimension sample; Time2State can directly encode all channels within the sliding window simultaneously, because the convolution kernels of the network work on all channels in parallel. Although the data loading time increases with the increase of the number of channels, this extra time is negligible compared with the computation time. In other words, the computation time of Time2State, HDP-HSMM, and HVGH is not sensitive to the number of channels. In contrast, the computation time of TICC, AutoPlait and ClaSP increases fast. Note that the computation time of AutoPlait is more or less related to the number of states. Changing the number of channels will affect its estimation of the number of states. Thus its curves are not very smooth. Moreover, the curve of AutoPlait stops at 10 dimensions because it refuses to work and simply outputs one state when dimension$\geq$10. The computation time of ClaSP increases fast because it must process each time series channel by channel, which is needed to adapt ClaSP for multivariate time series data. When the dimension reaches 20 (which is quite common in practice), the computation time of Time2State-CPU and Time2State-GPU is about 10.25s and 3.5s, respectively, indicating about 18.7 and 54.8 times faster than TICC (191.82s), respectively.

The above experimental results clearly demonstrate the significantly higher scalability of Time2State than the baselines. With the increment of length and dimension of time series, the advantage of Time2State becomes more and more pronounced, and this advantage can be further boosted with GPU. As far as we know, TICC, HDP-HSMM and AutoPlait cannot be accelerated by GPU or require non-trivial efforts to make them run on GPU.

5.4 Effectiveness of LSE-Loss

To evaluate the effectiveness of the proposed loss function, we compare it to state-of-the-art baselines, which are Triplet-Loss [13], Temporal Neighborhood Coding (TNC) [33], Contrastive Predictive Coding (CPC) [21, 34] and TS2Vec [40]. As stated in the original paper [13, 33], all of these baselines are independent of the architecture of the encoder. For a detailed introduction of these loss functions, refer to the original paper [13, 21, 33, 34, 40].

5.4.1 Performance of Different Loss Functions. For a fair comparison and to ensure that the difference in performance is not caused by the differences in the encoders' architecture, we followed the convention in [13, 33] and used the same encoder network across all compared baselines. We highlight that we perform no hyperparameter optimization of the encoder architecture for all datasets. For the choice of the encoder, we directly adopt the encoder network used in [13]. In addition, TNC requires a discriminator in the training stage, so we use the discriminator implemented in the



(a) ARI (b) NMI

Fig. 7. The effectiveness of LSE-Loss and baselines.

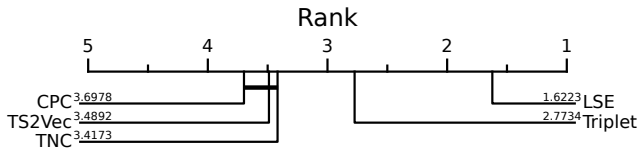


Fig. 8. Critical Difference Diagram on all time series data for LSE-Loss and baselines with a confidence level of 95%. A smaller rank means better performance.

original paper of TNC [33]. Except for the loss function, all other hyperparameters are fixed across the experiments.

Using the above experiment setup, we run Time2State on all datasets used in § 5.2. Fig. 7 shows that LSE-Loss outperforms baselines by a large margin. Fig. 8 is the Critical Difference (CD) diagram on all time series data, where methods that are not connected by a bold line are significantly different in average ranks. This validates that LSE-Loss significantly outperforms baselines in average ranks. This result indicates that LSE-Loss is more effective in learning representations from time series data.

5.4.2 Embedding Space Visualization. To intuitively show the effectiveness of LSE-Loss and further explore the causes of performance differences in these loss functions, we plot the representations in embedding space. We run the encoder with different loss functions with the same settings as above. During this experiment, we first train four encoders. These encoders use the same architecture and hyperparameters but are trained with different loss functions. Then we use a sliding window of size 256 and step size 50 to slide across a time series picked from the synthetic dataset. At the same time, these windows are encoded by the encoders. The raw time series are partially plotted in Fig. 9.

Fig. 9 shows the embedding space and representations learned by LSE-Loss and baselines. Points of the same color denote the representations of windows from the same latent state. The gray points are the representations of windows that span two states, and the trajectories of gray points imply the transition of states. It is obvious that the representations learned by LSE-Loss have a clearer inter-state boundary and greater inter-state distance (relatively). However, the representations learned by the baselines have a relatively poor structure. The distance between the representations of windows from different states is very close, and the transition trajectory is unclear, which is not conducive to clustering. The above result clearly illustrates the effectiveness of LSE-Loss.

5.5 Parameter Study

5.5.1 Effect of the Range Parameters. We conduct variable-control experiments on all datasets to explore the effect of parameters s and w. As is discussed in [33], window size w should be self-defined according to the domain interests on state granularity (e.g., small window for fine-grained

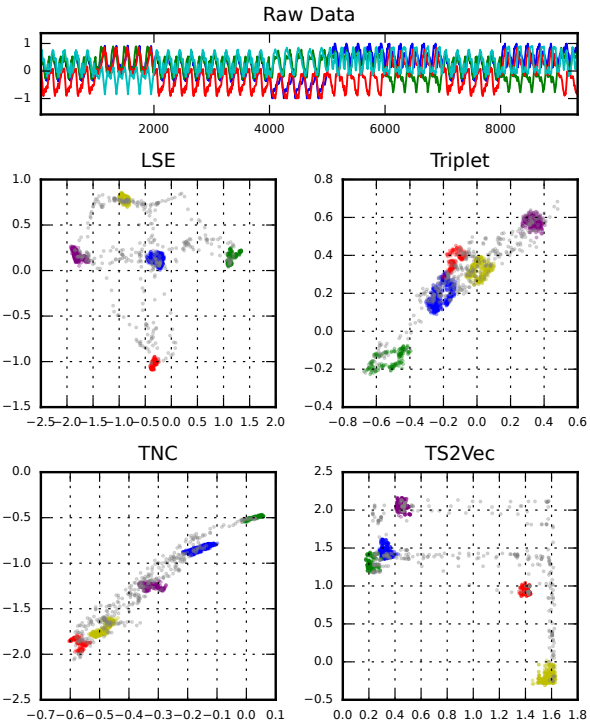
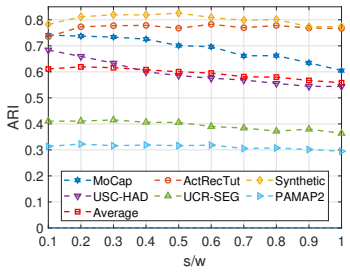
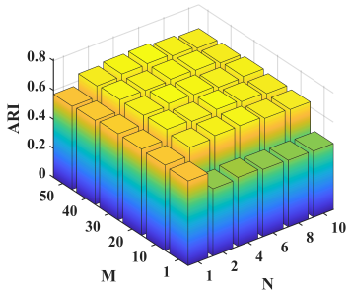


Fig. 9. Embedding space learned by LSE-Loss and baselines.



(a) Effect of the range parameters



(b) Effect of M and N

Fig. 10. The effect of different parameters.

activity detection). Meanwhile, the impact of these two range parameters is highly-correlated in generating the sampling distribution, which makes independent adjustment to have redundant calculation or ineffective detection. For example, $s = 100$ is large compared to $w = 100$, while small compared to $w = 1000$. When a very small s/w is applied, the performance may not be improved but a lot of redundant calculations are paid; otherwise, with large s/w, although the computation cost is reduced, the performance would decline. Therefore, w.l.o.g., we studied the joint effect of s/w in range $\{0.1, 0.2, \ldots 1.0\}$ in this part.

From Fig. 10(a), we can see that with the increment of s/w, the overall performance of Time2State first increases (the inapparent increasing stage on USC-HAD is below 0.05), then remains stable, and finally decreases slowly. A possible explanation for the above phenomenon is: Small s/w may cause Time2State to encode redundant information between two windows, while large s/w may cause Time2State to miss information between two windows. Especially when $s{>}w/2$, the sliding window may not cover all data points evenly. These factors could all impact the performance of our embedding clustering method DPGMM. The above results indicate s should be selected in proportion to w. Referring to the average performance in Fig. 10(a), empirically, setting s/w in $20\% \sim 40\%$ helps maintain a good balance between detection speed and accuracy.

5.5.2 Effect of M and N. We conduct variable-control experiments on all datasets to explore the effect of M and N. During the experiments, we fix M and change N in $\{1, 10, 20, 30, 40, 50\}$, and fix N and change M in $\{1, 2, 4, 6, 8, 10\}$, respectively. Note that $M = 1, N \neq 1$ is a special case where only the $\mathcal{L}_{intra}$ is used, and $M \neq 1, N = 1$ is a special case where only the $\mathcal{L}_{inter}$ is used. In addition,

when $M = 1, N = 1$, the encoder cannot be trained (i.e., the encoder will be the one with random parameters). This is also an ablation study that explores the effect of $\mathcal{L}_{intra}$ and $\mathcal{L}_{inter}$.

Fig. 10(b) shows the result on USC-HAD. From Fig. 10(b), we can see that when $M \neq 1, N \neq 1$, the performance of Time2State and LSE-Loss is quite stable, indicating that Time2State is not sensitive to M and N. An interesting phenomenon is that, even with very few intra-state samples (i.e., very small N, e.g., N=2), Time2State can still work stably. When $M = 1$ or $N = 1$, the performance of Time2State drops sharply. The same phenomenon can be observed on the results of all the other datasets, which are provided on the project page. The above phenomenon suggests that: 1) Time2State is not sensitive to the selection of M and N as long as the integrity of the structure of LSE-Loss is maintained; 2) Only a small amount of intra/inter-state samples can make Time2State work stably, which greatly benefits the scalability; 3) $\mathcal{L}_{intra}$ and $\mathcal{L}_{inter}$ both play a crucial role in LSE-loss, which is also in line with our intuition provided in § 4.2.2.

6 CONCLUSION

Time series data from many applications record the historical activities of the object under study. Identifying and inferring the latent states of the physical objects can benefit many downstream tasks like event detection. However, existing methods either need to specify the number of states, generally unknown in advance, or incur intensive computation. We designed a fully unsupervised, scaleable framework, Time2State, for inferring the latent state of time series data. Extensive experiments with real-world and synthetic datasets showed that Time2State pushes the state-of-the-art with a large margin w.r.t. effectiveness and scalability.

REFERENCES

[1] ActRecTut. https://github.com/andreas-bulling/ActRecTut.

[2] MoCap. http://mocap.cs.cmu.edu.

[3] PAMAP2. http://www.pamap.org/demo.html.

[4] Time Series KMeans. https://tslearn.readthedocs.io/en/stable/gen_modules/clustering/tslearn.clustering.TimeSeriesKMeans.html.

[5] USC-HAD. http://sipi.usc.edu/had/.

[6] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record* 28, 2 (1999), 49–60.

[7] Abdul Fatir Ansari, Konstantinos Benidis, Richard Kurle, Ali Caner Turkmen, Harold Soh, Alexander J Smola, Bernie Wang, and Tim Januschowski. 2021. Deep Explicit Duration Switching Models for Time Series. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., New York, NY, 29949–29961.

[8] David M Blei and Michael I Jordan. 2006. Variational inference for Dirichlet process mixtures. *Bayesian analysis* 1, 1 (2006), 121–143.

[9] Shohreh Deldari, Daniel V. Smith, Amin Sadri, and Flora Salim. 2020. ESPRESSO: Entropy and ShaPe AwaRe TimE-Series SegmentatiOn for Processing Heterogeneous Sensor Data. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 3, Article 77 (2020), 24 pages. https://doi.org/10.1145/3411832

[10] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. 2021. Time-Series Representation Learning via Temporal and Contextual Contrasting. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, 2352–2359.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *kdd*, Vol. 96. AAAI press, 226–231.

[12] Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. 2018. Som-vae: Interpretable discrete representation learning on time series. *arXiv preprint arXiv:1806.02199* (2018).

[13] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. 2019. Unsupervised Scalable Representation Learning for Multivariate Time Series. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., New York, NY.

[14] Shaghayegh Gharghabi, Chin-Chia Michael Yeh, Yifei Ding, Wei Ding, Paul Hibbing, Samuel LaMunion, Andrew Kaplan, Scott E Crouter, and Eamonn Keogh. 2019. Domain agnostic online semantic segmentation for multi-dimensional time series. *Data mining and knowledge discovery* 33, 1 (2019), 96–130.

[15] Peter D Grünwald. 2007. *The minimum description length principle.* MIT press.
[16] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. 2017. Toeplitz Inverse Covariance-Based Clustering of Multivariate Time Series Data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 215–223. https://doi.org/10.1145/3097983.3098060
[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. IEEE, Piscataway, NJ, 9729–9738.
[18] Kouki Kawabata, Yasuko Matsubara, and Yasushi Sakurai. 2018. StreamScope: Automatic Pattern Discovery over Data Streams. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management* (Houston, TX, USA) *(aiDM'18)*. Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages. https://doi.org/10.1145/3211954.3211959
[19] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).
[20] Wei-Han Lee, Jorge Ortiz, Bongjun Ko, and Ruby Lee. 2018. Time Series Segmentation through Automatic Feature Learning. arXiv:1801.05394 [cs.LG]
[21] Sindy Löwe, Peter O' Connor, and Bastiaan Veeling. 2019. Putting An End to End-to-End: Gradient-Isolated Learning of Representations. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., New York, NY.
[22] Qianli Ma, Jiawei Zheng, Sen Li, and Gary W Cottrell. 2019. Learning Representations for Time Series Clustering. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., New York, NY.
[23] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2014. AutoPlait: Automatic Mining of Co-Evolving Time Sequences. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 193–204. https://doi.org/10.1145/2588555.2588556
[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, Vol. 26. Curran Associates, Inc., New York, NY.
[25] Masatoshi Nagano, Tomoaki Nakamura, Takayuki Nagai, Daichi Mochihashi, Ichiro Kobayashi, and Wataru Takano. 2019. HVGH: Unsupervised Segmentation for High-Dimensional Time Series Using Deep Neural Compression and Statistical Generative Model. *Frontiers in Robotics and AI* 6 (2019).
[26] Mathias Perslev, Michael Jensen, Sune Darkner, Poul Jø rgen Jennum, and Christian Igel. 2019. U-Time: A Fully Convolutional Network for Time Series Segmentation Applied to Sleep Staging. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., New York, NY.
[27] Alex Rodriguez and Alessandro Laio. 2014. Clustering by fast search and find of density peaks. *science* 344, 6191 (2014), 1492–1496.
[28] Patrick Schäfer, Arik Ermshaus, and Ulf Leser. 2021. ClaSP - Time Series Segmentation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 1578–1587.
[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[30] Douglas Steinley. 2006. K-means clustering: a half-century synthesis. *Brit. J. Math. Statist. Psych.* 59, 1 (2006), 1–34.
[31] Yifan Sun, Yaqi Duan, Hao Gong, and Mengdi Wang. 2019. Learning low-dimensional state embeddings and metastable clusters from time series data. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., New York, NY.
[32] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. 2006. Hierarchical dirichlet processes. *Journal of the american statistical association* 101, 476 (2006), 1566–1581.
[33] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. 2021. Unsupervised Representation Learning for Time Series with Temporal Neighborhood Coding. arXiv:2106.00750 [cs.LG]
[34] Aaron Van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv e-prints* (2018), arXiv–1807.
[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., New York, NY.
[36] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.
[37] Chengyu Wang, Kui Wu, Tongqing Zhou, Guang Yu, and Zhiping Cai. 2022. TSAGen: Synthetic Time Series Generation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* 19, 1 (2022), 130–145. https:

//doi.org/10.1109/TNSM.2021.3098784

[38] Makoto Yamada, Akisato Kimura, Futoshi Naya, and Hiroshi Sawada. 2013. Change-point detection with feature selection in high-dimensional time-series data. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*. International Joint Conferences on Artificial Intelligence Organization.

[39] Lina Yao, Quan Z. Sheng, Wenjie Ruan, Xue Li, Sen Wang, and Zhi Yang. 2015. Unobtrusive Posture Recognition via Online Learning of Multi-dimensional RFID Received Signal Strength. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, New York, NY, USA, 116–123. https://doi.org/10.1109/ICPADS.2015.23

[40] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. 2022. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8980–8987.

[41] Nengwen Zhao, Jing Zhu, Rong Liu, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Label-Less: A Semi-Automatic Labelling Tool for KPI Anomalies. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, New York, NY, USA, 1882–1890. https://doi.org/10.1109/INFOCOM.2019.8737429

Received April 2022; revised July 2022; accepted August 2022